Techniques for the Combination of Multiple Predictors Applied to Recommender Systems

Michael Jahrer

Techniques for the Combination of Multiple Predictors Applied to Recommender Systems

Master's Thesis

at

Graz University of Technology

submitted by

Michael Jahrer

Institute for Theoretical Computer Science (IGI),
Inffeldgasse 16b/1
Graz University of Technology
A-8010 Graz, Austria

1 th December 2009

Advisor: Univ.-Ass. DI. Dr. Robert Albin Legenstein

Techniken für die Kombination von mehreren Prediktoren angewendet auf Empfehlungssysteme

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Michael Jahrer

Institut für Grundlagen der Informationsverarbeitung (IGI),
Inffeldgasse 16b/1
Technische Universität Graz
A-8010 Graz

1. Dezember 2009

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Univ.-Ass. DI. Dr. Robert Albin Legenstein

Abstract

Learning from data, in order to make predictions, is an important problem in computer science. To tackle this problem, many learning algorithms have been developed. It is well-known that prediction performance can be improved by combining several individual predictions that are based on distinct learning processes. Such combination techniques are called ensemble methods. Two widely used ensemble techniques are bagging and boosting. Different predictions are obtained by either resampling the training set (bagging) or laying focus on hard examples (boosting). Combining different kinds of learning algorithms (e.g. linear regression, k-nearest neighbors, neural networks, kernel ridge regression) is not the aim of bagging and boosting. For this problem, known methods are stacking and cascade learning. Furthermore, we analyze residual training as a combination algorithm. This thesis consists of an exhaustive experimental evaluation of different combination and training algorithms for ensemble learning. In the experimental section, we evaluate ensemble learning techniques on 14 UCI datasets and two larger datasets (UCI Letter, MNIST). The application of ensemble learning to recommender systems is analyzed with the Netflix dataset, that consists of 10^8 movie ratings samples. We found that simple linear combination of predictions is not optimal in with respect to the minimization of the prediction root mean squared error (RMSE). To predict ratings with collaborative filtering we use a set of predictions from different models (SVD, KNN, Restricted Boltzmann machine, Asymmetric Factor model, Global Effects) from the participation on the Netflix prize. These models are state of the art in the field of collaborative filtering. To perform all reported experiments, a C++ based framework for ensemble learning was developed. We provide a description of this framework in the last section of this thesis.

Kurzfassung

Das Lernen aus Daten, um Vorhersagen treffen zu können, ist ein wichtiges Problem der Informatik, und viele Algorithmen wurden entwickelt um Prediktoren zu trainieren. Es ist bekannt, dass die Vorhersagegenauigkeit verbessert werden kann, wenn man die Vorhersagen einzelner Prediktoren kombiniert. Methoden zur Kombinationen von Prediktoren werden Ensemble Methoden genannt. Zwei weit verbreitete Techniken sind bagging und boosting. Bagging basiert auf dem Resampling einen Datensatzes. Boosting fokusiert sich sukzessive auf die schwer zu erlernenden Beispiele. Die Kombination von verschiedenen Lernalgorithmen (z.B. lineare Regression, k-nearest neighbors, Neuronale Netzwerke oder Kernel Ridge Regression) ist nicht das Ziel von bagging oder boosting. Für diese Art der Probleme eignen sich stacking und cascade learning. Weiters analysieren wir das Trainieren auf dem verbleibenden Modellfehler (Residuals). Diese Diplomarbeit fokusiert sich auf eine experimentelle Auswertung verschiedenster Kombinationen von einzelnen Lernalrogithmen mit verschiedenen Ensemble Lernmethoden. Dabei führen wir eine experimentelle Auswertung auf UCI Datensätzen aus. Weiters werden Ensemble Lernmethoden auf den Datensätzen UCI Letter und MNIST analysiert. Die Anwendung auf dem Gebiet der Empfehlungssysteme wird mit Hilfe des Netflix-Datensatz ermittelt. Dieser hat eine Größe von 10^8 Ratings. Das Ensemble von Vorhersagen liefern verschiedene Collaborative Filtering Algorithmen (SVD, KNN, Restricted Boltzmann Maschine, Asymmetric Factor Modell, Globale Effekte). Ein häufig verwendete einfache Methode der Kombination von Vorhersagen ist lineare Regression. Wir zeigen dass lineare Regression nicht optimal ist wenn auf das Fehlermaßroot mean squared error (RMSE) optimiert wird. Im Zuge der Diplomarbeit wurde ein auf C++ basierendes Software Framework entwickelt. Das letzte Kapitel beschreibt das Framework in groben Zügen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A machine learning algorithm learns a hidden function from data. We can aggregate multiple results from various learners. This thesis is about ensemble learning, where the results of different learning algorithms are combined in a clever way.

Theory and practice in supervised learning has a long history, and many learning algorithms were proposed in the literature. A learning algorithm is a technique for learning a mapping from input space to its output space from a given dataset. Input and output space are typically high dimensional, the task of a learner is to approximate the mapping function Ω in order to get low error on unknown test examples (unseen examples). This means the algorithm must generalize to new data points. If input and output space is high dimensional the problem of sparseness is a big issue.

The application of different algorithms on the same problem leads to different approximations of the mapping function Ω. This behavior can be used to construct a better approximation so that the error on the test set becomes lower than the error of any individual algorithm.

Generalization is a major concern in all supervised models. There is always a tradeoff how well the model can fit the data, also known as bias-variance tradeoff. If the fit is too accurate it may not perform well on new data. If the model is not powerful enough to extract the underlying function, the error on unknown data is also high.

1.1 Ensemble of Models

An "ensemble" is a set of trained models which are combined in a clever way [1]. The goal of ensemble learning is to aggregate learned models in order to improve the prediction accuracy on unknown input data (generalization). The success of combining multiple models depends on the error of each individual and the correlation between them [2]. In a good ensemble, the pool of prediction models are accurate individually and learn different structure from the data to add valuable information. So in practice, how well an ensemble performs is data dependent. For example if a neuronal network has enough expressiveness to learn the underlying function of the given problem well enough, there is less or no room for improvement (one of the experiences based on our experiments). Another point why ensembles can fail is due to the small sample size of the training data. Consider the case of only a few hundreds available train examples. Then the combination of models can lead to overfitting (another experience based on our experiments). But in general if the combination schema is applied correctly, the ensemble's prediction is better than each individual in it.

To measure accuracy, the following two error measures are widely used, the first one is the classification error or accuracy. We use this to measure the performance on classification tasks. This is simply the percentage of wrongly classified examples. There are N samples in total and n_{false} is the number of wrong classified samples.

$$E_{crate}[\%] = \frac{n_{false}}{N} \cdot 100\% \tag{1.1}$$

The second one is the normalized form of the squared error, the RMSE or root mean square error. It has a smoother behavior compared to the classification error. The classification error has relatively large jumps, the number of steps depends on number of samples and number of targets. Later, when we discuss optimization, it has several advantages to optimize directly the RMSE of the ensemble. This is because the RMSE is more like a probablistic error measure when we train a classification problem.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2} \tag{1.2}$$

Where the prediction values are denoted by p_i and the targets values by y_i. This measure is used in the Netflix Prize [4] to rank the participant's submissions.

1.2 Ways of Constructing an Ensemble of Models

The following ideas are a loose summary of techniques for building an ensemble of predictions. In other words, how to build a library of diverse predictors as input for an ensemble learning method/combiner. Most of the concepts focus on ending up in a fairly accurate set of diverse models.

Different learners

The construction of an ensemble with different types of models is a very promising and popular approach [11]. We use this kind of ensembling in the evaluation Section 5. Recommender systems achieve great accuracy when operating with an ensemble of different learners [24], [29], [42], see Section 6 for details. The motivation behind is that different learners (like k-nearest neighbors or neural networks) achieve good accuracy while the predictions are less correlated. Both properties are needed in order to gain performance with ensembling.

Modify the Trainings Tet

Train the algorithm multiple times on a modified version of the training set. This leads to different prediction behavior when predicting new data. The idea is to reduce the variance in the model by taking the average prediction over multiple training runs with slightly different data. When doing sampling with replacement, this process is called "Bagging" [6]. One can show that this can improve the prediction accuracy a lot if the predictions are uncorrelated. Models with high prediction variance are needed in order to apply bagging successfully. Decision trees are ideal candidates for bagging.

There is another approach to re-sample the training data without replacement, namely "Dagging" [40]. Here, the training sets are disjoint subsets, hence it is only applicable when a lot of samples are available.

Modify the Sample Weights

In most data sets, the training error of an algorithm per example is not equally large. The training error is an indicator of how good the model can fit to the data, low error means good fit. Some examples are easy to learn, while others are hard. This can come from noise in the data set or the algorithm's model is not powerful enough to capture all the variance in the data. An interesting idea is to give each example i in the training list a weight w_i. In the first round each example has $w_i = 1$. Now, run the algorithm. In the second round modify the sample weights, examples with large training error receive a higher weight, while examples with low error get lower weights. This procedure is used in “Boosting” to construct an ensemble of predictors, each predictor focus on examples, where the learners in previous rounds are not good at. In Section 4.3 we present Adaboost.M2 [34] as a train list sampling approach to boosting. This has the advantage that the learn algorithms itself needs no modification. The standard Adaboost algorithm [13] assumes that the learner can handle weighted train samples.

Randomness

Some models extract latent parameters from data (the internal weights), for example neural networks or decision trees. Often, the optimal parameters can not be calculated in a closed form. The error surface contains many hills and valleys with a lot of local minima. Therefore an initial set of weights is chosen and the parameters are optimized via gradient descent. For this reason, the initial parameter set has an important impact on the final parameters. This can be used to construct an ensemble, just by choosing different initial parameters. The success of this method is known to be low because the predictors are highly correlated [10].

Noise

Adding noise to the training data $\mathbf{X}$ leads to diversity in the model, this can be used to build the ensemble. Similar to the previous paragraph, the outcomes are highly correlated and therefore less successful in combination. Adding L2-regularization to the error function is equivalent to add stochastic Gaussian noise to the input samples.

Different Accuracy on Different Regions in Input Space

The prediction accuracy of prediction models can be dependent of where the input example $\mathbf{x_i}$ is located in input space. If we have enough data, a decision tree can be used to split the input space and assign a different model as predictor on a particular region of input space. A small neural net, called gating network is used in [23]. This gating net is trained simultaneously with the ensemble to “gate through” different models on different inputs. The largest problem of this approach is to handle high dimensional input vectors.

Force Diversity

Most models optimize a quadratic error function. It is possible to add a penalty term to the error function for equal prediction values compared to existing ones in the ensemble. So this can be used to force the diversity of the ensemble by constructing a model, which should be as different as possible and still have low error. This technique is known as “negative correlation learning” and is well described in the PhD thesis of Brown [10]. This approach can be applied easily to every gradient decent based training algorithm.

Repeat the Experiment where the Data came from

If we have the ability to repeating the machinery, which generates our training data, there is an easy way to get a good ensemble by train models on different collected subsets of training data. For example generate the data on different days and average over many models.

Vary Model Architecture

By varying the architecture of a neural network, one can expect different prediction behavior, this can be used to construct an ensemble [10]. Different distance functions in a k-nearest neighbor algorithm can lead to different predictions (Pearson vs. euclidean). An ensemble is constructed by vary different model parameters, add regularization or change the error function.

Vary Model Parameters

Many algorithms such as linear regression, k-nearest neighbors or kernel ridge regression have strong meta-parameters. The regularization constant is important to control the model complexity, kernel hyperparameters have a large impact on the model. By varying this meta-parameters an ensemble of many predictors can be constructed. This technique is used by Caruana et.al. in combination with different models in “Ensemble Selection” [11] to generate a large library of models.

Artificial Training Examples

Sampling new data from the training distribution can lead to an increase in robustness. In [28] significant improvement is shown when little data is available. They report results on UCI datasets when only a fraction of the data is used for training the model. It can be shown that artificial training examples in an OCR (optical character recognition) data set can help to improve the generalization ability of a learner. There are various reports, that people augmented the MNIST data set with duplicates, generated by elastic distortions of training samples. This special artificial training samples incorporate knowledge to the problem by known geometric invariance.

1.3 Organization of this Diploma Thesis

The content of the thesis is the experimental evaluation of ensemble learning techniques. We use 14 smaller UCI data sets, the UCI LETTER, the MNIST data set of handwritten digits and 18 predictors from the Netflix Prize challenge. The results show that a well constructed ensemble of models always outperform the best single individual.

In Chapter 1.2 we motivate possible ways of constructing an ensemble. One of the most promising approaches is to combine results from many different models. This is demonstated it in many experiments. Chapter 2 gives an introduction to supervised learning. We explain what generalization means and how to control the learning process in order to prevent overfitting. Chapter 3 consists of a description of all base learners we use. They are explained by formulas and an algorithm block of pseudo code, which sketches how each model is trained. We describe how the training data is normalized and how the algorithm dependent parameters are optimized. Chapter 4 gives an overview of ensemble learning methods and how they are applied in order to combine learners. Chapter 5 describes the setup and the outcomes of the experimental evaluation on 14 smaller UCI data sets. We report significantly wins of ensemble methods over single learners in 6 of 14 data sets. Furthermore the results on LETTER and MNIST are reported here. Chapter 6 demonstrates the abilities of ensemble learning when applied on recommender systems. The improvement is impressive compared to the best individual predictor. We use the Netflix Prize data set and investigate different blending techniques. The outcome is that blending predictors with a neural network works best. Chapter 7 gives an introduction of the developed sofware for the ensemble learning. It is a C++ based framework where the focus is laid on training speed. The last Chapter 8 consists of discussions of the results and possible extentions.

Chapter 2

Supervised Learning

The concept of supervised learning comes from the supervisor, acting as a teacher in the learning process. In other words, the teacher says during training how the output of the learner should be. Supervised learning is besides unsupervised and reinforcement learning part of the scientific discipline machine learning. C.M.Bishop [5] and T.Hastie et.al. [18] are good introductory books to the topic. The first is more related to a probabilistic view to the machine learning topic, the second one has a strong statistics viewpoint.

2.1 Definition

The goal of supervised learning is to build a learning system that models a target function. Training is performed on a list of tuples, each tuple consists of an input and a target vector. Targets can also be class labels, in the case of a classification problem. The list of tuples is called training set. A good learner predicts targets as accurate as possible, for any given input vector. This property is called generalization (generalize to new data).

Training data is essential in supervised learning. It can come from different sources like artificial data, sensory data, databases, human generated data etc. Each dataset has its own underlying function, which the learner is trying to model. The error of the moel is the difference between the true and the predicted target. The proper error measure is problem or data set dependent, it gives a real-valued number of the performance of the learner on a test set. Our goal is to minimize the prediction error on the test set.

There are two related problem types in supervised learning, the first is regression and the second classification. For example the “Boston Housing” dataset from the UCI data set repository [1] is a regression problem. The goal is to predict the crime rate (real-valued) based on a number of features like pupil-teacher ratio, accessibility to highways or nitric oxides concentration. A popular classification data set is the MNIST [2], the aim is to predict the label of handwritten digits (0...9). The input vectors are 784-dimensional (28x28 pixel grayscale image) and the targets are labels.

The error measure is typically different in context of supervised regression or classification. In regression it is usual to take root mean square error (RMSE), mean square error (MSE) or mean absolute error (MAE) as the error to be optimized. In classification problems it is more convenient to use classification error (accuracy) or area under the ROC curve curve (AUC, for binary problems). For more than two target classes it is common to use a confusion matrix to visualize the false classified examples. The rows of the square confusion matrix are the predicted labels, the columns are the real labels. The number of diagonal elements is equivalent to the accuracy.

[1]http://archive.ics.uci.edu/datasets/Housing
[2]http://yann.lecun.com/exdb/mnist

Furthermore supervised learning can be categorized into offline and online learning. In offline learning, there is a data set available where the learner can build the internal model without any limits in accessing the data. For example, the Netflix Prize [4] is such an offline data set. It is a movie rating data set with 100M samples collected in the time from 1998 to 2005. The competitors have time to carefully analyze the dataset, build large predictive models and combine them in a sophisticated way. This is the sketch of the winner solution. The solution was provided approximately three years after start of the competition. In contrast to offline learning, online learning has access to a sample only once. The goal is the same, predicting targets as accurate as possible. For example, stock market prediction can be seen as online learning. The algorithm makes a prediction of the stock, little time after the real stock price is available, this information can be incorporated to the learner to further improve the prediction accuracy. In general, there is very much data availaible in an online learning setup, the data set grows continuously. Offline learning has equal or superior accuracy compared to online learning when the same amount of data is used. All experiments in this paper are done in an offline learning fashion.

Input features are also called input vectors or attribute vectors, the targets are sometimes called outputs or desired values. The type of features play a crucial role when do supervised learning. For most data sets, one real-value is a single feature. The number of available features assemble the feature vector. Each training example is a tuple of a feature vector and the corresponding target vector. A single feature can have “the categorical” type, which means only a discrete number of values are allowed. For example the feature “State” can have the values {“Nevada”, “Washington”, “Utah”,...}, or “Sex” can have the values {“male”, “female”}. There is a third type of value, namely “missing”. Such missing features are a problem for most machine learning algorithms. For example decision trees can work with missing feature natively. A possible work around for missing features is to replace missing values with the mean value (or the median) in the training set. This approximation works well in most cases, we also use mean feature values to replace missing features.

To be more general, the supervised learning problem can be formulated as function approximation from an input to an output space. A classical trainable model for function approximation are neural networks. Usually, the input dimension is much larger than the output dimension. When we look to the supervised learning problems from the UCI machine learning repository [3], almost all data sets have more training examples than the number of features. A feature is one dimension of the input space. This is the standard case when doing learning from data. There are a small number of exceptions, mainly in the domain of biochemistry, where many features and less observations are available. Such data sets consist of for example genomes, DNA microarrays, drug discovery or mass-spectrometric data. In the last chapter of the book from T.Hastie et.al. [18], there is a good introduction to such high-dimensional problems.

2.2 Formal Description of Learning Problems

Bold faced symbols ($\mathbf{X}$,$\mathbf{Y}$,$\mathbf{x}$...) are used for vectors and matrices, non-bold for scalars. In Figure 2.1 there is an overview of the data matrices used in this paper. Throughout the thesis we use a multiple feature/input and multiple target/output notation, which is dedicated to cover the high dimensional input and output space. The data matrix is $\mathbf{X}$, a NxF matrix with elements x_{ij}, where N is the number of training examples and F is the size of the input (input dimensionality). x_{ij} is the j-th feature of sample i. The rows of $\mathbf{X}$ are the input features. Target values y_{ij} are denoted as matrix $\mathbf{Y}$ of size NxT, where T is the number of target values (output dimensionality). y_{ij} is the j-th target value of sample i. Each single target is a row in $\mathbf{Y}$. Predictions p_{ij} are denoted as matrix $\mathbf{P}$ of size NxT. Any learner uses $\mathbf{X}$ and $\mathbf{Y}$ to build the internal predictive model. A single prediction is a row in $\mathbf{P}$. The prediction model is $\Omega : \mathbb{R}^F \mapsto \mathbb{R}^T$, which represents the function approximation from input to output space. When working with classification problems, C is the number of classes, the corresponding encoded target vectors $\mathbf{y}_i$ are

[3]http://archive.ics.uci.edu/ml/index.html

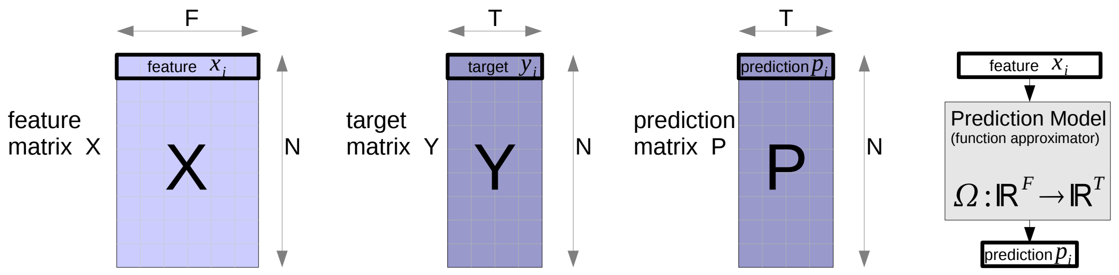


Figure 2.1: Mathematical notation of the supervised learning problem, used in this paper. A learner Ω appoximates the real-valued function $\Omega : \mathbb{R}^F \mapsto \mathbb{R}^T$. The train data is the matrix $\mathbf{X}$, the corresponding targets are $\mathbf{Y}$. Both matrices has the same number of N rows. The matrix $\mathbf{P}$ consists of predictions.

C-dimensional with labels l_i. We use the value of $+1$ to encode class labels and -1 for other values in $\mathbf{y}_i$.

When we view the data in form of a list, we use $\mathcal{L} = \{(\mathbf{x}_1, \mathbf{y}_1)...(\mathbf{x}_N, \mathbf{y}_N)\}$ as notation for the training set. Each sample is a tuple of the feature vector and the target. This is typical for gradient descent based algorithms, where the training is done epoch-wise, in one particular epoch example-wise.

2.3 Curse of Dimensionality

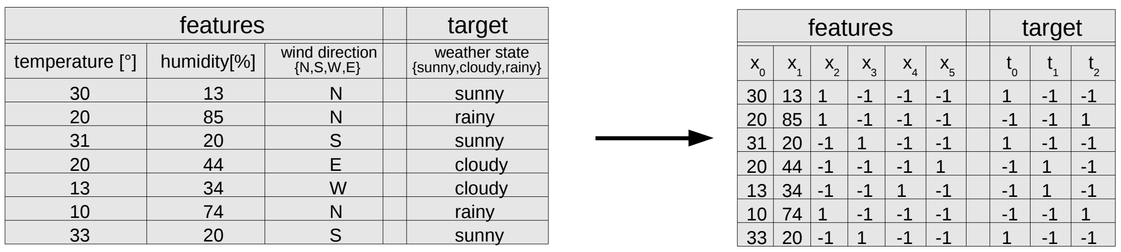
The curse of dimensionality means that it is difficult to work with high dimensional data. This concerns input features. Consider the case of an one-dimensional input and 10 training samples. When we go to two inputs, 100 training samples are required to sample the input space equally. The previous mentioned dataset “Boston Housing” has 14 features, which would result in an equivalent of 10^{14} training samples. We see that real world data do not have enough observations to get a dense sampling of the input space. Targets from almost all the input space are not unknown. Therefore it is hard for a learner to build a reliable model based on the observations.

Dimensionality reduction is in some cases useful. There exist several techniques, an easy approach is feature selection, where only a subset of the features is used to train the supervised learning model. More advances techniques in dimensionality reductions are PCA (project to subspace with most variance), KernelPCA (non-linear version of PCA) and Autoencoder (multi-layer neural network, that is trained to reproduce the output by pass the data through an bottleneck - the code layer). Less dimensions makes the handling of the data much easier, furthermore it reduce the storage space.

Occam’s Razor theory of building models by using the minimum number of parameters is very effective when handling high dimensional data sets. When two different learners with equal performance have a different number of free parameters, choose the one with less parameters.

2.4 Encoding of inputs and outputs

During this thesis we use one-out-of-n encoding for categorical features and categorical targets. This lead us to an simplified view of any supervised learning problem and ensemble learning strategies. Categorical inputs are encoded by $+1$ and -1 values, we use the number of categories for this particular feature as replacement for real-valued inputs. The Figure 2.2 shows a sketch of such an encoding on an artificial classification data set. The first two features have numeric values, the third one is categorical. The target labels are categorical and encoded in the same way. Combining predictions with the ensemble learning technique ”stacking” (explained in Section 4.4) is solving a special linear regression problem.



features				target
temperature [°]	humidity[%]	wind direction {N,S,W,E}		weather state {sunny,cloudy,rainy}
30	13	N		sunny
20	85	N		rainy
31	20	S		sunny
20	44	E		cloudy
13	34	W		cloudy
10	74	N		rainy
33	20	S		sunny

features							target		
x_0	x_1	x_2	x_3	x_4	x_5		t_0	t_1	t_2
30	13	1	-1	-1	-1		1	-1	-1
20	85	1	-1	-1	-1		-1	-1	1
31	20	-1	1	-1	-1		1	-1	-1
20	44	-1	-1	-1	1		-1	1	-1
13	34	-1	-1	1	-1		-1	1	-1
10	74	1	-1	-1	-1		-1	-1	1
33	20	-1	1	-1	-1		1	-1	-1

Figure 2.2: Transformation of a classification problem with numerical and categorical inputs to a regression problem with numeric input features only. This transformation turns any supervised learning problem to this unified view.

The technique of blend-stopping (explained in Section 4.7) is a directly minimzation of the linear combination, hence the requirement is that the computation time should be low. The predicted values are considered a likelihood vector, this enables us to successfully apply linear regression as a combiner.

The following example shows how a classification problem can be transfered to a multi-target regression problem by 1-of-n target encoding. Assume we have C possible class labels. The training sample i has the label l_i and the corresponding target vector is $\mathbf{y}_i \in \{-1, 1\}^C$. For instance, when we have three classes $C = 3$ and the target is the second class, the target vector would result in $\mathbf{y}_i = (-1, +1, -1)$.

$$\mathbf{y}_i = (y_{i1}, ..., y_{iC}), \text{ where } y_{il_i} = 1 \;, y_{ij} = -1 \text{ for } j \neq l_i \tag{2.1}$$

The algorithm predicts $\mathbf{p}_i \in \mathbb{R}^C$, the predicted class is given by the maximum response, $l_i = \text{argmax}_{j=1...C}\, p_{ij}$.

2.5 Overfitting and Underfitting

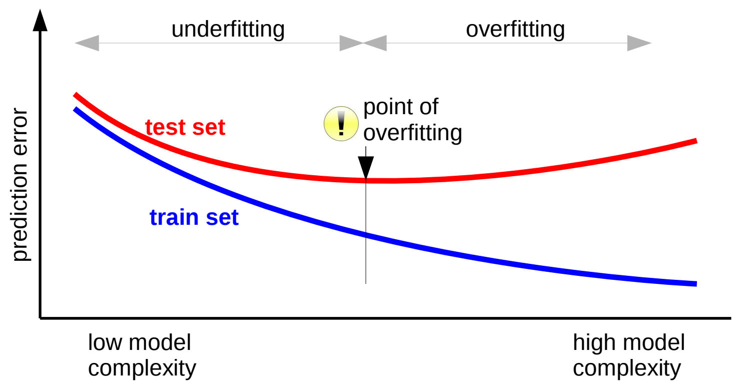


Figure 2.3: The model complexity influences the train and the test error. There are two regions, the first is known as underfitting, where the learner in not strong enough to capture the train data. This results in high train and test error. Overfitting is the well known phenomenon of memorizing the train set, this lead to poor performance on test samples.

The term “overfitting” implies that the performance on test set is much lower than on training set. In contrast, “underfitting” means that the model can not learn the data well enough, the learner is too weak.

Both, the performance on the train and the test set is low. In the area of overfitting, the learner starts to capture all the noise in the data set and not the real function behind. The predictive error on unknown test samples rises. Any learned model should generalize well on new data.

In the machine learning literature, the bias and variance analysis are often used to analyze the expected prediction error of a learner. The bias denotes the accuracy of the learner, low bias is desired. The variance is the spread of the error distribution on a set of test samples, low variance reduces the error. High learning bias occurs, when the model can not identify the underlying data structures, it is not powerful enough or has too less free parameters. In the Figure 2.3 we vary the model complexity. The left side called underfitting region, this results in high bias and low variance. The right side has low bias and high variance, which is typical for overfitting the data. The point of overfitting is be best tradeoff between low bias and low variance.

2.6 Controlling Overfitting - Selection of Good Meta Parameters

The classic approach in supervised learning is to have a train and a test set. There two sets should have the same statistical properties. The test set should not be used for training and tuning of the algorithms. The simplest apporach is to split a validation set from the train set, the size is typically 20% and do performance measuring all tuning on this set.

It is common to use powerful learning models, such as neural networks, support vector machines or decision trees to capture different aspects in the data. For small data sets, the learner is able to achieve nearly zero training error. This is the tail of the blue line in Figure 2.3, overfitting the data is clearly a problem here.

Let us start with training data and a learner, a reliable measure of the true accuracy (accuracy on new test samples) is not an easy task. For example, we want to know during training when we should stop learning. Error on the train set should go down when the model complexity increases. Simple models, for example linear regression, are known as they can not fit the data well enough (underfitting). One possible way of measure the accuracy is to split a validation set $\mathcal{L}^V$ from the training data $\mathcal{L}$ and do not train on the validation set. Predictions on $\mathcal{L}^V$ are now a good indicator of the real performance of the learner. For example in the Netflix Prize [4], the train list $\mathcal{L}$ has a size of 100M. Within this data set a validation set of 1.4M samples, called "probe set", is also available. This validation set is a representative set for measuring the real error. It comes from the same distribution as the test set. Because of the huge size in this example, the error on 1.4M data points is very reliable to determine when the model begins to overfit the data. For smaller data sets, it can be shown that splitting a random validation set is not reliable enough to estimate the true error. A typical size of a validation set is 10% to 50% of all available data.

We can virtually enlarge the validation set by splitting the training list $\mathcal{L}$ into k equal sized sets $\mathcal{L}^{(-k)}$ and train k copies of the model (with same initial conditions) on the remaining data. This is called "k-fold cross validation" and is well known in the scientific machine learning literature. We use k-fold cross validation during training to determine what parameter set yields best performance.

Early stopping is well known for gradient descent based algorithms to stop training, when the error on a validation set becomes minimal. We use this in combination with k-fold cross validation for example in neural networks and gradient boosted decision trees. Furthermore, techniques such as pruning or weight sharing (for example in convolutional neuronal networks) can be used to fight against overfitting. One of the most popular methods is regularization. The basic idea is to penalize large weights during training. $L2$-regularization for example adds a quadratic term with respect to the weights to the error function.

The technique of k-fold cross validation is not only useful in training a single algorithm. Also for ensemble learning, where the prediction of the training set is needed. The extreme case of k-fold cross validation is to use $k = N$ (N...number of training samples), which is also called leave-one-out cross validation or LOOCV. This is computational very expensive, in particular when we have many training examples.

An alternative to cross validation is bagging. Bagging constists of many boostrap replicates of the training set, the model is applied to each of them. A boostrap sample is an equal sized training set, drawn from the original data with replacement. This means that some training samples are not in the new training set, the procedure is also called bootstrap sampling. The non-drawn samples act now as validation set. Since bagging works with many such bootstrap samples, the out-of-bag estimate (mean over validation samples) is a good measurement of the real accuracy.

Chapter 3

Basic Regression Models

In this section we list and describe the details of the used regression models. The first linear lodel is the simplest of them, neural networks are generic function approximators and performs well on many applications. K-nearest neighbors is a memory based algorithm and the prediction is based on a weighted sum of nearest neighbors. Kernel ridge regression is a very powerful kernelized linear regression technique, the big disadvantage is that it requires O(N^2) memory, where N is the number of training examples. Last, we introduce from the tree algorithm family the gradient boosted decision tree, which combines a lot of clever ideas to improve accuracy. The following boxes summarize important ideas, we use during the evaluation.

The metaparameters of all algorithms are tuned in order to optimize the squared error of all targets T on all N samples: $E = \sum_{n=1}^{N} \sum_{t=1}^{T} (p_{nt} - t_{nt})^2$ on a k-fold cross validation set.

On classification problems, the predicted class label is selected by taking the index of the maximum output value: $l_i = \textbf{argmax}_j \; p_{ij}$

We always normalize the training input data (feature-wise) to have zero mean and standard deviation of 1. Other normalization is denoted explicitly.

3.1 Data Normalization

Many machine learning models need proper data normalization, for example neural networks need well normalized data in order to find a good local minima by gradient descent. K-nearest neighbors algorithm rely on a distance measure in feature space. In the case of unnormalized data, dimensions with high variance would dominate the euclidean distance. For this reasons, we use a simple and widely applied method of normalization. Every input dimension is normalized to have mean value of 0 and standard deviation equals 1. Raw numeric values from the dataset are denoted by $\mathbf{X}^{orig}$. The features we use to train the models and the ensembles are saved in the matrix $\mathbf{X}$.

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{ij}^{orig} \quad \text{: mean of the feature } j \text{ in the training data} \tag{3.1}$$

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{ij}^{orig} - \mu_j)^2} \quad \text{: standard deviation of the feature } j \text{ in the training data} \tag{3.2}$$

$$x_{ij} = \frac{x_{ij}^{orig} - \mu_j}{\sigma_j} \text{ : data normalization (centering and scaling), } i = 1...N, j = 1...F \tag{3.3}$$

Mean value and standard deviation per input dimension is denoted by μ_j and σ_j. Please note that σ_j can have values near to 0, if input dimensions have very small or constant values. In order to ensure stable normalization behaviour, we limit the value to $\sigma_j \geq 0.01$.

3.2 The MSE Error Function and L2 Regularization

The performance measure we use for cross validation of all models the mean squared error (MSE) or the square root of the MSE, the RMSE. This implies a quadratic error function to be optimized. Furthermore the MSE is a smooth function. On the other hand, directly optimizing a misclassification rate can be hazardous because it is has large steps in learning curve and optimizers are prone to get stuck in a first local minima.

To control large model parameters, we use L2 regularization whenever possible. The tradeoff between MSE and this penalty is given by the regularization constant λ. The global error function has the following form.

$$E = E(\mathbf{Y}, \mathbf{P}) + \lambda \cdot \sum_{i=1}^{\text{all parameters}} |w_i|^d \tag{3.4}$$

The first term $E(\mathbf{Y}, \mathbf{P})$ is the error function over all examples as a function of the targets $\mathbf{Y}$ and the corresponding predictions $\mathbf{P}$. A quadratic error results in $E(\mathbf{Y}, \mathbf{P}) = \sum_{i=1}^{N} \sum_{j=1}^{T} (y_{ij} - p_{ij})^2$.The second term is responsible to penalize weights, this acts as a regularizer to control the model complexity. Large values of the constant λ reduce the ability to learn the data. The exponent d sets the type of the regularization. L2 regularization uses $d = 2$. The following values of d has been used by several applications and have established names in the literature.

- $d = 0$: L0, penalizes the number of non-zero parameters, related to MDL (min. description length)
- $d = 1$: L1, also called Lasso, leads to sparse models, some parameters are exactly 0
- $d = 2$: L2, quadratic penalty, leads to many small weights, ridge regression in linear model

The math in deriving learning rules for algorithms is easiest when using L2 regularization. For example the linear regression technique has the nice property of having a closed solution when L2 regularization is used.

3.3 Linear Regression - LR

A linear regression model predicts new input samples by a linear combination of input features with weights $\mathbf{W}$. The weights are denoted by a FxT matrix, so new inputs $\mathbf{x}$ (row vectors) are predicted by

$$\mathbf{p} = \mathbf{x} \cdot \mathbf{W} \tag{3.5}$$

Estimation of the weights can be done by gradient descent, but a linear solver is more efficient. Linear combination weights can be obtained by evaluating the following expression. The matrix $\mathbf{X}$ is the data set (features) and $\mathbf{Y}$ the target values.

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{3.6}$$

One can penalize large weights by modifying the loss function with L2-regularization of the weights, this is also known as "weight-decay". Following function needs to be minimized with respect to $\mathbf{W}$. When we apply the norm $||\cdot||$ to a matrix, we take the frobenius norm $||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{\#rows}\sum_{j=1}^{\#cols}|a_{ij}|^2}$.

$$\min_{\mathbf{W}} \quad \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|^2 + \lambda\|\mathbf{W}\|^2 \tag{3.7}$$

The next equation is the closed solution to this minimization problem.

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y} \tag{3.8}$$

The identity matrix is denoted by $\mathbf{I}$ and λ is the ridge regression constant (or Tikhonov regularization [39]). The regularizer λ controls the smallest eigenvalues of $\mathbf{X}^T\mathbf{X}$, thus the magnitudes of the weights $\mathbf{W}$ shrinks with higher values of λ. In our software implementation we multiply the constant λ by the number of equations N.

```
Input: Data matrix $\mathbf{X}$, targets $\mathbf{Y}$, new test feature $\mathbf{x}$
Output: The prediction model $\Omega(\mathbf{x})$
Tunable: Ridge regression constant $\lambda$
1 $\mathbf{W} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$
2 $\Omega(x) = \mathbf{x} \cdot \mathbf{W}$
```

Algorithm 1: The LR - "linear regression" learning algorithm

3.4 Polynomial Regression - PR

Polynomial regression is linear regression with a non-linear transformation on the input features. This means the training features $\mathbf{x}$ are enlarged by the help of a binomial series $(1 + \mathbf{x})^\alpha$.

The number of terms, in the case of adding all cross interactions to augment the feature set is $F \cdot (F + 1)/2$ when having a degree-2 polynomial. It is clear that this only works for a small number of features. In the evaluation we are going to use the shortcut "poly2" for a polynomial regression with degree 2 and no cross-terms. "poly2-cross" is the same as poly2 but with all cross interaction terms.

3.5 Artificial Neural Networks - NN

Artificial neural networks are functions approximators, based on simple computational units, called neurons. The model was developed in the 70' and is somewhat "brain-inspired" where biological neurons aggregate weighted inputs to produce a non-linear output. In biology, neural nets use spikes (small electric impulses), artificial neural networks operate in the real-valued domain. The net is built of single neurons, connected together via weights. The structure can be defined by the designer, but in general it is layer oriented. This enables a more structured view on the arrangement of neurons and a net structure can be fully described by the number of hidden layers and the number of neurons in each layer. Learning is typically done with gradient descent [27], [26].

In the past, neural networks have beed used in many applications, such as handwritten digit recognition [26], email spam detection, financial prediction, biometrics, image recognition and feature extraction. They often outperform simple methods like linear regression or k-nearest neighbors methods and

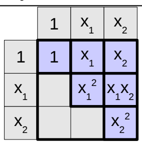
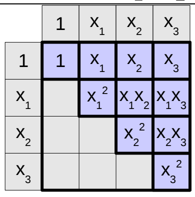
feature $\mathbf{x}$	2nd order (no cross terms)	2nd order (poly2-cross)
$[x_1, x_2]$	$[x_1, x_2, x_1^2, x_2^2]$	 $[1, x_1, x_2, x_1^2, x_2^2, x_1x_2]$
$[x_1, x_2, x_3]$	$[x_1, x_2, x_3, x_1^2, x_2^2, x_3^2]$	 $[1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3]$

Table 3.1: In polynomial regression, additional input features are constructed with the help of a polynomial. The first column shows examples of two and three dimensional input features. In the second column there are poly-2 terms without the interactions between them. The content of the third column shows the construction of poly-2 terms with all cross interactions by an outer product. Terms in the blue squares are the new features.

compare well to SVMs. One of the big advantages of neural networks is that all learned knowledge is encoded in the weights, this makes the prediction very fast.

The internal modeling of the data, found by the weights, are hard to interpret. All learned information, during training a Neural Network, are encoded in its weights. An artificial neural network can model complex interactions within the data and has no problem in handling collinear inputs. The output is bounded by the activation function of the output layer. But it is also possible to place linear output units for prediction.

An alternative usage of a neural networks is the training of an autoencoder (as mentioned in Section 2.3). An autoencoder is a layer-symmetric net (with one coding layer in the middle), the goal is to perform non-linear dimensionality reduction. In this type of network architecture, the inputs should be equal to the outputs, the idea is to reconstruct the output by passing the data through a bottleneck layer. The low-dimensional code in the middle layer can be used as new dimensionality-reduced features. Further informations can be found under [21].

There exists a computational efficient algorithm to train neural networks, the backpropagation algorithm. In Algorithm 2, there is a sketch how the stochastic gradient training is done. We wont go into the details of the backpropagation algorithm because it is well known in machine learning. A good paper for backprop and training neural networks is [27] from LeCun et.al.

When training a classification task, where the targets have values of -1 and $+1$, it is not optimal to place a tanh unit at the output layer. The tanh unit has an output swing of -1 to $+1$, thus it will never approximate the targets with zero error because the targets are placed to the asymtotes of the tanh function. Based on many experiments on the MNIST and UCI data sets we found a simple output mapping, which suggests to muliply the output of the tanh unit with 1.2. This keeps the non-linearity and enables the net to cover the output range.

```
  **Input**: Data matrix $\mathbf{X}$, targets $\mathbf{Y}$, new test feature $\mathbf{x}$
  **Output**: The prediction model $\Omega(\mathbf{x})$
  **Tunable**: Neural architecture based constants (number of neurons, number of layers, learnrate,..)
1 foreach epoch $e = 1..E$ do
2     foreach sample $i = 1..N$ do
3         Forward calculation
4         Backward calculation: store $\Delta\mathbf{W}$ (calculated with backprop)
5         Weight update: $\mathbf{W}^{new} = \mathbf{W} - \eta\Delta\mathbf{W}$
6     end
7 end
8 $\Omega(x) = $ NetEval$(\mathbf{x}, \mathbf{W})$
```

Algorithm 2: The NN - "neural network" learning algorithm

3.6 K-Nearest Neighbors - KNN

Finding k-nearest neighbors in the feature space is one of the oldest machine learning algorithms. The algorithm is instance-based and uses a distance metric (e.g. euclidean distance) to find k-nearest training samples to the one, which is going to be predicted. For regression, the prediction of a new test feature $\mathbf{x}$ is generated by a weighted sum over k-nearest targets $\mathbf{y}$ in the training set. The KNN requires no training, on every new prediction, the algorithm must search through the whole training set to find best neighbors. The algorithm is very strong when distances are well defined. The following formula is used to predict a test feature $\mathbf{x}$. The k-nearest neighbors algorithm is a popular approach in collaborative filtering (decribed in Section 6), the prediction accuracy can be improved by applying a non-linear envelop (e.g. sigmoid function) over the distances $d(\mathbf{x}, \mathbf{x}_k)$, see [42] as reference. We also tried to wrap a sigmoid function around the distances, but we observed no improvement. The prediction vector is given by

$$\mathbf{p} = \frac{\sum_{k\in D} \mathbf{y}_k \cdot d(\mathbf{x}, \mathbf{x}_k)}{\sum_{k\in D} |d(\mathbf{x}, \mathbf{x}_k)|} \tag{3.9}$$

The set D consists of indices of the k-nearest Neighbors to $\mathbf{x}$ in the training samples, $\mathbf{y}_k$ are the corresponding target vectors in the training set. This means we calculate the distances to the input features and select the largest k. The distance metric $d(\mathbf{a}, \mathbf{b})$ is used to compare two feature vectors and find the k-nearest neighbors in the feature space $\mathbb{R}^F$. The whole prediction formula (3.9) is a weighted average of k targets in the training set, hence the name "instance based" learner. Since the distance measure can have negative values (e.g. Pearson correlation), the denominator is the sum of absolute distances.

```
  **Input**: Data matrix $\mathbf{X}$, targets $\mathbf{Y}$, new test feature $\mathbf{x}$
  **Output**: The prediction model $\Omega(\mathbf{x})$
  **Tunable**: The integer constant $k$, to select k-nearest neighbors
1 $\Omega(x) = \frac{\sum_{k\in D} \mathbf{y}_k \cdot d(\mathbf{x}, \mathbf{x}_k)}{\sum_{k\in D} |d(\mathbf{x}, \mathbf{x}_k)|}$, $D$ is the set of indices to the k-nearest neighbors in the train set $\mathbf{X}$
```

Algorithm 3: The KNN - "k-nearest neighbors" learning algorithm

A popular distance metric is the euclidean distance $d_e(\mathbf{a}, \mathbf{b})$, the following formula is the definition of the euclidean distance between two feature vectors $\mathbf{a}$ and $\mathbf{b}$.

$$d_e(\mathbf{a}, \mathbf{b}) = \sqrt{\frac{1}{F} \sum_{i=1}^{F} (a_i - b_i)^2} \tag{3.10}$$

another distance measure is the Pearson correlation coefficient, given by

$$d_p(\mathbf{x}_i, \mathbf{x}_j) = \mathcal{Z}(\mathbf{x}_i)^T \mathcal{Z}(\mathbf{x}_j) \tag{3.11}$$

where $\mathcal{Z}(\mathbf{x})$ denotes the z-score of the vector $\mathbf{x}$. The z-score is calculated by centering the values in $\mathbf{x}$ around its mean and scale it to standard deviation to 1 (the same idea as in Section 3.1). The advantage of the Pearson correlation is that the distance is calculated by a dot product of z-normalized input features. This can be done very efficiently by matrix-matrix multiplication. We show later that the euclidean distance can also be computed efficiently using matrix-matrix multiplication routines.

In order to find the closest match we use the inverse of the euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{d_e(\mathbf{x}_i, \mathbf{x}_j) + 0.01}$ as distance metric, it works well in our experiments. The inverse function is necessary, because we are selecting the k-largest distances in the feature space. The number of 0.01 in the denominator makes the inverse stable for small values of d_e. This forms the set D, the indices to the k-nearest neighbors.

K-nearest neighbors models have quadratic prediction time $O(N^2)$, N is the number of training samples, which is not suitable for generating fast predictions (in the case of a large training set). On the other hand, no training is required.

3.7 Kernel Ridge Regression - KRR

Linear regression can be viewed in a dual representation (Section 6.1 in Bishop [5]). Where standard linear regression needs dot products of input dimensions (columns of data matrix $\mathbf{X}$), kernel ridge regression relies on dot product of input features (rows of data matrix $\mathbf{X}$). If a linear kernel $k(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b}$ is used, KRR is equivalent to linear regression. Recall the prediction of a linear regression is $\mathbf{p} = \mathbf{x} \cdot \mathbf{W}$. Where the linear weights are calculated by $\mathbf{W} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$. Now we can reformulate the prediction formula (Section Kernel Methods, 6.1 Dual Representation, C.Bishop [5])

$$\mathbf{p} = (\mathbf{x} \cdot \mathbf{X}^T) \cdot (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{Y} \tag{3.12}$$

The matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^T$ is called the “gram matrix”, it contains all dot products of input features and is symmetric. Also the first term $\mathbf{x} \cdot \mathbf{A}^T$ are dot products of the new input feature with the training set $\mathbf{A}$. Therefore we can kernelize the dot products by defining a mapping $\phi(\mathbf{x})$ to a high dimensional space

$$k(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) \tag{3.13}$$

We experimented with few different kernels, the following two kernels are most successful. The first is the polynomial kernel $k(\mathbf{a}, \mathbf{b}) = (\alpha \cdot \mathbf{a}^T\mathbf{b} + \beta)^p$ and the gauss kernel (or RBF kernel) $k(\mathbf{a}, \mathbf{b}) = \exp(-\frac{\|\mathbf{a}-\mathbf{b}\|^2}{\sigma^2})$.

```
**Input**: Data matrix $\mathbf{X}$, targets $\mathbf{Y}$, new test feature $\mathbf{x}$
**Output**: The prediction model $\Omega(\mathbf{x})$
**Tunable**: Ridge regression constant $\lambda$, kernel hyperparameters
1 $\mathbf{K} = k^{dot}(\mathbf{X}, \mathbf{X}^T)$, where $k^{dot}(\mathbf{A}, \mathbf{B})$ is the kernelized matrix dot product
2 $\mathbf{W} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}$
3 $\Omega(x) = k^{dot}(\mathbf{x}, \mathbf{X}^T) \cdot \mathbf{W}$
```

Algorithm 4: The KRR - “kernel ridge regression” learning algorithm

In Algorithm 4 you can see the steps required to train the KRR regression algorithm. We use the notation of $k^{dot}(\mathbf{A}, \mathbf{B})$ to denote the kernelized matrix-matrix product. This means that on each point of the target matrix, the dot product gets calculated with the current kernel. The computational and memory intensive step is the inverse of the Gram matrix $(\mathbf{K} + \lambda\mathbf{I})^{-1}$, which time complexity is $O(N^3)$. This limits the number of training samples to around 60000, when using KRR as learn algorithm (assuming single precision an a 16GB machine).

3.8 Gradient Boosted Decision Trees - GBDT

The shortcut “GBDT” is associated with many great ideas. We will explain them in this section. First, this learning algorithm comes from the family of trees. A decision tree consists of nodes and leafs. A single tree has one root node, this is the starting point of the algorithm. When the tree makes a prediction, it compares a certain feature x_i of the feature vector $\mathbf{x}$ with a threshold θ. If the feature is larger than θ, the algorithm continues recursively with the left subtree, else with the right subtree. When a leaf is a termination node, the prediction is a constant value (in most trees). The constant can be the average of the targets from the train samples reching this termination node. A decision tree uses axis-parallel splits to slice and dice the feature space. We can now summarize, a node stores a threshold θ, an index i to the feature x_i and two pointers to a left and a right subtree (again nodes). A leaf stores a constant (the prediction) and is itself a termination node. The challenge of training a decision tree is to find good split values θ and good corresponding single feature indices i. We use a greedy algorithm to train the decision tree, which uses the best split in terms of reducing the RMSE after each split. One tree is trained for every target. Begin with the root node, where the training algorithm has access to all input features. The first split is the one that reduces the RMSE best when predicting two constant values. The next node for the next split is choosen as the one with the most training samples. The process of spliting is stopped when one of the following criterias is met. First, no split improves the RMSE on the training set. Second the maximum number of splits are reached. Or third, when the number of training examples, which are passed through the tree, has a size less than 2 (too less data available).

A tree alone has several disadvantages. For instance, axis-parallel splits can not model a smooth function, the prediction is a constant for some regions of the input space. The tree is known to easily overfit the data. And it is prone to have an unstable behavior, small changes in the train data can lead to total different trees. Breiman has published a useful idea in 2001, the name of the paper was “Random Forests” [8]. The core idea is to limit the number of feature candidates in finding the optimal split per node. Per node, a new random subspace of features is generated. A good empirical value is starting with a random subset of $\sqrt{F}$ features taking in consideration to find the optimal split. No pruning is used here. This modification leads to better generalization behavior, in other words lower error on the test set.

Gradient boosting is an idea from Friedman [15], [16] and has a distinctive relation to training on residuals (see Section 4.6, ensemble learning methods). The algorithm runs epoch-wise, in each epoch the learner is trained on the residuals of the outcomes from all previous learners. In the first epoch the targets $\mathbf{Y}_1$ are the original residuals, the learner’s prediction is $\mathbf{P}_1$. In the second epoch, the targets are $\mathbf{Y}_2 = \mathbf{Y}_1 - \eta\mathbf{P}_1$. In the third epoch, the targets are $\mathbf{Y}_3 = \mathbf{Y}_1 - \eta\mathbf{P}_1 - \eta\mathbf{P}_2$. This means that only a part of the learned model is subtracted from the targets. Hence the name “gradient” because gradient boosting appoaches the target with the gradient of the learner’s prediction. The learning rate η controls how much of the predictions are subtracted by the targets, this mechanism results in learning only a fraction of the data in one epoch. The learning curve can be compared to neural network, where after some epochs the gradient boosted decision tree begins to overfit. This is the point to stop training (early stopping mechanism). Typical values for η are 0.1. The final effect of gradient boosting applied to learning decision trees is to smooth the predictions.

Both ideas, random forests and gradient boosting improves the accuracy and generalization ability of the decision tree as base learner. This has the drawback of longer training and prediction time.

Algorithm	Free parameters	Initial values
linear regression	ridge regression λ	$\lambda = 0.01$
neural network	number of training epochs n	$30 < n < 100$
k-nearest neighbors	k-best neighbors	$k = 4$
kernel ridge regression	ridge regression λ	$\lambda = 0.1$
	gauss kernel width σ	$\sigma = 5.0$

Table 3.2: A listing of tunable parameters and their initial value per regression model. Three of the four models has exact one optimization parameter, the kernel ridge regression with a Gauss kernel has two metaparameters. We optimize the parameters with a simple coordinate search technique for 20 epochs.

3.9 Optimization Process

To compare different ensemble learning methods, each of them must be tuned to the specific data set in order to get a comparative result. As part of the ensemble, every individual algorithm has metaparameters, such as regularization λ. These parameters need to be optimized automatically. Table 3.2 lists the available parameters per model for auto-tuning. Linear regression, neural networks, k-nearest-neighbors and kernel ridge regression are used as basic models in the evaluation on 14 UCI datasets. The algorithms cover different approaches to supervised learning. The linear model learns linear dependencies of input features to output variables. Neural networks have layer-oriented structure and are somewhat brain-inspired. The k-nearest neighbors algorithm is a memory based algorithm, its prediction is a weighted sum of best correlating neighbors of the training set. And the kernel ridge regression algorithm is a powerful kernel method. Figure 3.1 shows an overview how a single algorithm is tuned. We use the cross validation RMSE as feedback of the performance.

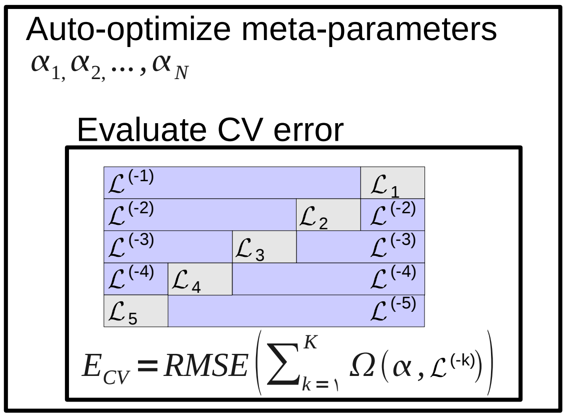


Figure 3.1: A sketch of how metaparameters get optimized. The optimizer (e.g. structured coordinate search) is the outer loop. The goal is to find good metaparameters $\alpha_1, ..., \alpha_N$ in order to minimize the cross validation error (CV error). The CV error is calculated over the leave-out sets $\mathcal{L}_1, ..., \mathcal{L}_K$. On every cross fold, the training of the model is performed on $\mathcal{L}^{(-k)}$, predictions are stored in $\mathcal{L}_k$. E_{CV} denotes the merged RMSE over all cross folds. In other words $\mathcal{L}^{(-k)}$ are the training sets and $\mathcal{L}_k$ are the validation sets.

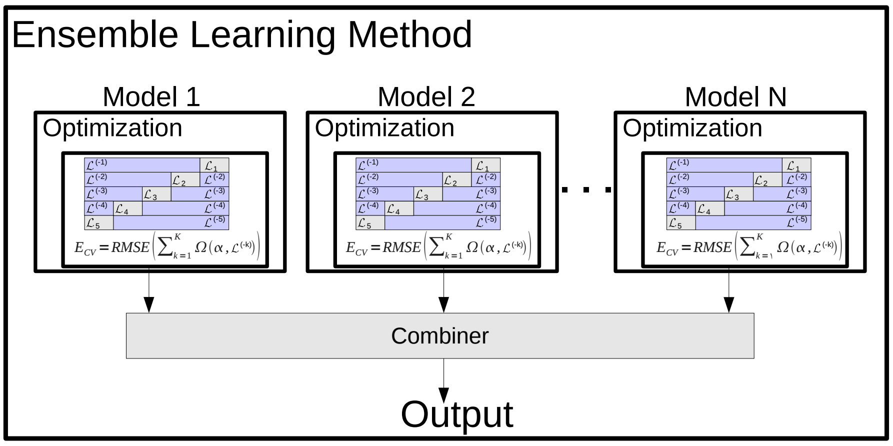


Figure 3.2: Training of the ensemble. Basically, in the process of ensemble learning, each of the basic models are trained and tuned sequentially. The ensemble output is a combination of all basic models. For example, Stacking calculates a linear combination with non-negative weights at the end of tuning each individual training (loose coupling). In contrast, residual training optimizes the linear combination of all algorithms in each optimization epoch. In other words, residual training directly optimizes the whole linear blend. Cascade learning takes the prediction from the last algorithm in the chain.

Cross Validation

To estimate the performance on unknown data, the simplest method is a hold-out set (equivalent to 1-fold cross validation). Averaging over multiple hold-out sets leads to cross validation (CV). The error on the CV is usually a little higher compared to the real error, this comes from the smaller size of the training set. The extreme case is holding back only one example, this leads to leave-one-out cross validation. In our experiments we tried different sizes, from 4-fold-CV to 10-fold-CV. More folds are better, but increases the training time.

The error on the cross validation set is a reliable feedback to search for dataset-dependent meta-parameters. We use this technique throughout all base models, for details see Figure 3.1 and 3.2.

Parameter Tuning

The process of searching for optimal metaparameters (such as regularization in a linear model, the k in a KNN or kernel hyperparameters in KRR) is done without derivates [9]. These searchers are direct search methods, also known under derivative-free optimization. For example genetic algorithms fall in this category. Several algorithms have been proposed, a popular one is the Nelder-Mead Simplex algorithm [25]. Due to high complexity of Nelder-Mead we use a simple coordinate searcher to optimize a few metaparameters simultaneously.

The name coordinate search comes from the epoch based search, where only a single parameter (one coordinate) is changed at a time. If a new best point was found, the searcher keeps this value and the search direction. Is the new parameter worse, the search direction of this parameters is switched and the step size is reduced. In Algorithm 5 there is a complete pseudo code for the structured coordinate search algorithm. The optimizer takes as input the function, which needs to be optimized. In our case it is the prediction RMSE on the cross validation set. References to N algorithm dependent parameters p_i are inputs to the searcher. After the optimization has finished, the output is the parameter set $\mathbf{p}$. Initial values $\overline{\mathbf{p}}$ and search step size $\mathbf{e}$ can be set at the begin of the coordinate search. New values are actual best values multiplied by the search step e_i. A good value for e_i is 0.8.

Optimization in a high dimensional parameter space is very difficult. The found solution manly depends on the initial parameter set $\overline{\mathbf{p}}$. Coordinate search is prone to find the first local minima and has problems escaping from it. As part of our implementation, we extend the coordinate search with a shrinkage of the search width when the error gets worse. This enables us to stabilize the search around a found solution. This modification is not part of the pseudo code in Algorithm 5.

```
Input: A function $f(\mathbf{p})$, $N$ meta-parameters $\mathbf{p} = [p_i]$
Output: Tuned metaparameters $\mathbf{p}$
Tunable: Start values $\overline{\mathbf{p}}$, search steps $\mathbf{e}$
Init search directions $\mathbf{d} = 0$
Init parameter position $i = 1$
Init best error $E_{min} = \infty$
Init parameters $\mathbf{p} = \overline{\mathbf{p}}$
foreach search epoch $j$ do
    $\mathbf{p}^{\mathbf{new}} = \mathbf{p}$
    if $d_i = 0$ (negative search direction of $p_i$) then
        $p_i^{new} = p_i \cdot e_i$ (try a smaller value)
    else
        $p_i^{new} = p_i \cdot {e_i}^{-1}$ (try a larger value)
    end
    $E_{new} = f(\mathbf{p}^{new})$  $(calculate\ new\ error)$
    if $E_{new} < E_{min}$ then
        $p_i = p_i^{new}$ (new best parameter found)
    else
        $(reverse\ search\ direction)$
        if $d_i = 0$ then
            $d_i = 1$
        else
            $d_i = 0$
        end
    end
    $i = i + 1$  $(set\ to\ next\ parameter\ position)$
    if $i > N$ then
        $i = 1$
    end
end
```

Algorithm 5: Optimize a function $f(\mathbf{p})$ with many parameters p_i by coordinate search

Retraining

In the first step, algorithm parameters were optimized by cross validation. Under “retraining” we understand to train the model with best parameters on the whole training data $\mathcal{L}$. For example, stop criteria for gradient descent based learning algorithms (such as neural networks) are also taken from the cross validation run. This re-evaluation of the model leads to better prediction accuracy by training on all available data, which lowers also the test error.

For predicting new test samples, we always use the retrained model. In other words we train the learner at the end again with best found meta-parameters (optimized on the RMSE of the cross validation set). This makes use of all available training data.

Chapter 4

Ensemble Learning Methods

Many techniques exist for the problem of how to combine predictions. We will describe in the following a few established and successful methods in ensemble learning. These algorithms rely on the outcome of many predictive models. The goal is to build a predictor, which has good generalization properties on new test samples.

4.1 Linear Combination

For a regression problem with quadratic objective function, optimal linear combination can be achieved via linear regression. We have an input matrix $\mathbf{X}$ and corresponding target values $\mathbf{Y}$. The solution to the minimization problem $\min_{\mathbf{W}} \ \|\mathbf{XW} - \mathbf{Y}\|^2$ is $\mathbf{W} = (\mathbf{XX}')^{-1}\mathbf{XY}$. In the context of ensemble algorithms, the columns of $\mathbf{X}$ are the feature-wise predictions and the rows $\mathbf{x_i}$ are so called feature vectors. We use no regularization is linear combination. This simple linear combination works fine (outcome of the final analysis). One can apply a non-negative constraint to the combination weights α, as suggested by Ting and Witten [41] in their stacked generalization framework. Stacking is explained in Section 4.4.

4.2 Bagging

Bagging [6], [40] is an acronym for "**b**oostrap **agg**reagat**ing**". It is a simple technique of resampling the training set with replacement and was first mentioned by Breiman [6]. Consider the training list $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i,)\}$ and the predictive model $\Omega(\mathbf{x}, \mathcal{L})$ as a function of the new input vector and the list of train data. The idea of bagging is now to construct new train sets $\mathcal{L}^{(b)}$ in order to produce diverse prediction behavior of the base model Ω. Bagging works with B copies of the learner, each individual is trained on a new boostrap sample $\mathcal{L}^{(b)}$. There are many sample duplicates in each boostrap set. Breiman shows that the amount of sample replicates in each bootstrap set $\mathcal{L}^{(b)}$ converges to about 63% [6]. The final outcome of the bagged ensemble is the average of all trained models

$$\Omega^B(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \Omega(\mathbf{x}, \mathcal{L}^{(b)}) \tag{4.1}$$

Breiman [6] shows that this works well for unstable models, where the build model behaves different if the train list is changed. For example, decision trees are well suited for bagging. He demonstrated the power of bagging on various UCI data sets with classification trees as learner. In his work, he figured out that 100 bootstrap replicated are enough to build a good ensemble. In general, bagging is a relative simple way to improve accuracy of a given method by adding computational power to the problem.

Bagging is also well suited for parallel computing because each run can be calculated independently. In Algorithm 6, there is a simple sketch of the whole procedure.

If the individual predictions in bagging runs are uncorrelated then the error of the committee is decrease by $1/N$, where N is the size of the ensemble. But this is never the case in real world data.

```
  Input: prediction model $\Omega(\mathbf{x}, \mathcal{L})$, training list $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$
  Output: bagged prediction of model $\Omega^B(\mathbf{x})$
1 foreach Bootstrap sample $i = 1..N$ do
2     Draw a new set $\mathcal{L}^{(i)}$ by sampling with replacement
3     Build the model $\Omega^{(i)}$
4     Add $\Omega^{(i)}$ to the ensemble
5 end
6 $\Omega^B(\mathbf{x}) = avg_i\, \Omega^{(i)}(\mathbf{x})$
```

Algorithm 6: The bagging algorithm for boost the performance of a prediction model

The gain in prediction accuracy of the bagged ensemble is mainly due to reduction in variance. Averaging over many slightly different models takes the way of the middle, the average prediction. For models with high biases, such as linear regression, bagging offers no advantages over the model itself.

4.3 Boosting

Boosting is a technique to improve accuracy of a pool of learners [13]. The basic idea is to construct a linear ensemble of models, each specialized to the training data for which previous learners had difficulties. In the early 90' R. Schapire has done the first study with boosting weak learners [33]. The most successful algorithm is AdaBoost. This technique was first mentioned in 1995 by Schapire [12]. It is designed to solve a binary classification problem by selecting a different base classifier in every cascade layer, based on a weighted sample error function, where the focus is on the hard-to-learn examples. When the AdaBoost stops learning, the final classifier is a linear combination of the selected base learners from all rounds.

```
  Input: Base learners $\Omega^t(\mathbf{x}, \mathcal{L})$, training list $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}, y_i = \{-1, 1\}, \mathbf{x}_i \in \mathbb{R}^F, i = 1..N$
  Output: The boosted binary prediction model $\Omega^T(\mathbf{x})$
1 For the weighted error function, initialize $w_i = \frac{1}{N}$
2 foreach training round $t = 1..T$ do
3     Fit a classifier $\Omega^t(\mathbf{x})$ to minimize the weighted error function on $\mathcal{L}$
4     $\epsilon_t = \sum_{i=1}^{N} w_i \cdot (y_i \neq \Omega^T(\mathbf{x}_i)) / (\sum_{i=1}^{N} w_i)$ (calc weighted train error)
5     $\alpha_t = 0.5 \cdot ln(\frac{1-\epsilon_t}{\epsilon_t})$ (calculate the weight of the classifier)
6     $\forall i : w_i^{new} = w_i \cdot exp(-\alpha_t y_i \Omega^T(\mathbf{x}_i))/Z$ (reweight and normalize examples, lay focus on
      hard-to-learn)
7 end
8 $\Omega^T(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t \Omega^t(\mathbf{x}))$
```

Algorithm 7: The AdaBoost algorithm for solving a binary classification problem with a pool of binary base learners

In the AdaBoost Algorithm 7, the examples have equal weights at $t = 1$ (first epoch), the first step is to fit a classifier which has the lowest error on the training set. This learner is called Ω^t. In the next steps ϵ_t (the confidence score) and α_t (the normalized confidence score) get updated. At the end of round t, all examples get reweighted, examples with higher train error receive a higher weight on the next round

($t = 2$). Samples which the model fits well have low error and their weights become lower. The error function of the AdaBoost ensemble learning algorithm has an exponential shape. The following equation is the error for a single example. The target is y and the prediction is p. The error is minimal, if $y = 1$ then $p = \infty$, if $y = -1$ then $p = -\infty$.

$$E = e^{-y \cdot p}, y \in \{-1, 1\}, p \in \{-1, 1\} \tag{4.2}$$

AdaBoost uses weak classifiers, which means a single classifier must have a slightly better performance than random guessing. The AdaBoost algorithm has nice generalization behavior. It can be shown that the algorithm maximizes the classification margin, even when the training classification error has reached value of 0.

Schwenk and Bengio investigate the ability of boosting neural networks for multi-class learning by using the AdaBoost.M2 algorithm [34],[31]. This algorithm modifies the original AdaBoost by a pseudo-loss term, which rely on a resampling technique based on the training error per example. Here, the reweighting of the error function is done by modify the sample probabilities. Algorithm 8 summarizes the AdaBoost.M2 learning algorithm.

```
Input: Base learners $\Omega^t(\mathbf{x}, \mathcal{L})$, training list $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$
Variables: $\mathbf{y}_i \in \mathbb{R}^C$ where $\mathbf{y}_i$ is generated by one-hot encoding [a] from $c_i$, $\mathbf{x}_i \in \mathbb{R}^F, i = 1..N$
Output: The boosted multi-class prediction model $\Omega^T(\mathbf{x})$
Initialize example probabilities equally by $d_i = \frac{1}{N}$
foreach training round $t = 1..T$ do
    if $t = 1$ then
        $\mathcal{L}^t = \mathcal{L}$ (in the first round take the original train set)
    else
        $\mathcal{L}^t = sample\,from(\mathcal{L}, \mathbf{d})$ (select samples with replacement with probability $d_i$)
    end
    Train the model $\Omega^t(\mathbf{x})$ on example list $\mathcal{L}^t$
    Predict the training set $\mathcal{L}^t$ and store real-value predictions ([0 1] normalized) in $\mathbf{p}_i \in \mathbb{R}^C$
    Calculate per-example margins: $q_i = \frac{\sum_{\forall c \neq c_i}(1+p_{i(c_i)}-p_{ic})}{C-1}$ ($c_i$ is the target class label)
    Calculate: $\epsilon = 0.5 \cdot \frac{\sum_{i=1}^{M} d_i \cdot \sum_{\forall c \neq c_i}(1-p_{i(c_i)}+p_{ic})}{C-1}$
    Calculate: $\beta_t = \frac{\epsilon}{1-\epsilon}$
    Update sample probabilities: $\forall i \in M : d_i = \beta_t^{0.5 \cdot q_i}$
    Normalize sample probabilities: $\forall i \in M : d_i = \frac{d_i}{\sum_{j=1}^{M} d_j}$
end
$\Omega^T(\mathbf{x}) = \arg max_c(\sum_{t=1}^{T} \beta_t \Omega^t(\mathbf{x}))$
```

Algorithm 8: The AdaBoost.M2 algorithm for solving a multi-classification problem with a pool of real-valued multiclass learners

[a]One value is 1 for the matching class, 0 otherwise. E.g. $\mathbf{y}_i = (0, 0, 1, 0)$ for a 4-class problem with target=3rd class

Algorithm 8 is designed to solve a multi-class problem (AdaBoost in its original form is only capable to solve binary classification problems). We give now the explanation how AdaBoost.M2 works. The algorithm works round-based, in the first round ($t = 1$) the model is trained on the full train data $\mathcal{L}$, after the model has finished training, every training example $(\mathbf{x_i}, y_i)$ is predicted by the trained model. The goal is to determine which examples are learned well. This is called the example margin q_i, it describes how well is the target class separated from all others. By calculating the pseudo-loss ϵ and β we are now able to update the sample probabilities d_i. After normalization, the next round starts with sampling a modified train set $\mathcal{L}^t$ with size N by sampling with replacement and using the per-example probabilities d_i. The AdaBoost.M2 algorithm stops when T rounds are performed. At the end we get a classification model, which is based on the trained models on different sample sets. The prediction class is the maximum response of the output vector.

The nice property of boosting is that it maximizes the margin between the positive and negative classes. This lead to good generalization behavior. AdaBoost is sensitive to wrong-labeled examples, they can destroy all the good properties. This means incorrect training data can lead to weird behavior and bad overall performance. In the experimental section we show results on the UCI LETTER dataset of the AdaBoost.M2 algorithm with neural networks as base learner (Section 5.3).

4.4 Stacking

Stacking or stacked generalization was first mentioned by Wolpert in 1992 [46]. Ting and Witten [41] give a comprehensive overview of the ensemble learning method Stacking. The basic idea is to train different prediction models in parallel on the same data. The training stops when each individual model begin to overfit on a k-fold cross validation set. This means the models are independently trained and combined with a loose coupling. I.e. there is no interaction in the training of the models. At the end the models are combined, as recommended by Ting and Witten [41], with a linear combination, where the blending weights are restricted to be non-negative. They investigated other combination strategies, such as decision trees or k-nearest neighbors algorithms, but non-negative linear combination delivered best overall results. The blending/combination model is called high-level model or level-1-model. Combination is more effective when low-level-learners predicts class probabilities rather than a single class label when solving classification problems. This means that all classification problems are converted with a simple encoding (see Section 2.4 for details) to multi-response regression problems.

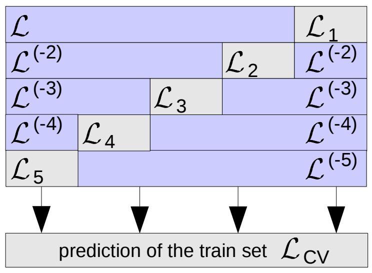


Figure 4.1: Prediction of the training set by cross validation to minimize the effect of self-influence. Self-influence is the effect of predicting examples, while training on these examples. A prediction of the train set with the model itself would go the zero error.

The ensemble learning algorithm stacking works as follows. Given a training list $\mathcal{L} = (\mathbf{x}_i, \mathbf{y}_i), i = 1...N$ and a list of supervised trainable models Ω^t. We have T models in total. Each of the models are called a low-level model or level-0 generalizer, which need to be combined is a clever way. All models are trained on the same training list $\mathcal{L}$. For example level-0 models can be naive bayes, decision tree, neural networks or k-nearest neighbors. The more diverse the models the better. The training should be controlled by k-fold cross validation. After the training of the t-th model has finished, the best prediction of the cross validation set L_{CV} of each model is stored (see Figure 4.1 for an collection example). Then the model Ω^t gets retrained with all available data, we store all model parameters in order to predict any new test sample. At the end we get for each model Ω^t a prediction of the complete training set, which is called L^t_{CV}. The predictions L^t_{CV} are a NxT matrix of outputs of class probabilities (in a classification problem). This is the input to the level-1 model, the combiner. Tine and Witten compares three different combination methods, a decision tree, a KNN and a linear combiner (MLR - multiple linear regression). Only the last combiner is suitable to improve the ensemble's performance over the best single prediction in various experiments. In Algorithm 9 we show how a linear equation system can be solved efficiently by having the non-negativity constraints on the weights. We use for the constants $\epsilon = 10^{-9}$ and $E = 1000$.

```
**Input**: Data matrix $\mathbf{A} \in \mathbb{R}^{NxF}$, targets $\mathbf{b} \in \mathbb{R}^{Nx1}$, precision $\epsilon$, maximum number of iterations $E$
**Output**: Non-negative regression weights $\mathbf{x} \in \mathbb{R}^{Fx1}$ (minimize the MSE: $\min_{\mathbf{x},\mathbf{x}>0} \|\mathbf{Ax} - \mathbf{b}\|^2$)
1  $\mathbf{r} = 1$ (init vector to zero, $\mathbf{r} \in \mathbb{R}^{Fx1}$)
2  $\mathbf{x} = 1$ (init solution to zero)
3  $\mathbf{C} = \mathbf{A}^T\mathbf{A}$ (precalculate the covariance matrix, $\mathbf{C} \in \mathbb{R}^{FxF}$)
4  $\mathbf{D} = \mathbf{A}^T\mathbf{b}$ (precalculate, $\mathbf{D} \in \mathbb{R}^{Nx1}$)
5  $e = 1$
6  while $\|\mathbf{r}\| < \epsilon$ do
7      $\mathbf{r} = \mathbf{D} - \mathbf{C} \cdot \mathbf{x}$ (the residuals)
8      foreach $i = 1..F$ do
9          if AND($x_i = 0$,$r_i < 0$) then
10             $r_i = 0$
11         end
12     end
13     $\alpha = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{r}^T\mathbf{Cr}}$ (adjust step size)
14     foreach $i = 1..F$ do
15         if AND($r_i < 0$,$\alpha > -x_i/r_i$) then
16             $\alpha = -x_i/r_i$
17         end
18     end
19     $\mathbf{x} = \mathbf{x} + \alpha \cdot \mathbf{r}$ (the weight update)
20     foreach $i = 1..F$ do
21         if $x_i < 10^{-9}$ then
22             $x_i = 0$
23         end
24     end
25     if $e = E$ then
26         break
27     end
28     $e = e + 1$
29 end
```

Algorithm 9: A iterative linear Regression solver with non-negative constraint to the weights

4.5 Cascade Learning

Cascade generalization from Gama and Brazdil [17] couples learners $\Omega_t, t = 1...T$ by a cascade-like structure, where algorithms in later cascades have access to all the predictions of algorithm in lower levels (probabilities per output of the train set). The output of the whole ensemble generates Ω_T (the last model in the cascade). The dimension F of the input increases continuously by C in every level of learning. C is the number of classes in a classification problem.

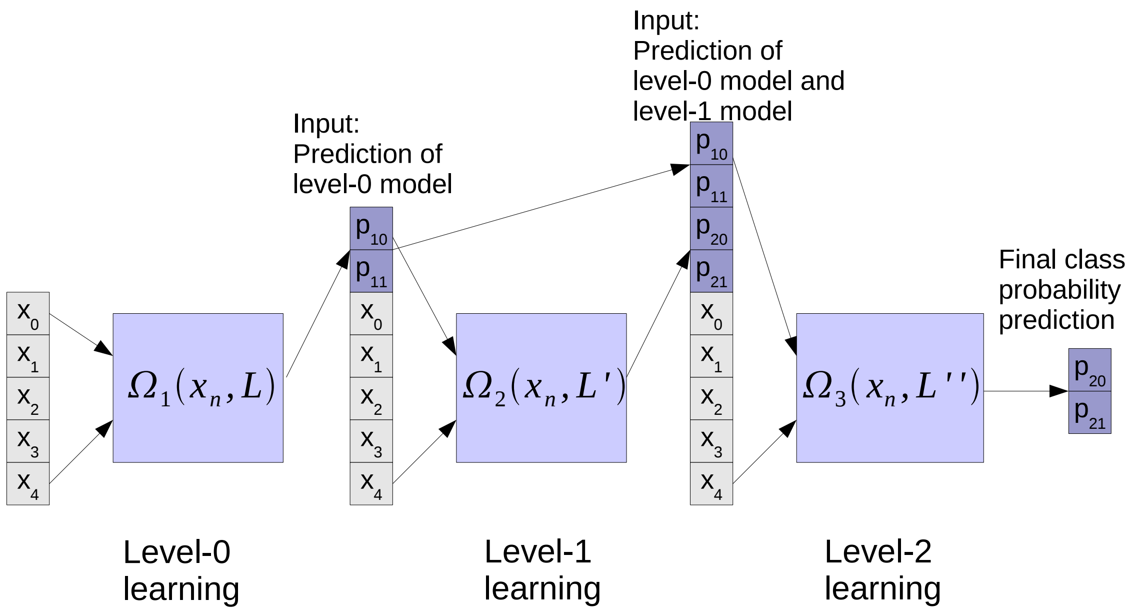
The ensemble learning algorithm "cascade learning" or cascade generalization fits the first model Ω_1 in the first cascade layer, based on the whole train list $\mathcal{L} = (\mathbf{x}_i, \mathbf{y}_i), i = 1...N$. The model in the second cascade Ω_2 are trained on $\mathcal{L}'$, this in an enlarged train set. In Equation 4.4 there is shown how the second-level train list is assembled. The new features $\mathbf{x}'_i$ are the original features $\mathbf{x}$ with the predictions of the Ω_1 appended. We get this predictions during training by taking the cross-fold prediction of the train set. For the third cascade layer (see Equation 4.5) the features $\mathbf{x}''_i$ are $\mathbf{x}_i$ appended by predictions from all previous layers.

$$\mathcal{L} = \{\mathbf{x}_i, \mathbf{y}_i\} \tag{4.3}$$

$$\mathcal{L}' = \{\overbrace{\{\mathbf{x}_i, \Omega_1(\mathbf{x}_i, \mathcal{L})\}}^{\mathbf{x}'_i}, \mathbf{y}_i\} \tag{4.4}$$

$$\mathcal{L}'' = \{\overbrace{\{\mathbf{x}_i, \Omega_1(\mathbf{x}_i, \mathcal{L}), \Omega_2(\mathbf{x}_i, \mathcal{L})\}}^{\mathbf{x}''_i}, \mathbf{y}_i\} \tag{4.5}$$

The Equation 4.3 denotes the original training list, Ω_1 learns on it. Equations 4.4 and 4.5 are the augmented training data for learners Ω_2 and Ω_3. This means output of the learners in previous cascades are inputs to higher-level cascades. In Figure 4.2 there is an overview of the “cascade learning” process.



Input: Prediction of level-0 model and level-1 model
Input: Prediction of level-0 model
Final class probability prediction
x_0 x_1 x_2 x_3 x_4
$\Omega_1(x_n, L)$
p_{10} p_{11}
$\Omega_2(x_n, L')$
p_{20} p_{21}
$\Omega_3(x_n, L'')$
Level-0 learning
Level-1 learning
Level-2 learning

Figure 4.2: Cascade generalization. Models on higher cascade levels have access to predictions from lower level models. The training data here is for example 4-dimensional. The output is 2-dimensional. The last model in the cascade makes the prediction of the whole ensemble.

Computation in cascade learning is serial, therefore parallelization of the cascade is impossible. In contrast to cascade learning, stacking can be trained in parallel.

4.6 Training on Residuals

In many supervised learning problems, a single model can not learn the given task perfectly. It always makes an error on the training samples. This error is called “residual” or model error. One part of the error is unavoidable because of noise in the data, the other part is the systematic error, the error that the model’s structure can not capture. The principle of “training on residuals” is to fit another model to the residuals. In Figure 4.3 there is a graphical sketch of the information flow. All models Ω_t in the ensemble are trained on the same training features $\mathbf{X}$, the targets are taken from the residuals of the previous model. The first model Ω_1 (layer 0) is trained on the original training list $\mathcal{L} = (\mathbf{x}_i, \mathbf{y}_i), i = 1...N$. The second model Ω_2 (layer 1) is trained on the targets $\mathbf{Y} - \Omega_1(\mathbf{X})$. This means that learner Ω_t tries to learn the structure of the data, which was not learned by the models in the previous layers $1...t-1$. The final outcome can be either the prediction of the last algorithm in the chain or a linear combination of the predictions in all layers.

We need predictions of the training targets in order to create residuals of the training set. The residuals are the original targets subtracted by the model prediction (Figure 4.3). When we speak about prediction, we mean the prediction of a particular model in the chain. As prediction we use the values from the cross validation metaparameter selection process. This has the property that the validation samples on each fold of the cross-validation are not part of the model's training set (see Section 2.6).

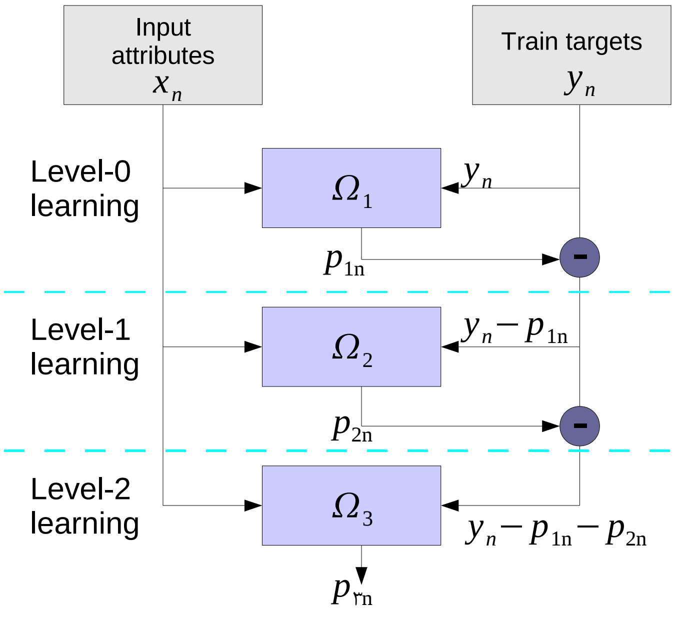


Figure 4.3: Training on residuals of other models.

The problem with training on residuals is the effect of self-influence. This leads to over-optimistic cross validation estimates of the true error. We use the best prediction of the cross validation process in order to generate the new training targets. The effect of self-influence starts when training > 1 models.

In the experimental evaluation, we combine this method with "blend stopping" (see below). This means that every time a model Ω_t is added to the ensemble and trained, we try to minimize the whole ensemble's RMSE by stopping the training of this model when the error of the linear combination over all models $\Omega_1...\Omega_t$ is minimal.

4.7 Blend Stopping

This technique is used in combination with the "training on residuals" ensemble method. As the name indicates, we search for meta-parameters, or stop training a gradient descent algorithm, when the error of the combination (blending) of the ensemble becomes minimal. The RMSE is minimized through a regularized linear regression of the current ensemble. In other words, we stop training of the individual model when the error of the linear blend is minimal. This idea stems from the Netflix Prize competition, where we used it to minimize the error of an ensemble of predictions rather than the individual predictors

(Sec. 3.2 in BigChaos Grand Prize Report [42]).

Blend stopping relies on the cross validation error on the training set where one constant predictor is added to the ensemble. Even in the first model in the ensemble, we add a constant prediction to the linear combination, the constant acts like an prior estimate or can center the predictions. The objective is

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \|\mathbf{y}_i - \overbrace{[1\ \mathbf{p}_i]^T\mathbf{w}}^{\text{lin. combination}}\|^2 + \lambda \sum_i w_i \tag{4.6}$$

where N is the number of training examples. The notation $[1\ \mathbf{p}_i]$ indicates that the constant 1 is appended to the ensemble's predictions. Training targets are denoted by $\mathbf{y}_i$, the regression weights are denoted by the column vector $\mathbf{w}$ and $\mathbf{p}_i$ are the predictions from the cross validation process. The resulting weights $\mathbf{w}$ without the use of cross validation would lead to large overfitting. This phenomenon was also discovered by Leo Breiman in 1996 in the paper "Stacked Regressions" [7]. He suggested to use leave-one-out cross validation in order to get predictions for every model on the training data.

The second problem, which Breiman [7] analyzed, is that all models in the ensemble try to predict the targets and therefore they are highly correlated. This could be a problem for the linear combination, for example very high/low weight pairs can occur when predictions differ only slightly. He suggest to use again cross validation in combination with ridge regression to select a good value for the ridge regression constant λ (the same constant as in the model linear regression, Section 3.3).

Breiman [7] used non-negative weights with the constraint that the weight sum is 1, $\sum_i w_i = 1$. In contrast to him, we allow negative weights. In all our experiments with blend stopping (in combination with training on residuals) we use a small constant regularization constant $\lambda = 10^{-5}$.

4.8 Ensemble Selection

The technique of ensemble selection [11] was proposed in 2004 by Caruana et.al. In 2009, Alexandru Niculescu-Mizil and his 11-member team from IBM research won the KDDCup09 [1]. The winning entry was based on ensemble selection on a library of 1200 single models. The KDDCup09 was a large-scale supervised learning problem with 3 different binary targets. The number of training samples are 50000, the number of features was 15000 (numerical and categorical). The goal was to achieve the best AUC score on a hidden test set (size 50000).

The technique "ensemble selection" [11] is based on a library of thousands of models. The models are build using different learning algorithms with various parameter settings. The combination strategy is based on a greedy forward selection algorithm. Ensemble selection starts with the best model of the library, in the next round ensemble selection chooses the predictor, which improves the error the most. This improvement is typically measures on a validation set. In the third round, again a predictior is selected from the whole library. The combined predictor approximates a weighted average. Ensemble selection enables us to optimize any performance metric (such as RMSE, AUC, accuracy,...). In [11] they investigate the optimization on 10 different error measures. The combined result is a non-negative combination of the selected learners, because predictors can be selected multiple times. One of the advantages of ensemble selection is that is can be fully parallelized, which means the library (the most time consuming part) can be build independently.

We were not able to do experimental evaluation of this ensemble learning technique, due to the requirement of having access to thousands of predictions. Despite of this aspect, ensemble selection is one of the latest achievements and is a state-of-the-art technique in ensemble learning. The success of ensemble selection is proved on many datasets and contests. The big advantage is that is is able to optimize any performance metric effectively.

[1] Fact-sheet of the winning entry: `http://www.kddcup-orange.com/factsheet.php?id=66`

4.9 Issues in Overfitting and Objective Evaluation

Our intention is to perform an objective evaluation of our experiments. This means that the whole ensemble and all its individual models are build by using the data from a training set. During training, the test set is not used. It is very important to consider the issue of self-influence when evaluating methods. For example we observe some over-optimistic estimate of the cross validation error when training on residuals. The learning strategy of “cascade learning” is based on a chain of models. Estimated errors at the end can be much better than on a real test set. In general, when a single model is trained, the error on a test set should be lower compared to the cross-validation error because we retrain the model on all available data.

Chapter 5

Evaluation on Datasets

This is the main part of the thesis, the evaluation of different models and ensemble learning techniques on various data sets.

5.1 Toy Data

Toy data is generated by an artificial process. The input space is typically two-dimensional. The optimal decision boundary can be calculated by using optimal Bayes decisions, when the sampled distribution is known (see Figure 2.5 in Hastie [18]). This makes visualization easy. After training we can divide the input space into a discrete grid and apply the algorithm on each grid point. In this way we obtain a color-coded image of the predictions on the input space. This gives us a feeling how different types of models behave. Three different toy problems are analyzed: “Spiral”, “Circle” and “Chess”. All of them are binary classification problems, the first class is encoded with small red circles, the other class with blue circles (see Table 5.1). As mentioned in previous sections, we use 1-hot encoded target vectors to train all learners, hence a two-class classification problem leads to two outputs p_0 and p_1. The target values are given by $+1$ for class red and -1 for class blue. We visualize the prediction value v with $v = 0.5 \cdot (p_0 - p_1)$, furthermore we clip v outside of $[-1.5, +1.5]$ to avoid large prediction values.

By illustrating predictions as images, we obtain a good impression of what was learned. Even a non-expert in machine learning can see how the algorithm classifies the data. The next items summarize the advantages of the use of toy data for the comparison of learning algorithms.

- Decision boundaries can be visualized
- Behavior of the algorithm on regions, where no training data is available can be observed
- The power of the algorithm to split the input space is visualized
- The operation point of a classifier is equivalent to the selection of a color as threshold, this works for binary problems

The Figure 5.1 shows the Spiral dataset, separated by a KNN model.

The three toy data examples are generated by the Spider machine learning framework [44] (Spiral-data: Figure A.1, Circle-data: Figure A.2 and Chess-data: Figure A.3, all are available in the Appendix). Spider is an object orientated environment for machine learning in Matlab. It is written by members from the Max Planck Institute for Biological Cybernetics, the code is available online [1]. In the Figure’s caption we note the command, for generating the data (Appendix).

[1]`http://www.kyb.tuebingen.mpg.de/bs/people/spider/main.html`

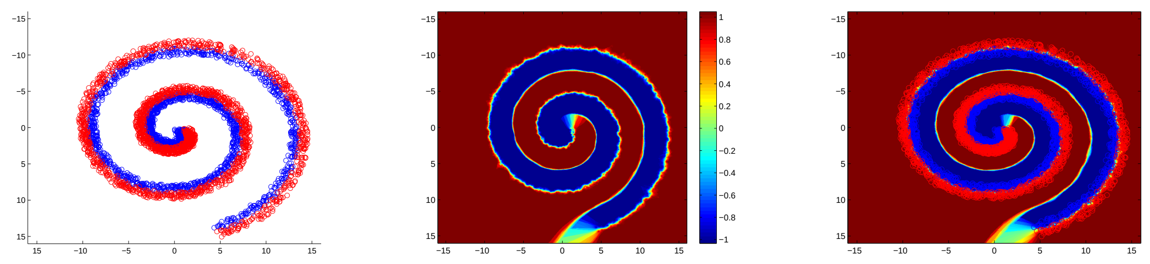


Figure 5.1: Predictions of a KNN model on the Spiral data set. The Spiral data set (left), and the prediction of a KNN model (middle) with $k = 8$. The right plot shows the prediction with the data superimposed.

K-Nearest Neighbors
The behavior of the KNN algorithm can be easily explained for all three data set. The prediction depends on the closest neighbors in the feature space. This technique of nearest neighbors works perfect, if the input space is densely sampled.

Polynomial Regression We use polynomial regression with all cross terms, hence the algorithm name PolyX-cross. This results in a feature space extension with all interaction between feature dimensions. For the toy examples polynomial regression was not able to achieve low classification error, the predicted output is typical for polynomials.

Gradient Boosted Decision Tree The predicted values are typically aligned in rectangles, this is the behavior of a Decision Tree. A tree is based on axis-parallel splits of one particular input feature. Gradient boosting enables the tree to result in a more smooth surface. We observed an improvement in the Spiral and the Circle data set when the GBDT is trained with random split values (GBDT-randomSplit). In other words the threshold value in the nodes is just a random value, not the value that reduces the RMSE best. The Chess data set is perfectly suited for decision trees due to the axis parallel data alignment.

Neural Networks The number of the neurons and the number of layers specify the power of a neural network. On the Spiral dataset the neural net with only one hidden layer was not able to learn the structure of the data, the small 2 hidden layer 5-5 net has too little expressiveness, the 30-30 net was able to learn the task. On the circle dataset, all net configurations were able to separate the red ring. In the chess data, only the larger 2 hidden layer net delivers good results.

Kernel Ridge Regression The Gaussian kernel works perfect for all toy tasks, the poly and the tanh kernel have worse results.

In toy data, the input space is sampled very densely compared to real-world problems (see Section 2.3, The curse of dimensionality). Predictive behavior of models are not well comparable to high-dimensional inputs, good and meaningful visualization is impossible when having many input features.

5.2 14 smaller UCI Data Sets

In Section 4 we discussed many ensemble learning methods. To determine, which technique works best, a large evaluation on various UCI data sets is performed in this section.

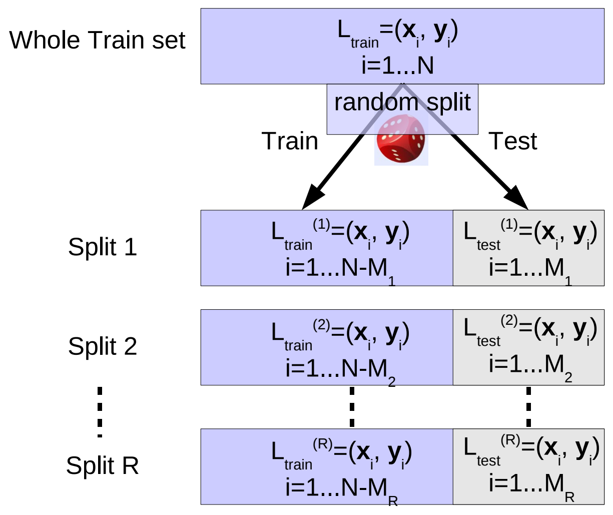

Whole Train set
$L_{train}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...N
random split
Train
Test
Split 1
$L_{train}^{(1)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...N-M_1
$L_{test}^{(1)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...M_1
Split 2
$L_{train}^{(2)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...N-M_2
$L_{test}^{(2)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...M_2
Split R
$L_{train}^{(R)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...N-M_R
$L_{test}^{(R)}=(\mathbf{x}_i, \mathbf{y}_i)$ i=1...M_R

Figure 5.2: Measurement of prediction accuracy by average over multiple test-set splits. We use as test set a random subset of the available data (20% on average). The number of splits is R, by averaging over many simulated test sets we can estimate the prediction accuracy and the variance (the stability) of the result. Furthermore we apply a paired-t test to report significant improvements. We used $R = 100$ in this thesis.

We use data sets from the UCI machine learning repository from the University of California, Irvine [3]. It is a very comprehensive collection of different databases. Most of the UCI data sets consist of a training table of features and targets/labels with no predefined test set. We divide each dataset into two sets, the training set and the test set, see Figure 5.2. Each data sample was choosen as training sample with probability 0.8. This resulted in a training set that consisted of 80% of the data samples on average. The test set was given by the remaining samples. We generated 100 such splits for each data set. By averaging over the errors of 100 splits we obtain the mean and the standard deviation of the classification error (accuracy). For all performed experiments we use our machine learning framework. This framework was named ELF (ensemble learning framework), see Section 7 for an introduction. Table 5.1 gives an overview over the datasets used in this section.

The aim of the evaluation is to determine which combination or configuration and which ensemble learning algorithm is the best. The big picture of the experimental evaluation can be found in Figure 5.3. We use four different supervised machine learning techniques as base learners (linear regression, k-nearest neighbors, kernel ridge regression, neural networks). Additionally, we explore in different combinations and different orders of the base learners. We compare “Stacking”, “Cascade Learning”, “Training on Residuals” and “Cascade Learning combined with Training on Residuals”. We do all the evaluations on 14 different UCI data sets, which are listed in Table 5.1. These data sets were chosen for easy comparison with experiments in [41] and [17]. We skip larger data sets (over 1000 train samples) due to the fact that runtime in the exploration process would increase a lot (Figure 5.3). All the data sets have no predefined test set.

The data sets are summarized in the following.

CREDIT

The CREDIT data set [30] has 8 categorical and 6 numerical attribute values. The number of samples is 690. It is a binary classification task and concerns credit card applications. It can be found under `http://archive.ics.uci.edu/ml/datasets/Statlog+(Australian+Credit+Approval).`

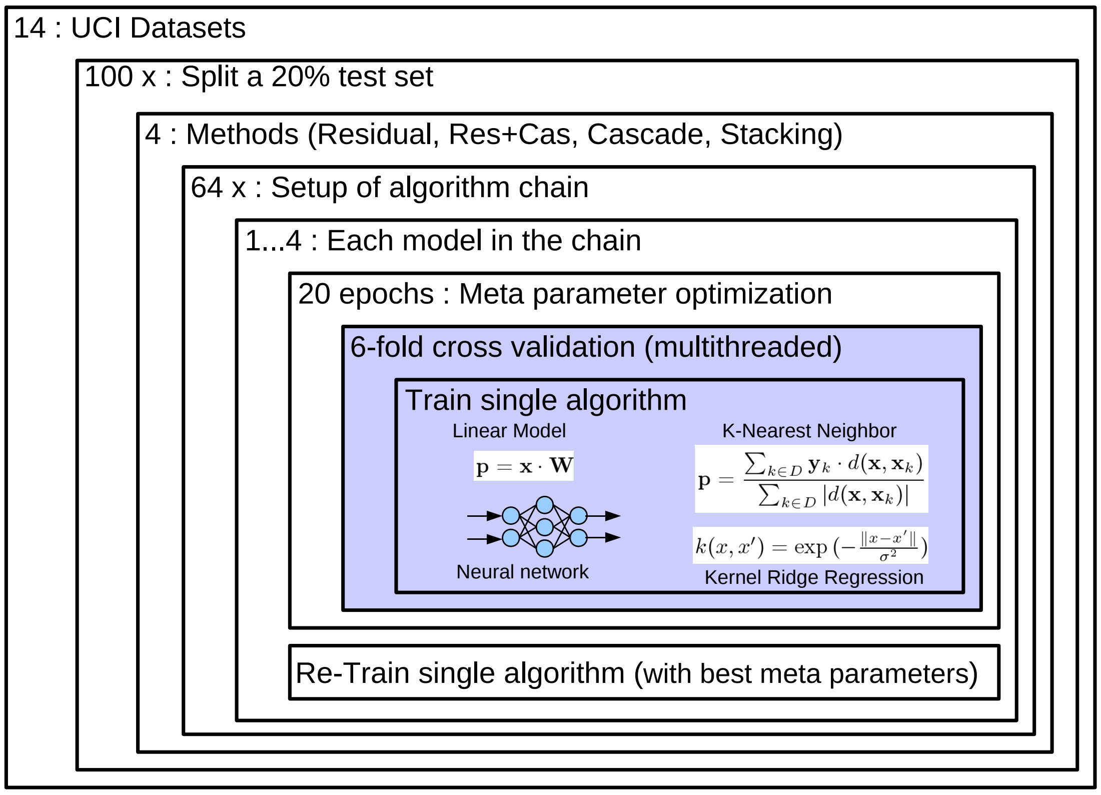

14 : UCI Datasets
100 x : Split a 20% test set
4 : Methods (Residual, Res+Cas, Cascade, Stacking)
64 x : Setup of algorithm chain
1...4 : Each model in the chain
20 epochs : Meta parameter optimization
6-fold cross validation (multithreaded)
Train single algorithm
Linear Model
$\mathrm{p} = \mathbf{x} \cdot \mathbf{W}$
Neural network
K-Nearest Neighbor
$\mathrm{p} = \frac{\sum_{k \in D} \mathbf{y}_k \cdot d(\mathbf{x}, \mathbf{x}_k)}{\sum_{k \in D} |d(\mathbf{x}, \mathbf{x}_k)|}$
$k(x, x') = \exp(-\frac{\|x - x'\|}{\sigma^2})$
Kernel Ridge Regression
Re-Train single algorithm (with best meta parameters)

Figure 5.3: Cascaded loops in evaluation on UCI datasets. The explaination goes from the outer loop to the inner. The outer loop (14x) is responsible for the analysis of all 14 data sets. The next loop (100x) splits the data set randomly into train and test set. We evaluate 4 different ensemble learning methods (4x). The next loop (64x) creates different chains of models, the order of four different models is varied. Each model in the chain is going to be optimized (4x). We search for maximum number of 20 epochs per algorithm in order to optimize meta parameters (20x). The optimization is done on a 6-fold cross validation set, this runs multithreaded. In total we make $14 \cdot 100 \cdot 4 \cdot 64 \cdot 4 \cdot 20 \cdot 6 = 1.7 \cdot 10^8$ model training steps.

BALANCE

The BALANCE data set has 4 numerical attribute values. The number of samples is 625. It is a 3-class classification task. The data was collected in order to model psychological experiments. The goal is to model the scale balance equation: (left-distance * left-weight) and (right-distance * right-weight). It can be found under `http://archive.ics.uci.edu/ml/datasets/Balance+Scale`.

BREAST

The BREAST data set [45] has 10 numerical attribute values. The number of samples is 699. It is a binary classification task. The data stem from medical science. It can be found under `http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)`.

DIABETES

The DIABETES data set [38] has 8 numerical attribute values. The number of samples is 768. It is a binary classification task. The data are collected from females, at least 21 year old and Pima In-

Name	#classes	#feat	#feat(intern)	#samples
CREDIT	2	14	43	690
BALANCE	3	4	5	625
BREAST	2	10	11	699
DIABETES	2	8	9	768
GERMAN	2	20	62	1000
GLASS	6	9	10	214
HEPATITIS	2	19	20	155
IONOSPHERE	2	34	35	351
IRIS	3	4	5	150
SONAR	2	60	61	208
SURVIVAL	2	3	4	306
VEHICLE	4	18	19	846
VOTES	2	16	49	435
WINE	3	13	14	178

Table 5.1: Overview of small UCI datasets. With #feat(intern) we denote the number of attributes, which are actually used inside the algorithms (e.g. categorical value encoding, adding of a constant feature column).

dian heritage, the task is to classify the disease (positive or negative). It can be found under `http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`.

GERMAN

The GERMAN data set has 7 numerical and 13 categorical attribute values. The number of samples is 1000. It is a binary classification task. The data are collected from the Institut für Statistik und Ökonometrie at university of Hamburg and contains credit related attributes. It can be found under `http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)`.

GLASS

The GLASS data set has 9 numerical attribute values. The number of samples is 214. It is a 6-class classification task. The data are collected from the Home Office Forensic Science Service (UK), the task is to classify the type of glass based on chemical related attributes. It can be found under `http://archive.ics.uci.edu/ml/datasets/Glass+Identification`.

HEPATITIS

The HEPATITIS data set has 19 numerical attribute values. The number of samples is 155. It is a binary classification task. The data is based on medical properties, the task is to determine if the patients die or live. It can be found under `http://archive.ics.uci.edu/ml/datasets/Hepatitis`.

IONOSPHERE

The IONOSPHERE data set [36] has 34 numerical attribute values. The number of samples is 351. It is a binary classification task. The data are collected by a radar in Goose Bay, Labrador. The target is to determine the free electron status good or bad. It can be found under `http://archive.ics.uci.edu/ml/datasets/Ionosphere`.

Name	LM	PR	NN	KNN	KRR
CREDIT	13.46±2.5	15.13±2.0	14.44±2.3	16.70±3.1	**13.25**±3.0
BALANCE	12.26±2.5	8.50±1.9	8.42±1.6	11.26±2.5	**8.38**±1.6
BREAST	3.95±1.8	3.27±1.6	3.43±1.2	**3.02**±1.4	3.51±1.6
DIABETES	23.57±2.1	**22.10**±2.8	23.51±3.5	23.53±3.1	23.32±3.5
GERMAN	24.74±3.6	25.07±3.1	**24.20**±2.8	26.19±3.4	25.10±2.9
GLASS	41.86±5.7	38.01±5.3	36.93±8.3	**30.19**±6.8	30.33±6.1
HEPATITIS	15.81±6.0	15.90±4.2	**13.43**±6.6	15.59±5.9	13.79±4.9
IONOSPHERE	12.48±4.0	8.12±2.7	12.19±2.9	15.35±4.0	**7.39**±2.8
IRIS	16.39±8.8	4.89±4.2	**3.73**±3.4	5.05±4.1	5.54±3.5
SONAR	23.13±5.6	18.77±7.4	24.14±5.7	16.24±6.1	**13.62**±4.5
SURVIVAL	24.39±5.3	**24.20**±3.9	24.38±4.2	26.07±4.6	25.81±5.5
VEHICLE	23.51±3.5	20.86±1.4	**17.46**±2.6	29.30±2.9	19.31±2.7
VOTES	4.12±1.5	4.49±2.1	4.91±2.4	7.17±2.7	**4.01**±1.6

Table 5.2: Test error for single models on small UCI data sets. Reported is the percentage of classification error (mean ± std). Bold faced numbers are the winners per data set. LM=linear model, PR=polynomial regression, NN=neural network, KNN=k-nearest neighbors, KRR=kernel ridge regression.

Name	LM	PR	NN	KNN	KRR
CREDIT	0.65±0.04	0.67±0.04	0.65±0.05	0.71±0.04	**0.63**±0.04
BALANCE	0.56±0.03	0.56±0.03	**0.38**±0.03	0.51±0.03	0.46±0.02
BREAST	0.37±0.04	0.36±0.05	**0.32**±0.05	0.32±0.06	0.33±0.07
DIABETES	0.80±0.03	**0.77**±0.03	0.79±0.04	0.82±0.04	0.79±0.04
GERMAN	0.82±0.04	0.83±0.03	**0.81**±0.03	0.85±0.03	0.82±0.03
GLASS	0.61±0.03	0.62±0.03	0.60±0.04	**0.53**±0.05	0.54±0.04
HEPATITIS	0.69±0.08	0.70±0.08	**0.62**±0.10	0.64±0.12	0.65±0.05
IONOSPHERE	0.64±0.07	0.55±0.04	0.62±0.06	0.70±0.08	**0.49**±0.06
IRIS	0.60±0.07	0.35±0.06	0.35±0.07	0.32±0.08	**0.30**±0.07
SONAR	0.81±0.06	0.76±0.13	0.81±0.07	0.68±0.09	**0.63**±0.06
SURVIVAL	0.83±0.04	0.85±0.04	0.84±0.05	0.87±0.08	**0.83**±0.06
VEHICLE	0.60±0.02	0.57±0.02	**0.51**±0.02	0.61±0.02	0.52±0.02
VOTES	**0.36**±0.04	0.37±0.07	0.38±0.07	0.45±0.06	0.38±0.04

Table 5.3: Test error for single models on small UCI data sets. Reported is the RMSE (mean ± std). Bold faced numbers are the winners per data set. LM=linear model, PR=polynomial regression, NN=neural network, KNN=k-nearest neighbors, KRR=kernel ridge regression.

IRIS

The IRIS data set has 4 numerical attribute values and is one of the most popular sets used in pattern recognition. The number of samples is 150. It is a 3-class classification task. The data is from 1936 collected by Fischer, the target is to recognize iris plants based on physical dimensions of the sepal and petal. It can be found under `http://archive.ics.uci.edu/ml/datasets/Iris`.

SONAR

The SONAR data set has 60 numerical attribute values. The number of samples is 208. It is a binary classification task. The data comes from bouncing sonar signals under different angles off a metal cylinder. It can be found under `http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)`.

SURVIVAL

The SURVIVAL data set has 60 numerical attribute values. The number of samples is 306. It is a binary classification task. The data stem from a long-term study on the survival rate when having breast cancer. It can be found under `http://archive.ics.uci.edu/ml/datasets/Haberman's+Survival`.

VEHICLE

The VEHICLE data set has 18 numerical attribute values. The number of samples is 846. It is a 4-class classification task. The data comes from classifying vehicles, based on image shape descriptors/features. It can be found under `http://archive.ics.uci.edu/ml/datasets/Statlog+(Vehicle+Silhouettes)`.

VOTES

The VOTES data set has 16 categorical attribute values. The number of samples is 435. It is a binary classification task. The data are collected by Jeff Schlimmer and includes votes for each of the U.S. House of Representatives. The task is to classify inputs whether they belong to democrat or republican. It can be found under `http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`.

WINE

The WINE data set has 13 categorical attribute values. The number of samples is 178. It is a 3-class classification task. The data results from a chemical analysis of wines grown in Italy. It can be found under `http://archive.ics.uci.edu/ml/datasets/Wine`.

The UCI Results

Table 5.2 lists the classification test errors for every single algorithm. Bold faced numbers denote the winners (lowest classification error) per dataset. Again, statistics are based on 100 random test splits. Remember that the metaparameters of every algorithm are optimized to lower the RMSE over the cross validation data. For completeness, we list the RMSE values per algorithm in Table 5.3, the best RMSE value does not necessarily correspond to best classification error. The trend is that algorithms with low RMSE also has low classification error. RMSE as an error measure is more stable for optimization of metaparameters (like class probabilities), therefore lowering the classification error directly leads to getting stuck early in local minima. We tried this and the outcome points to worse test accuracy.

Classification error is a discretized measure, while RMSE is smooth real valued. Table 5.4 compares best ensemble techniques (using the best chain configurations for each of the ensemble learning techniques). Table 5.5 lists the best configurations (the models and the exact ordering). Table 5.6 and Figure 5.4 compares the errors of the best single versus the best ensemble learning technique.

Now we briefly return to single model results. The goal of each individual model is to minimize the prediction RMSE (see Equation 1.2). 8 of the 14 datasets ($= 57.1\%$) have the minimum both on RMSE and classification error (CREDIT, DIABETES, GERMAN, GLASS, HEPATITIS, IONOSPHERE, SONAR, VEHICLE). All other tested data sets have different winners. Furthermore, we analyze the accuracy of the learner, which is equivalent to classification error. The worst single model is linear regression (0 wins) followed by polynomial regression and k-nearest neighbors (both 2 wins), neural networks (4 wins) and the single model winner is **kernel ridge regression** (5 wins).

In Table 5.4 we see the comparison of the results from ensemble learning methods. The technique Training on Residuals never wins, Stacking has 3 wins, Residual+Cascade Learning wins 5 times and **Cascade Learning** has 7 times the best accuracy, therefore it is the overall winner. Additionally to the mean classification error we report the standard deviation of them, it is quite large compared to the margin of the winner to the first runner-up. A significance analysis of the margin is given in Table 5.6 and Figure 5.4.

For each ensemble learning technique, we try all possible model chains (as shown in Figure 5.3), the winners are listed in Table 5.5. A chain is a special ordered configuration of learners, combined by one of the 4 ensemble learning techniques. Such a chain can be a linear regression, followed by a k-nearest neighbors model, where the predicted output of the linear regression is feed into the KNN's training list, this is the configuration for cascade learning. We have 4 base learners, in exploring all different chain combinations of size 1, 2, 3 and 4 we end up with 64 different setups. In Stacking, the order of the models do not play a role, because of the loose coupling, therefore we have in Stacking 16 different setups. Our intention was to figure out a best configuration chain, but that was unsuccessful. Different chains of single models are winners, so there exists no "best configuration" (see Table 5.5). J. Gama and P. Brazdil in [17] mentioned that weak models should be placed in front of strong models. Based on our results, we can not confirm their suggestion. Every data set has its own best setting.

In Table 5.6 we analyze the significance of the improvement of the best ensemble model compared to the best single model. It turns out that ensembling often outperform the best single model. Ensemble learning wins on 13 of 14 cases, only on the GERMAN data set kernel ridge regression delivers the same performance as single model. Statistical significance analysis over 100 test splits shows that ensemble learning outperforms the best single model 6 times with $p < 0.05$ in a paired t-test. This is done by simply using the function `ttest` from Matlab.

The number of training samples is the dominating value for training time of the ensemble, because some single models (KNN, KRR) have quadratic runtime with respect to the sample size. The simulation time goes from 0.5 hours (HEPATITS, 155 samples) to 10.7 hours (GERMAN, 1000 samples) on an Intel i7 QuadCore Desktop PC with 12GB RAM running on 3.8GHz. The meta-parameter optimization process with 6-fold cross validation is run in parallel (2x3 threads).

Graphical visualization of the accuracy of best ensembles and the best single models is plotted in Figure 5.4. For each data set and each test run we place the result (classification error) to the corresponding plot. Some test splits result in the exact same classification error, therefore the histogram-like plot style. In only one (BALANCE) of the 14 data sets the outcome of ensemble learning clearly outperforms the best single models, here the red samples (ensemble learning) have a clear margin to the blue samples (single model). In all other data sets the winner is not very clearly visible. The spread of the observed classification errors is high compared to the difference of the mean errors on ensemble versus single model learning. This means we have a high variance of the experimental analysis on all 14 UCI data sets. As discussed before, further analysis of the significance can be found in Table 5.6.

All detailed outcomes, assorted per dataset, are attached in the Appendix section, the corresponding

Dataset	Residual	Residual+Cascade	Cascade	Stacking
CREDIT	13.65±3.0	13.65±3.0	**13.43**±2.9	13.78±2.3
BALANCE	7.88±2.4	4.40±2.2	**4.31**±2.2	8.79±2.1
BREAST	3.11±1.3	3.05±1.6	**2.96**±1.4	3.11±1.3
DIABETES	22.51±2.9	**22.29**±3.2	22.49±3.0	22.55±3.1
GERMAN	23.59±2.8	23.59±2.8	**23.46**±2.8	**23.46**±2.8
GLASS	30.41±6.3	**30.16**±6.5	30.59±5.8	30.91±6.2
HEPATITIS	15.00±6.8	15.00±6.8	14.32±6.0	**14.01**±5.7
IONOSPHERE	7.26±2.9	**6.79**±3.0	7.04±2.9	8.24±2.7
IRIS	3.52±3.1	**3.22**±2.9	3.26±2.6	4.20±3.6
SONAR	13.12±5.2	13.12±5.2	**11.29**±3.9	12.72±5.5
SURVIVAL	24.66±4.7	24.86±4.4	25.20±5.3	**24.59**±4.5
VEHICLE	19.08±2.7	18.24±2.5	**17.55**±2.5	18.52±3.0
VOTES	3.93±2.0	4.13±1.9	**3.72**±1.9	4.02±1.9
WINE	1.10±1.5	**1.05**±1.5	1.33±2.0	1.13±1.6
Number of Wins	0	5	7	3

Table 5.4: Comparision of best ensemble learning methods for each dataset. Values are classification error ± standard deviation on 100 random test sets. Per result, the best configuration of single models (ordering) is used here. The listing of best configurations is in Table 5.5.

tables are: Table A.1 for CREDIT, Table A.2 for BALANCE, Table A.3 for BREAST, Table A.4 for DIABETES, Table A.5 for GERMAN, Table A.6 for GLASS, Table A.7 for HEPATITIS, Table A.8 for IONOSPHERE, Table A.9 for IRIS, Table A.10 for SONAR, Table A.11 for SURVIVAL, Table A.12 for VEHICLE, Table A.13 for VOTES, Table A.14 for WINE.

Dataset	Residual	Residual+Cascade	Cascade	Stacking
CREDIT	L	L	**K-R-N**	K-R-L-N
BALANCE	R-K-L-N	N-L-K	**N-L-R-K**	L-N
BREAST	N-L-R	L-K	**L-R-N**	K-R-L
DIABETES	R-N-L	**R-L-K**	R-L-K	K-L
GERMAN	R	R	**R**	**R**
GLASS	R	**R-L**	R-L	R
HEPATITIS	N	N	L-R	**K-R-L-N**
IONOSPHERE	R-N-L	**R-K-N**	R-L-K-N	R
IRIS	K-N	**N-L-R**	N-L	R-L-N
SONAR	R	R	**K-L-R**	K-R
SURVIVAL	N-R-L	N-R-L	L-N	**K-N**
VEHICLE	N-L-R	N-L-K	**N-K-R-L**	K-R-N
VOTES	N-R-K-L	N-R-K-L	**N-R**	K-R-L-N
WINE	L-N	**L-R**	R-L	K-R-L-N

Table 5.5: Compare the success of different ensemble configurations per dataset, we report the winning chains. Here, we list the **winners** with lowest classification error.

Dataset	Best Single	Best Ensemble	p-value	significant? ($p < 0.05$)
CREDIT	13.65±3.0 (L)	13.43±2.9 (K-R-N,Cas)	0.598	no
BALANCE	8.96±2.1 (N)	4.31±2.2 (N-L-R-K,Cas)	0.000	yes
BREAST	3.24±1.3 (R)	2.96±1.4 (L-R-N,Cas)	0.112	no
DIABETES	23.10±3.2 (L)	22.29±3.2 (R-L-K,Res+Cas)	0.075	no
GERMAN	23.46±2.8 (R)	23.46±2.7 (K-R-N,Sta)	0.999	no
GLASS	30.41±6.3 (R)	30.16±6.5 (R-L,Res+Cas)	0.796	no
HEPATITIS	14.93±6.9 (N)	14.01±5.7 (K-R-L-N,Sta)	0.310	no
IONOSPHERE	8.15±2.6 (R)	6.79±3.0 (R-K-N,Res+Cas)	0.001	yes
IRIS	4.23±3.4 (K)	3.22±2.9 (N-L-R,Res+Cas)	0.030	yes
SONAR	13.12±5.2 (R)	11.29±3.9 (K-L-R,Cas)	0.003	yes
SURVIVAL	25.37±4.9 (N)	24.59±4.5 (K-N,Sta)	0.209	no
VEHICLE	19.09±2.9 (N)	17.55±2.5 (N-K-R-L,Cas)	0.000	yes
VOTES	4.21±2.3 (N)	3.72±1.9 (N-R,Cas)	0.086	no
WINE	1.56±2.0 (L)	1.05±1.5 (L-R,Res+Cas)	0.036	yes

Table 5.6: Errors of the best single model vs. the best ensemble. The reported errors are classification errors±std, followed by the configuration with the corresponding ensemble learning method. The last two columns are the values from the paired t-test.

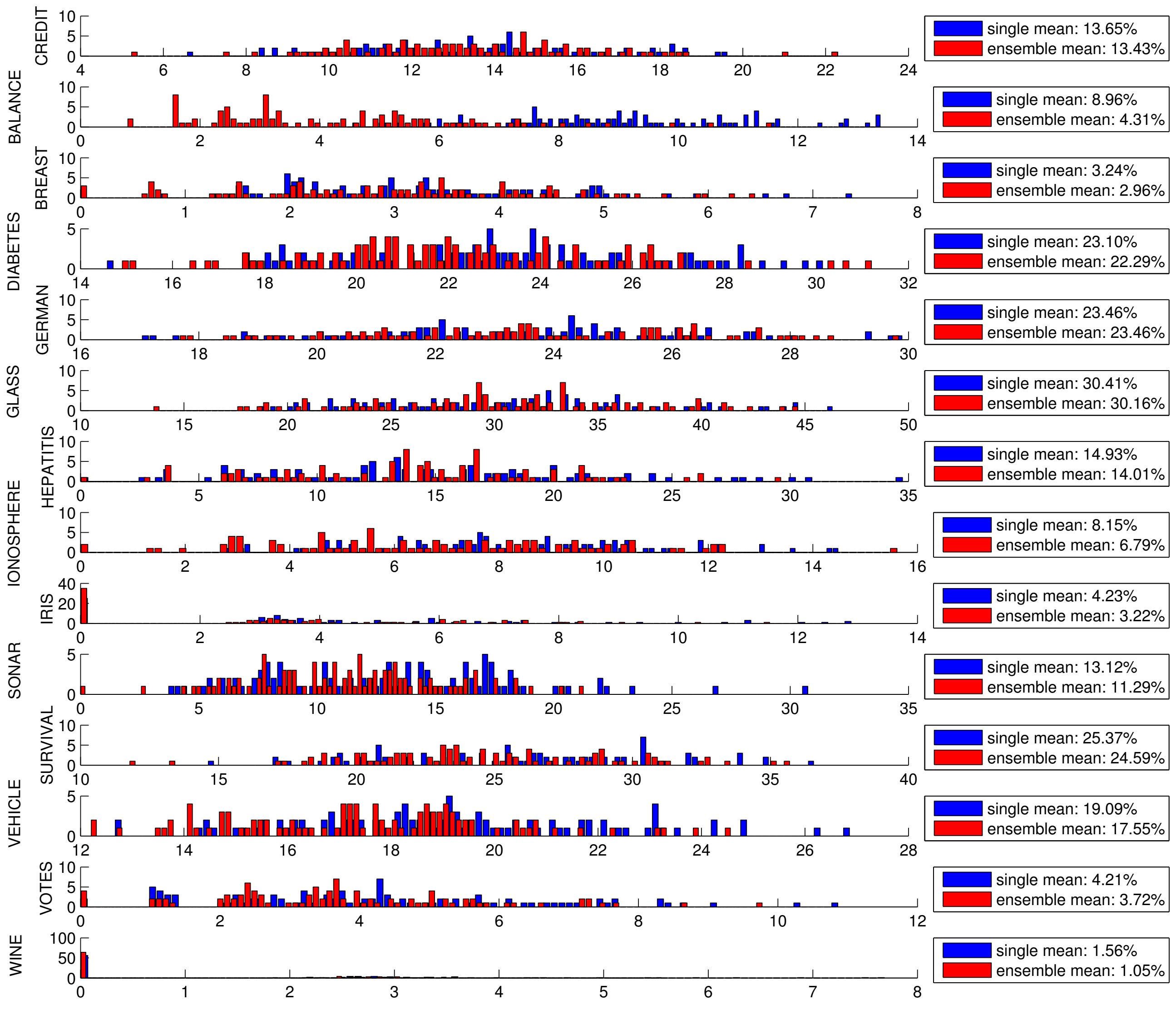


Figure 5.4: Distribution of the errors. Best single errors versus best ensemble, each result of the 100 test splits is plotted here. Ensemble learning is the clear winner on the BALANCE data set, in all other there is not clear, who the winner is. A high standard deviation of the errors is present on all data sets.

5.3 UCI LETTER - Boosting and Bagging Neural Networks

The UCI online archive [3] is a large repository of real world data sets covering various application domains. Their number of samples varies from a few tens to over 10^5. In the previous section, we picked sets with less than 1000 training samples, here we show results on the LETTER (letter recognition) data set. It is a popular alphabetic character dataset, used by many scientists ([11], [17], [40], [34]). The target is to classify attribute vectors to one of the 26 classes. Letters are described by 16 numeric values (moments of edges and counts), the number of training samples is 16000, the fixed test set has size of 4000 (the last 4000 in the training table). For details on the data set see [14].

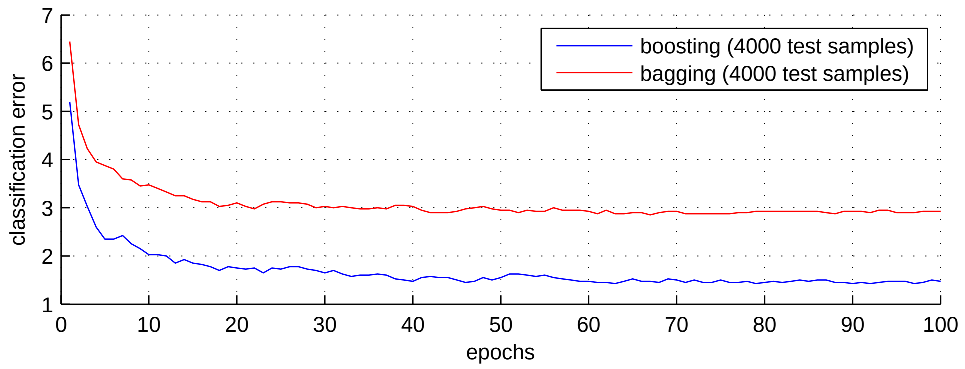


Figure 5.5: LETTER dataset. The blue curve corresponds to Boosting.M2 as described in Section 8 applied on the Letter UCI dataset with neural nets as base learner. The neural network hat 2 hidden layers with 70 and 50 neurons, the learn rate is 0.01 and no weight decay. We limit the number of training epochs to 200. The test classification error was about 1.5% after 100 boosting epochs.

The LETTER dataset is used for experiments of bagging from Section 4.2 and boosting described in Section 4.3. The base learner is a multi-layer neural network with 2 hidden layer, the number of neurons are 70 and 50. This is exactly the same net setup as proposed by Schwenk and Bengio in [34] (they use a 16-70-50-26 net). Our single net is trained with stochastic gradient descent for 160 epochs (determined by cross validation) with a constant learn rate of 0.01. The RMSE and the classification error on a 10-fold cross validation set are RMSE $= 0.135224$ and classErr $= 5.49\%$. On the 4000 predefined test samples, the net achieves RMSE $= 0.133441$ and classErr $= 5.55\%$ error. Figure 5.5 shows the learning curve of the Boosting.M2 resampling algorithm and the curve for bagging. The boosted neural net ensemble clearly outperforms the single model (1.5% versus 5.5% classification error) and also outperforms a bagged neural net ensemble. Bagging achieves an error rate of about 3%, which means a reduction of 50% compared to the single model result.

Both boosting and bagging shows outstanding performance compared to the single model result. The error rate of 1.5% from boosted neural networks coincides with the one reported in [34]. Bagging has 3% classification error, our result is superior compared to the reported 4.3% in [34]. Additionaly, we tried stacking different base models. Table 5.7 shows the results, we list the results according to their accuracy. Stacking is a loose coupling of learners, with the target to reduce the prediction RMSE, actually this holds for the stacked ensemble (RMSE $= 0.0990909$). Also the classification error of the stacked ensemble is lower (3.475%) compared to the best single model (KNN 4.625%).

All ensemble techniques boosting (1.5%), bagging (3%) and stacking (3.5%) improves the accuracy on the test set over the best single model (KNN with 4.6% error). Boosting neural nets outperforms the two other ensemble techniques.

#	model	test class. error	test RMSE	metaparameters
1	linear regression	42.575%	0.31799	$\lambda = 0.00026$
2	poly3 regression	32.175%	0.286507	$\lambda = 6.3e - 5$
3	kernel ridge regression	16.275%	0.581653	Gauss kernel, $\sigma = 2.0$, $\lambda = 0.0086$
4	neural network	5.25%	0.132987	2HL:70-50, $\eta = 0.01$, 124 epochs
5	k-nearest neighbors	4.625%	0.105756	euclidean distance, $k = 4$
	Stacking 1+2+3+4+5	3.475%	0.0990909	non-negative blending weights

Table 5.7: LETTER dataset. Performance of single models.

5.4 MNIST - Handwritten Digit Recognition

The widely used MNIST dataset of handwritten digits is a 10-class classification problem, details on the generation and various results can be found online under `http://yann.lecun.com/exdb/mnist/`. The name stems from modified and merged versions of two NIST databases (SD-1, SD-3). It is one of the larger datasets with 60000 training samples and 10000 samples in the predefined test set. The attribute vectors have 784 elements, which corresponds to a grayscale pixel image with size of 28 x 28. One of the most successful models on this dataset are convolutional neural networks, which are specially connected neural network architectures with heavy weight sharing (much more connections than weights). They have local receptive fields, similar to neurons in the brain (layer-oriented visual cortex). This means that the net structure is based on knowledge of the structure of the data. A convolutional net is designed to handle geometric data. The best known permutation-invariant model are deep belief nets (neural network with many hidden layers) pretrained with a stack of RBMs (Restricted Boltzmann Machines). Good explanations can be found in [26] or T.Hastie [18] Sec. 11.7 or in C.M.Bishop [5] Sec. 5.5.6.

We need a lot of time to tune and optimize various single models on the MNIST dataset, see Table 5.8 for the listing. The four models exhibit quite different performance. Linear regression is the fastest model to train but has the worst performance of 14.7% classification error on the test set. The next model is template based, a k-nearest neighbors model with an accuracy of 2.27%. A neural network with two hidden layers achieves a performance of 1.34% misclassified examples, the drawback of stochastic gradient descent training is the long time it takes (86 hours), but predictions can be made fast (20[s] for 10^5 samples). The last model with the best classification error is kernel ridge regression with a Gaussian kernel and 1.23% error. Tuning time for the KRR is about 10 hours. One of the interesting observations is that KRR has higher RMSE compared to KNN or NNs, but has the best accuracy in terms of misclassification error. The lowest RMSE is achieved by the neural net, see Figure 5.6.

As an example we show the learning curves of the large 2-layer neural network in Figure 5.6. We use 4-fold cross validation to prevent overfitting, but we stop the slow training after 60 epochs due to time constraints. The learning curve on the cross validation get saturated (short before overfitting). Both RMSE and classification error smoothly decrease during training. We use sigmoid output units in the net, the output is multiplied by the constant 1.2 in order to cover the output swing of $+1... - 1$.

Furthermore we tried Stacking as an combiner of the obtained results from Table 5.8. The results are shown in Table 5.9. The aim of Stacking is to reduce the RMSE of the combination, which is true for our experiments. Stacking is bad when looking at the classification error as error measure. The stacked ensemble achieves an accuracy of RMSE $= 0.0994859$ and 1.36% error, where the best single result has RMSE $= 0.1481$ and classification error of 1.23%. This evaluation shows clearly that reduction in RMSE does not necessarily correspond to an improvement in classification accuracy.

model	notes	training time (60k samples)	prediction time (10k samples)	RMSE train (4-CV)	RMSE test	class. error train (4-CV)	class. error test
LR - linear regression	regularization $\lambda = 0.0159$	1[min]	<1[s]	0.394749	0.391008	14.793%	13.83%
KNN - k-nearest neighbors	Pearson correlation, neighborhood size $k = 4$	30[min]	3[min]	0.126198	0.12802	2.27%	2.31%
NN - neural network	2 hidden, 1500 and 1200 neurons, learn rate $\eta = 0.003$, weight decay $\lambda = 0.0001$, 60 epochs	86[h]	20[s]	0.109234	0.102282	1.425%	1.34%
KRR - kernel ridge regression	regularization $\lambda = 10^{-7}$, Gauss kernel, $\sigma = 30$	10[h]	15[min]	0.1481	0.14336	1.30%	1.23%

Table 5.8: Single model results on the MNIST dataset. Training time includes searching 15 epochs for good metaparameters in LM, KNN and KRR with 4-fold cross validation. Errors on the test set are in general a little better because of the effect of retraining on all available training data by using optimized metaparameters.

models	RMSE train (4-CV)	RMSE test	class. error train	class. error test
LM + **KNN**	0.126176	0.127985	2.26667%	2.31%
LM + **KNN** + **NN**	0.104033	0.0995943	1.423%	1.37%
LM + **KNN** + **NN** + **KRR**	0.103855	0.0994859	1.426%	1.36%

Table 5.9: Stacked generalization (Section 4.4) applied to results on the MNIST dataset. Stacking hurts in terms of accuracy measured in classification error (KRR as single model achieves 1.23%). Stacking optimizes the combination RMSE.

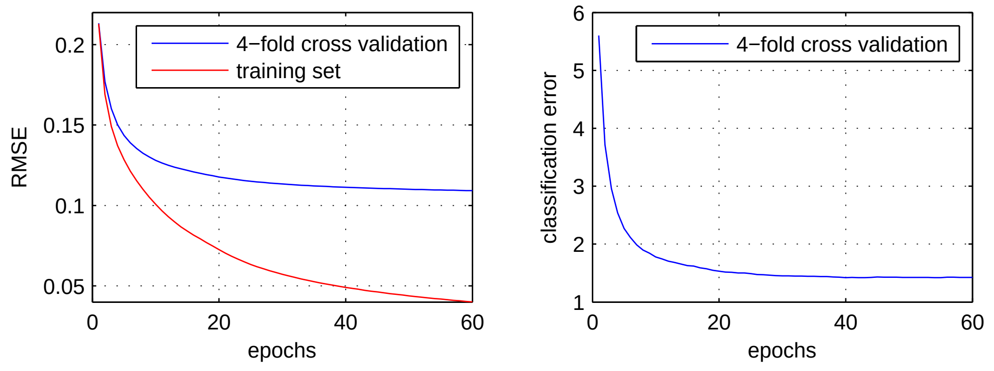
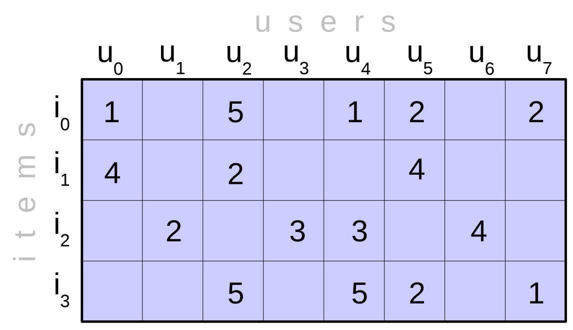


Figure 5.6: MNIST dataset. RMSE and classification error during training a neural network with 2 hidden layers (1500-1200) by stochastic gradient descent. Values are calculated from a 4-fold cross validation during training. Each epoch needs about 1 hour, 60 epochs trained in total. Retraining the model on the whole train set results in RMSE=0.102282 and 1.34% misclassification error on the 10k test set.

Chapter 6

Combining Predictions in a Recommender System

A recommender system helps users to navigate through portals or web shops with a lot of content by aggregating data generated by other users. For example, `amazon.com` provides each user with a personalized shop page on login ("your personal shop"), based on the user's past purchase data. A user is a unique person, which generates events, like purchases, ratings, bookmarks or clicks. An item is an individual product or a unique piece in an online portal (e.g. movies, jeans, watches, sunglasses). User-item predictions are personalized item scores for a user. Item-item correlations are unpersonalized relationships between items (not user dependent). A typical recommendation system consists of two parts, the first part is the prediction model, which is responsible for delivering accurate user-item predictions and item-item correlations. The prediction model should exhibit good scaling to large number of users and items. The second part is the recommendation module, this module is responsible for presenting a set of items to the user. A typical way of doing this is to predict for a given user all available items and sort their scores in descending order and take the top-K products as personalized recommendation (top-K recommendation).



users

items	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
i_0	1		5		1	2		2
i_1	4		2			4		
i_2		2		3	3		4	
i_3			5		5	2		1

Figure 6.1: The sparse user x item matrix in an recommendation system. The data sketch here is from a 5-star rating system. The target for the prediction model is to accurately predict the missing values in the matrix.

Consider the case of having a 5-star (from one the five starts, one star means bad and 5 stars means excellent) rating system on a web portal. The problem of prediction can be seen as a sparse user-item matrix (Figure 6.1). The goal is to make accurate prediction for the missing values. In collaborative filtering, the system infers a model from all available data. For example a low-rank matrix factorization leads to a very accurate prediction model. Another approach is to use content filtering for prediction, which uses meta information on user and item side, for example product sub-categories or geographic user information.

Collaborative filtering is based on data analysis. Data can be any source of user-generated information, such as purchases, ratings, clicks, bookmarks, favorite-adds, whishlist-adds, etc. We will call this data “events” in the following. Users and items become anonymous numbers in the collaborative filtering system. A user is fully described by the set of events (what the user does). The Netflix Prize [4] is a benchmark for collaborative filtering algorithms. It is the largest public available dataset, which contains 10^8 rating events, collected in a time period of 7 years. Before the Netflix Prize started in October 2006, the MovieLens dataset with about 10^6 ratings was the largest dataset for applying collaborative filtering algorithms on real world data [1].

Content filtering relies on additional data from users and items. In general, users complete a form when they create a login to the site, so we have the name, address, zip code, age, gender, preferences, etc. An item can be for example a music song, a t-shirt or a DVD. The same on item side, a shop or a portal keeps additional information of the items, such as main category, subcategory, price category, color, size, language(movies), etc. A common technique is to analyze this information by find similar users, who bought items from the same subcategory. Another approach is to use simple string matching to find products that the user like.

Research in the field of collaborative filtering has become popular since the 1M $ Netflix Prize was announced in 2006. Before that, the work of researchers in this field was focused on neighborhood models, such as item-item or user-user KNN algorithms. Herlocker et. al. give a good overview of neighborhood based approaches in [19].

The error function is crucial in measuring the accuracy of collaborative filtering algorithms. The Netflix Prize uses RMSE, which is a perfect measure in a competition due to the smooth behavior. Other error measures such as average rank or hitrate are more appropriate in evaluating the quality in a top-K recommender system.

$$RMSE = \sqrt{\frac{1}{|\mathcal{L}|} \sum_{(u,i)\in\mathcal{L}} (\widehat{r_{ui}} - r_{ui})^2} \tag{6.1}$$

The list of ratings is denoted by $\mathcal{L} = \langle (u_1, i_1), ..., (u_L, i_L) \rangle$, predicted ratings are $\widehat{r_{ui}}$ and the real ratings are r_{ui}. Other error measures such as mean absolute error or ranking based errors are discussed by Herlocker [20].

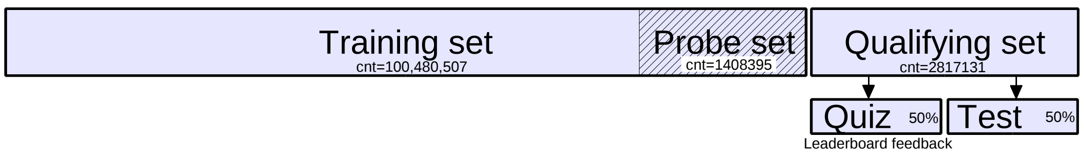


Figure 6.2: The Netflix Prize dataset. The training set size is about 100M ratings, the goal is to predict the qualifying set, which has a size of 2.8M ratings. Netflix provides a 4-digit accurate feedback from a 50% random subset, the “quiz set”. To win the competition, one must have the best RMSE score on the hidden “test set”.

The prediction of a collaborative filtering model should lie between a lower and an upper bound. In the case of 5-star ratings, the bounds are 1 and 5. One can show that predictions of a SVD model are not bounded, they can have values greater or lower than the bounds. Clipping at the bound values is not correct due to the loss of ranking possibility. When a recommendation system does top-K recommenda-

[1]http://www.grouplens.org/node/73, in 2008 groupslens published a 10M dataset (10^7 ratings)

tions, it is common to denote the predictions as “scores”. Scores can have any scale or offset, since the ranking of the scores stays the same.

In a web shop, there is often a lack of a rating system. The shop operator has a large database of the customer’s consumption history. In other words the webmaster has access to data of who bought what. This is known as binary view of the user-item rating matrix and it is similar to a “I like it” rating system. Purchase information has high-quality because the customer actually spends money on the item. The user-item matrix can be seen as completely filled with 1 for the purchased and 0 for all other entries. The matrix cells, which have value of 0 are potential candidates for recommended items. Consider the case of a full-filled matrix in the Netflix Prize dataset, the number of values in the matrix is about $500k \cdot 18k = 9 \cdot 10^9$ (9 billion numbers). The training time would be prohibitive for a SVD algorithm based on gradient descent. Training on a full-filled matrix would result in $O(U \cdot M)$ training time (standard is $O(|\mathcal{L}|)$, where $|\mathcal{L}|$ is the number of data points). U is the number of users and M is the number of items in the system. We found a simple way to do gradient descent with a SVD model on purchase data. The idea is to use a random subset of the zero values rather than all. Stochastic gradient descent runs for example 30 epochs through the training list $\mathcal{L}$ and updates the user and item features. This training can be done effectively user-wise. Now we draw per user a subset of items with target rating 0 addidionally to the 1’s. We found that 1% of $|M|$ works well for 0-sampling. Hu and Koren in [22] has done analysis on implicit feedback datasets, for example with data from IPTV [2] where users watch films with his set-top boxes. They propose an alternating least squared matrix factorization algorithm, that overcome the problem of handling the large filled user-item matrix by considering only the 1’s in the data.

During an active participation in the Netflix Price contest, Andreas Töscher (a colleague of me) and I have gained a lot of experience with various collaborative filtering algorithms (we are also part of the winning team!). The training set size is 100M, Netflix held back a qualifying set of 2.8M ratings, which should be predicted by the participants of the contest. RMSE feedback was calculated on a 50% random subset - called the quiz set, for details see Figure 6.2. One of the largest challenges is to handle the huge size of the data, hence we need algorithms with good asymptotic runtime, both in the training and in the prediction phase. For example user-user neighborhood approaches have horrible runtime bounds. Furthermore they have $O(U^2)$ memory consumption, where U is the total number of users. For the Netflix data set, we have 500k users, which leads to approx. 500GB of memory consumption when we store the upper triangle matrix of all user-user correlations in floating point accuracy. In the following, we will list and summarize collaborative filtering models in terms of speed, runtime and practical implementation challenges.

In Table 6.1 we list the most useful collaborative filtering algorithms from the Netflix Prize challenge. The RMSE column is the error that a single algorithm can achieve with good learning parameters (e.g. proper learn rate and regularization constants). Each single algorithm models the data in a different way, for example a linear combination of all single models would lead to a 0.87 RMSE score. Netflix own prediction algorithm, called “Cinematch” has an RMSE of 0.95 at the time when the competition starts in 2006 [4]. More sophisticated blending techniques lowers the RMSE below 0.87. In the following section we use the word “blending” as an acronym for combining a set of predictions.

The following list gives a technical overview of collaborative filtering algorithm listed in Table 6.1. We focus on the overview of each individual, not going deep into the training process. For further detailed explanation, we recommend to read the Netflix Prize Winner Reports [42], [24], [29].

- **KNN item-item**
  A prediction $\widehat{r_{ui}}$ in an item based k-nearest neighborhood model is made by calculating a weighted sum over k-nearest items, the weights are proportional to the correlations c_{ij} (similar to the k-nearest neighbors algorithm from Section 3.6). Therefore a precalculated item-item correlation matrix C is useful, due to the need of constant access time of any item-item correlation c_{ij}. This

[2]IPTV means Internet Protocol television

algorithm	RMSE (approx.)	training time	prediction time	memory consumption	implementation
KNN item-item	0.92	$O(U \cdot M^2)$	$O(U \cdot log(U))$	$O(M^2)$	easy-medium
KNN user-user	0.93	$O(M \cdot U^2)$	$O(M \cdot log(M))$	$O(U^2)$	easy-medium
SVD (matrix factorization)	0.90	$O(L)$	$O(1)$	$O(M + U)$	easy
AFM (asymmetric factor model)	0.92	$O(L)$	$O(1)$	$O(M)$	medium
SVD extended (SVD with asymmetric, time and frequency effects)	0.88	$O(L)$	$O(1)$	$O(M + U)$	medium-hard
RBM (Restricted Boltzmann Machine)	0.90	$O(L)$	$O(1)$	$O(M)$	hard
GE (global effects)	0.95	$O(L)$	$O(1)$	$O(M + U)$	medium
Blend of all models	< 0.87				

Table 6.1: A list of various collaborative filtering algorithms with asymptotic time and memory consumption. We add RMSE values on the Netflix Prize dataset (qualifying RMSEs) to give a value of accuracy for each of the models. The training time is needed before prediction can be made. The prediction time is defined to generate a single $\widehat{r_{ui}}$ value. Each algorithm has internal parameters/features, the size of them are listed as memory consumption. To make the algorithms run, there are various challenges, because some of them are totally new in this field. We list "how challenging" the implementation is in the last column. M is the total number of items (18k), U is the total number of users (500k) and L the total number of ratings (100M) in the data set. Red values are critical for large scale applications.

results in a memory consumption of $O(M^2)$ (M is the number of items). Training time is building the item-item correlation matrix C, which is limited to $O(U \cdot M^2)$ operations. For one prediction, the KNN selects k-best correlations to the item i, this can take up to $O(U)$ operations. Sort the list takes $O(U \cdot log(U))$, which is the upper bound in prediction time.

- **KNN user-user**
  The model is exactly the same as in the KNN item-item, but items and users are flipped. Hence the prediction and training bounds are also flipped. See Table 6.1 for the complete list. For the Netflix Prize dataset this method is unpractical due to the huge memory consumption. Here, we want to mention the possibility of learning an implicit factorization of the full user-user correlation matrix, this reduces the amount of required memory down to $O(U \cdot K)$ where K is the dimensionality of the factor matrices. This enables us to keep all the user-user correlations in memory by storing the factorized version, one particular correlation is then just a dot product of the corresponding features. For details see Töscher, Jahrer and Legenstein in [43].

- **SVD (matrix factorization)**
  This is probably the most popular collaborative filtering technique since the Netflix Prize started in 2006. A prediction is given by the dot product of a user feature $\mathbf{p}_u$ and an item feature $\mathbf{q}_i$: $\widehat{r_{ui}} = \mathbf{p}_u^T \mathbf{q}_i$, hence $O(1)$ runtime per prediction. The SVD learns two factor matrices, user features $\mathbf{P} = [\mathbf{p}_1, ..., \mathbf{p}_U]$ and item features $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_M]$ via stochastic gradient descent. In practice this means the whole training needs a few ten epochs over the whole dataset until convergence - therefore $O(L)$ training time. The model parameterizes two matrices with f rows, this leads to $O(M + U)$ memory consumption. We assume the number of f is a constant, in practice usual values are e.g. $f = 50$. Both, training and prediction time have optimal asymptotic runtime behavior, this makes the SVD an excellent candidate for large scale recommendation applications. As an extention to the excellent prediction capability of SVD models, item-item and user-user correlations can be calculated efficiently with a normalized euclidean distance measure between user or item feature vectors.

- **AFM (asymmetric factor model)**
  In the plain SVD model, a user is represented by the feature $\mathbf{q}_u$. The AFM model represents a user by the items he has rated, this means that no explicit user feature is stored as parameter. In other words, the AFM model parameterizes only item features. A so called "virtual user feature" $\mathbf{y}_u$ is given by $\mathbf{y}_u = |N(u)|^{-1/2} \sum_{i \in N(u)} \mathbf{p}_i$. The set of items, which was rated by the user u is $N(u)$. $\mathbf{p}_i$ are item-dependent features. One can show that the special normalization of the item feature sum is necessary when assuming normal-distributed feature values. This representation offers several benefits, for example integration of new data and new users without retraining the whole model. The prediction time is constant (like SVD), because we can store the precalculated virtual user features after training, $\widehat{r_{ui}} = \mathbf{y}_u^T \mathbf{q}_i$. Training time is similar to SVD, because the AFM is trained with stochastic gradient descent and a batch update on the virtual features p_i. Further explanation can be found in [42].

- **SVD extended**
  The Netflix Prize dataset comes with rating date information, this enables us to add additional user and item features, based on the time and rating frequency. We define as frequency the number of votes a user gives on a particular day t. Integration of this information is not straight forward, because for each additional feature the learn rate and regularization parameters has to be set correctly by optimizing them on a validation set. Training time rises by a constant, therefore the same asymptotic complexity as plain SVD: $O(L)$. The same applies for prediction time and memory consumption. Large extended SVD models, (called SBRAMF and extensions in [42]), have shown outstanding accuracy over the rest of collaborative filtering algorithms. They are specialized SVD models and need a lot of effort in training and tuning various meta-parameters.

- **RBM (Restricted Boltzmann Machine)**
  In general, a Boltzmann machine is a stochastic generative model. The restricted Boltzmann machine [32] is a neural network with one input layer and one hidden layer. Neither the visible nor the hidden units have connections to itself, which means that the net has no recurrences. For collaborative filtering, the number of visible units are the number of items in the system. The number of hidden units represents the number of features, each user is mapped to a low-dimensional representation in the hidden layer. Learning works well with contrastive divergence learning [32]. Prediction complexity is constant, because the probabilities of the hidden layer can be precalculated user-wise. This leads to a simple dot product enclosed by a sigmoid function for generating recommendations (same complexity as SVD). The accuracy of RBMs applied on collaborative filtering problems are superior compared to AFMs because of the non-linearity. Training is performed user-wise and converges after a few ten epochs.

- **GE (global effects)**
  Global effects [42] are based on user and item features, such as support (number of votes), mean rating, mean standard deviation, mean rating date, etc. The idea of global effects is to calculate "hand-designed" dependent features, which is equivalent to a SVD with either fixed item or fixed user features. The RMSE of 16 global effects applied to the Netflix Data is about 0.95. Global effects can be effective when applied to residuals of other algorithms. A detailed explanation can be found in the Netflix Prize Winner Report [42].

- **Combinations**
  A popular way of combining algorithms is residual training. We found that item-item KNNs are most effective, when they are applied on residuals on RBMs. This means the KNN and its correlations are calculated not on raw ratings, but on ratings from which the predictions of the RBM are subtracted (similar to the ensemble learning technique "training on residuals" from Section 4.6). When constructing such a residual chain, the ensemble of collaborative filtering (CF) results becomes more diverse, which is good for the final blend. The final blender has access to all the predictors generated by various CF models on various residuals of other models. Another way of combination is to insert predictions from another model into the training list of the current, but this approach was not very successful, in our experiments.

Combining different kinds of CF algorithms is the motivation of the next chapters. We use predictors (Table 6.2) from the Netflix Prize dataset as input. Many machine learning models are not directly applicable because of the huge number of samples. Blending is done on the probe dataset (1.4M samples), which is a hold-out set of the 100M training set. The probe set is not a random subset of the training data, it reflects the optimization goal by taking per user the latest 3 ratings. Furthermore every user was sampled with equal probability. We begin with simple methods like linear blending, then we move to binned blending, which is the application of learners on structured subsets. Furthermore, we find the most successful blending schema is a blend with a neural network. Additionally, the computational costly k-nearest neighbors and the kernel ridge regression algorithm are applied to blend predictions by averaging multiple models trained on small random subsets. Finally we compare the results.

6.1 Notation

Prediction blending is a supervised machine learning problem, therefore we adopt the notation defined in Section 2.2. Data matrix $\mathbf{X}$: NxF matrix of predictions, where N is the number of samples and F the number of predictors. To this matrix, we always assume to add a constant column. This is needed for linear regression to work correctly on uncentered targets. Target value $\mathbf{Y}$: Nx1 vector, in our case: Integer ratings from 1 to 5. The blending algorithm is formally a function $\Omega : \mathbb{R}^F \mapsto \mathbb{R}$, the input is a vector of individual predictions, the output is a scalar.

cnt	name	probe RMSE	description
-	-	0.9520	Netflix's "Cinematch" [4] algorithm is the baseline
1	SVD-1	0.9074	A rating matrix factorization with 300 features, trained with stochastic gradient descent, 158 epochs, learn rate $\eta = 8e-4$, regularization $\lambda = 0.01$, based on item-mean centered data.
2	SVD-2	0.9172	A rating matrix factorization with 20 features, trained with stochastic gradient descent, 158 epochs, learn rate $\eta = 0.002$, regularization $\lambda = 0.02$, based on item-mean centered data.
3	SVD-3	0.9033	A rating matrix factorization with 1000 features and adaptive user features (item-item similarity correction). 158 epochs, learn rate $\eta = 0.001$, regularization $\lambda = 0.015$.
4	SVD-4	0.8871	An extended rating matrix factorization with 150 features. Learnrates and regularizations are automatically tuned in order to minimize the RMSE on the probe set, thus they have individual values per trained parameters, details can be found in [42].
5	AFM-1	0.9362	An asymmetric factor model, where the user is expressed via the rated movies. 200 features were used, learn rate $\eta = 1e-3$, regularization $\lambda = 1e-3$, we multiply all η with 0.95 from epoch 30, 120 epochs were trained in total.
6	AFM-2	0.9231	An asymmetric factor model, where the user is expressed via the rated movies. 2000 features were used, learn rate $\eta = 1e-3$, regularization $\lambda = 2e-3$, 23 epochs were trained in total. Based on residuals from KNN-5.
7	AFM-3	0.9340	An asymmetric factor model, where the user is expressed via the rated movies. 40 features were used, learn rate $\eta = 1e-4$, regularization $\lambda = 1e-3$, 96 epochs were trained in total.
8	AFM-4	0.9391	An asymmetric factor model, where the user is expressed via the rated movies. 900 features were used, learn rate $\eta = 1e-3$, regularization $\lambda = 1e-2$, 43 epochs were trained in total.
9	RBM-1	0.9493	Restricted Boltzmann Machine with discrete input units. The number of hidden units is 10. Learnrate $\eta = 0.002$ and regularization $\lambda = 0.0002$.
10	RBM-2	0.9123	Restricted Boltzmann Machine with discrete input units. The number of hidden units is 250. Learnrate $\eta = 0.002$ and regularization $\lambda = 0.0004$.
11	KNN-1	0.9110	A item-item k-nearest neighbors model, similarity measure is the Pearson correlations. We use $k = 24$ neighbors, this model is based on residuals from AFM-1.
12	KNN-2	0.8904	A item-item k-nearest neighbors model, similarity measure is the set correlation, explained in [42]. We use $k = 122$ neighbors, this model is based on residuals from a chain of algorithms: RBM-KNN-GE(with time).
13	KNN-3	0.8970	A item-item k-nearest neighbors model, similarity measure is the Pearson correlation. We use $k = 55$ neighbors, this model is based on residuals of a RBM with 150 hidden units.
14	KNN-4	0.9463	A item-item k-nearest neighbors model, similarity measure is the Pearson correlation. We use $k = 21$ neighbors, this model is based on residuals from GE-2.
15	GE-1	0.9079	Global Effects model. Based on residuals from KNN-1, hence this result is the third algorithm in a chain of training on residuals.
16	GE-2	0.9710	Global Effects model, trained on raw ratings.
17	GE-3	0.9443	Global Effects model, trained on residuals from KNN-5.
18	GE-4	0.9209	Global Effects model with time integration, trained on residuals from AFM-2.

Table 6.2: This results are the input to various blenders. The table summarizes major results from the Netflix Prize competition. Please note that the RMSE on the probe set is 0.0030 to 0.0090 worse than the quiz RMSE. For example a 0.9100 probe score, would result 0.9030 quiz score, because the probe set is hold back when we predict the probe set itself. Probe set and quiz set have equal statistic properties.

6.2 Linear Combination

Assuming a quadratic error function, optimal linear combination weights $\mathbf{w}$ (vector of length N) can be obtained by solving the least squares problem.

$$\mathbf{X} \cdot \mathbf{w} = \mathbf{Y} \tag{6.2}$$

This is exactly the linear model from section 3.3. For any input vector $\mathbf{x}$, the blending model is given by $\Omega(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = x_1w_1 + x_2w_2 + ... + x_Nw_N$. We extend this model with L2 regularization (see equation 3.8), also called ridge regression. This has two reasons. L2 is the preferred regularization when the error function is quadratic. And second, when we blend many predictions, the effect of overfitting occurs, which means that the test error increases while the train error goes down. Cross-validation can be used in order to get a good estimate of the proper ridge regression constant λ.

6.3 Histogram based Linear Combination

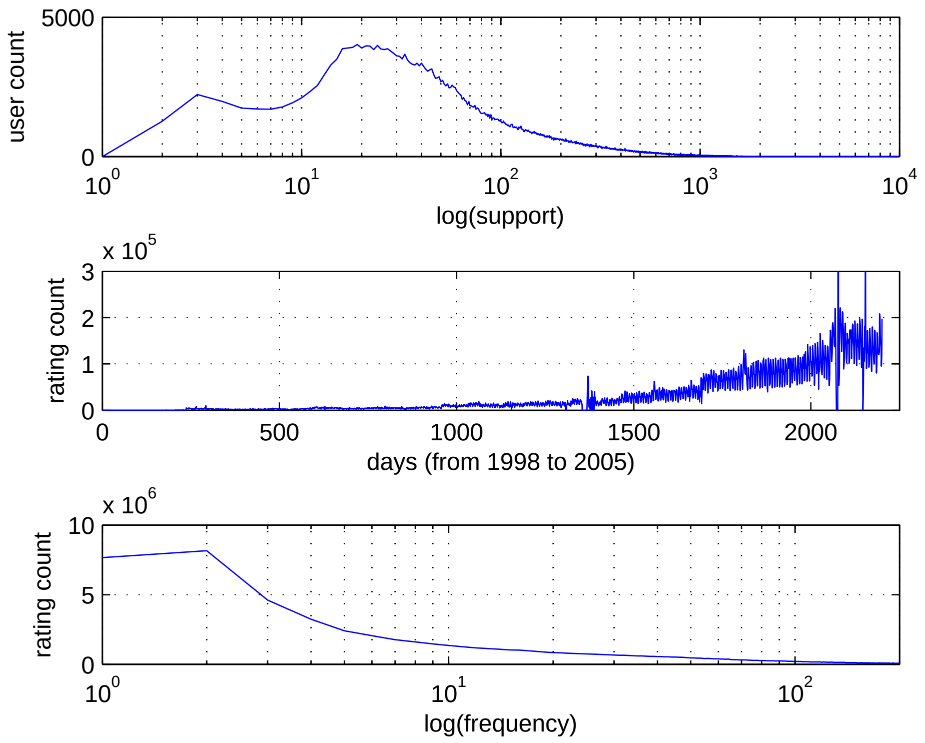


Figure 6.3: Statistical properties in the Netflix Prize data set. The first diagram shows the distribution of support. We understand under the support the number of votes, given from a user u. The support histogram has the mode on 20 votes/user, where the average number is 200. The second diagram shows the distribution of ratings over the time, most ratings were collected at the end of the time period. The aforementioned frequency histogram is shown at the bottom, most users gave one or two ratings per day.

Every CF algorithm is trained on the data set with the hold-out set removed. This hold-out set is used to determine the stopping time in gradient descent based algorithms (such as SVD). Blending is performed by training a supervised regression model on the hold-out set. In the Netflix data set, this set is called “probe set” and consists of 1.4M ratings. Due to the huge size of the probe set, one can divide the set into disjoint B bins and calculate separate blending weights per bin. Starting from pure linear regression, the blending weights $\mathbf{w}$ become $\mathbf{w}_b$, where b is the corresponding bin to the predicted rating $\widehat{r_{ui}}$. Each bin should approximately have the same number of ratings.

Figure 6.3 summarizes the main effects in the Netflix data set. This insight in the data motivates the following three criteria for binned linear regression, i.e. the training set can be split by using a histogram on one of the following criteria.

- Support : the number of votes from a user. The blender can now overweight particular predictions dependent on how many rating the user gave. RBMs are prone to receive high weight when the user has only a few votes in the data. SVDs are overweighted when many information from a user is available.
- Time : when the rating r_{ui} was performed. Predictions are mixed together with time dependency. When using this binning criteria, the blender can easily model time-dependent blending.
- Frequency : the number of ratings from a user at time t. The time t is measured in days from 1998. This criteria enables the blender to be selective, based on the user's rating day. The blender has the ability to give predictions other weights when a user votes many times on a particular day. The explanation in different number of votes on one day can be that all people in a household using the same account (see Figure 6.3).

A prediction is given by

$$\widehat{r_{ui}} = \mathbf{x}^T \mathbf{w}_b \tag{6.3}$$

This means we calculate B separate blending weights $\mathbf{w}_b$. For example, when we divide the Netflix probe-set into 5 support-bins (approximately equal sized bins) following formula gives the bin b. $|N(u)|$ denotes the number of rating of a user u.

$$b = \begin{cases} 1, & |N(u)| < 34 \\ 2, & 35 \leq |N(u)| < 70 \\ 3, & 71 \leq |N(u)| < 146 \\ 4, & 147 \leq |N(u)| < 321 \\ 5, & \text{else} \end{cases} \tag{6.4}$$

6.4 Neural Network

A neural network is a function approximator. Small nets can be trained efficiently on huge data sets, therefore it is very suitable for a nonlinear blending algorithm. We use a network with one hidden layer. The training of neural nets is sketched in Section 3.5. The output neuron has again a sigmoid activation function with an output swing of $[-1...+1]$, to generate rating predictions in the range of $[1...5]$ we use a simple output transformation, the output gets multiplied by 3.6 and the constant 3.0 is added (works well on our experiments). The learning rate η is initialized with 0.0005 and every epoch the constant 10^{-6} is subtracted. No weight decay is used. Pure stochastic gradient descent is applied to train the weights, no batch training or momentum term is used. In the result section, the number of neurons in the hidden layer is reported.

6.5 Tree Blending - BGBDT

Decision trees are known as a good supervised learning tool. Our idea is to use them in order to mix predictions. The main drawback of a single decision tree is the moderate accuracy. Breiman [6] shows how bagging can be extended to improve the accuracy of decision trees. The discretized output function of a tree limits the ability of modelling a smooth function. The number of possible output values of a tree corresponds to the number of leafs. For regression problems, such as blending predictions, this is a big disadvantage. Blending predictions should result in a smooth function. In 1999, Jerome Friedman introduced an interesting idea to improve the accuracy of a learning machine. He called his technique "Stochastic Gradient Boosting" [15], [16]. The core idea is to train the learner on residuals from itself, like the ensemble learning technique "Training on residuals" described in Section 4.6. The main difference is that not the full residual is taken, but only a fraction from it (controlled by the learn rate η). We described this approach in the previous Section 3.8 (GBDT).

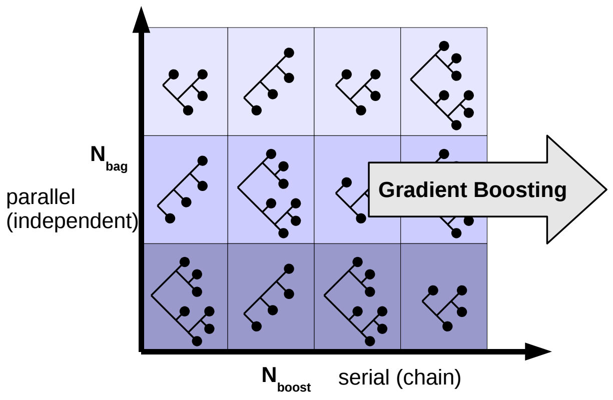


Figure 6.4: Bagged Gradient Boosted Decision Tree. A prediction from the BGBDT consists of results from $N_{bag} \cdot N_{boost}$ single decision trees. The Bagging and Boosting idea greatly improves the accuracy of the blended predictions.

Additionally we found it useful to add the random subspace idea in the determination of the optimal split on each node (as described in Section 3.8). Finally, we end up with a tree blending technique that combines the benefits from Bagging, Gradient Boosting and Random Subspace selection. The algorithm is called now BGBDT - Bagged Gradient Boosted Decision Tree (see Figure 6.4). The prediction of a BGBDT is much smoother than from a single tree. We use the bagging size N_{bag} and the number N_{boost} of boosting steps in the chain. One prediction of test feature is done internally with evaluation of $N_{bag} \cdot N_{boost}$ single trees. Good parameters are $N_{bag} = 128$, $N_{boost} = 1000$, $N_{subspace} = 2$ and $\eta = 0.01$ when blending about 20 predictions. For the analysis on the Netflix Prize dataset we use the user support, movie support and the raw time as additional input features. Further information of the BGBDT approach can be found in [42].

6.6 Kernel Ridge Regression Blending

The Section 3.7 describes the KRR algorithm in detail. We use it for blending CF predictions. KRR has a training time complexity of $O(N^3)$ and space requirements of $O(N^2)$, hence it is impossible to train on 1.4M data points. To make it work, we use a small subset of the data to train the KRR model. Doing this multiple times and average all outcomes, we obtain an accurate blending model. We evaluate the impact of the subset size with respect to the accuracy measured in RMSE. The Gauss kernel was applied successfully. We tune the kernel width σ and the regularization constant λ with cross validation.

6.7 K-Nearest Neighbors Blending

The template-based KNN algorithm from Section 3.6 is applied to blending predictions. KNN is very slow when the training set is large, because all distances are calculated on the fly (training and prediction time rise with $O(N^2)$). Therefore we use again a small subset of the train set to build the model. Averaging over many random subsets delivers our final prediction. Again, we investigate the impact of the size of the subset. We found out that KNN has very bad accuracy compared to all other blending techniques. One possible explanation is that predictions on new test samples are based on a weighted average of the features from the training set, which are themselves predictions. KNN is not able to deliver a successful blending model.

6.8 Results

Submissions on the Netflix Prize homepage (`www.netflixprize.com`) are not longer available, the constest has finished on July, 26th 2009. The winning team with the name "BellKor's Pragmatic Chaos" won with a leaderboard score of 10.09%. Since the submission system and the qualifying set is not available at this time point our experimental analysis relies only on probe set predictions. The process of training a CF algorithm on the Netflix Prize dataset is shows in Algorithm 10. In the first step of training, we remove the probe set from the dataset. This probe set acts as a hold out set and is a very reliable performance measure during training, because of its huge size of 1408395 ratings.

```
  Input: A CF algorithm, the data in $\mathcal{L}$, a probe set $\mathbf{p}$ (the hold-out set)
  Output: The prediction for the probe and the qualifying set
1 Remove the probe set $\mathbf{p}$ from $\mathcal{L}$
2 Train the CF algorithm, by optimizing the RMSE on $\mathbf{p}$. For gradient based algorithms it means
  that the stopping point (the #epoch) is optimized, for metaparater optimization we search a few
  ten epochs with a coordinate based optimizer.
3 Predict the probe set $\mathbf{p}$
4 Add the probe set $\mathbf{p}$ into $\mathcal{L}$
5 Re-train the CF algorithm, with found metaparameters from line 2.
6 Predict the qualifying set
```

Algorithm 10: Overview of the training process, used within the Netflix Prize competition

We decide to evaluate blending methods based on a 50% random subset of the 1.4M probe set. The probe set is splited into two halves, one half for training and the other half for testing (Figure 6.5). All reported RMSE values in the next Figures and Tables are evaluated on this test set. It is never touched during training the blending models.

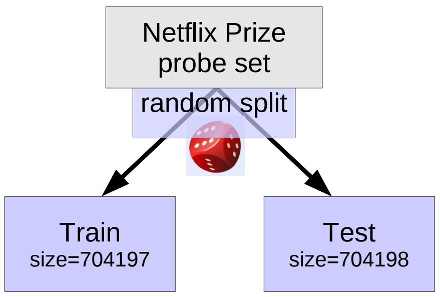


Figure 6.5: Split the Netflix probe set randomly into two sets, one for training one for test.

The described techniques are now trained on the 700k training set and evaluated on the test set, the relative difference in RMSE values seems to be marginal. Keep in mind that the Netflix Prize contest awards the winner 1M \$ for an 10% improvement in RMSE. The baseline was set by Netflix's internal collaborative filtering system "Cinematch" with a RMSE of 0.9525 on the Netflix quiz set (see Figure 6.2). The 10% improvement is equivalent to an RMSE of 0.8563. The leaderboard shows the submission feedback with an accuracy of 4 digits after the comma. Which means that a RMSE of 0.8600 or 0.8599 makes a difference. At the end of the Netflix Prize contest, the winning team has the better score on a hidden test set. The second placed team "The Ensemble" shows in his blog [3] that the rounded RMSE score of both teams were the same, the first-submitter rule determines the winner (both teams had a test set RMSE of 0.8567, which corresponds to 10.06% improvement).

Our reported RMSE values have at least 4 digit accuracy, a significance improved blending technique lowers the RMSE score compared to linear regression (RMSE $= 0.8752$ used as baseline) at 0.001. A marginal improvement results in a 0.0001...0.0002 lower RMSE. For example the KNN algorithm has significantly inferior performance (RMSE 0.884), this shows that KNN is the wrong supervised learning technique for blending CF predictions.

Linear Blending

The linear blending technique is used to have a baseline estimate of the RMSE on the test set. This is done with regularized linear regression, the ridge regression constant λ is set to minimize the RMSE on a cross validation set. The constant λ will have very small values, because the Linear Regression weights are estimated using a training set of 700k samples. The Table 6.3 shows the weights, from each of the input predictors (from Table 6.2) including the constant input. The largest weight has the constant 1 input because of the uncentered target values, the weight of 3.673 corresponds to the mean value of the targets. The second largest weight has the strongest model (with the lowest RMSE), SVD-4 has a weight of 0.237. Due to the nature of linear regression, the weights are not restricted to be positive. Some of the predictors receive negative weights, this can be interpreted as negative-compensation of a particular effect in the data.

SVD-1 (0.9074)	SVD-2 (0.9172)	SVD-3 (0.9033)	SVD-4 (0.8871)	AFM-1 (0.9362)	AFM-2 (0.9231)	AFM-3 (0.9340)	AFM-4 (0.9391)	RBM-1 (0.9493)	RBM-2 (0.9123)	KNN-1 (0.9110)	KNN-2 (0.8904)	KNN-3 (0.8970)	KNN-4 (0.9463)	GE-1 (0.9079)	GE-2 (0.9710)	GE-3 (0.9443)	GE-4 (0.9209)	const. 1
−0.009	0.093	0.077	0.237	−0.082	−0.083	−0.077	0.088	0.026	0.063	0.028	0.276	−0.102	0.004	0.099	−0.003	−0.074	0.173	3.673

Table 6.3: Blending weights of an optimal linear combination. This leads to 0.875277 RMSE on the test set. The RMSE on the 4-fold cross validation set is 0.8755. The number in the brackets is the RMSE per model.

Dataset Binning based on Support, Date and Frequency

This is linear blending on predefined subsets of training data, as described above. For each of the training and test samples we have the support (number of ratings), the date (day of the rating) and the frequency (number of ratings per day). Based on this information we split the data into 2, 5, 10 or 20 nearly equal sized bins. We compare these three blending methods: linear regression (LR), polynomial regression with and without cross terms (PR) and neural networks (NN). We tune the meta-parameters like regularization or number of epochs in neural network training on a 4-fold cross validation set separately per

[3] `http://www.the-ensemble.com/`

bin. The reported RMSE values are those on the test set (see Table 6.4 for all results), the RMSE on the internal cross validation set is the small numbers in brackets below.

The column "no bins" from Table 6.4 shows results of the investigated models without binning the dataset. This means that all 700k samples are used for training. The neural network with 30 neurons in the hidden layer performs best on all data (RMSE $= 0.873934$) second best is polynomial regression with 2nd order and all cross terms (RMSE $= 0.874488$), linear regression as baseline has RMSE $= 0.875277$. Based on the outcomes from Table 6.4 we see that the support binning criteria delivers best results. All investigated models have lowest RMSEs on the support bins. Most of the models perform best when dividing the data to 2 bins. The winner is the neural net on 2 support bins with RMSE $= 0.87365$. The best result with date-based binning is achieved by the neural net on 2 bins with RMSE $= 0.87415$. The best result with frequency-based binning is again the neural net on 2 bins with RMSE $= 0.87408$. Additionally we tried kernel ridge regression with a Gauss kernel on 20 support bins, because of the computational complexity the KRR can not applied to fewer bins. The KRR results in a RMSE of 0.87432, which is better than all other models on 20 bins.

Too many bins increase the RMSE. The best result are obtained by 2 bins. Neural networks are the best learners in our experimental analysis on binned based. The KRR model has very promising result on 20 bins, unfortunately it can not directly applied on larger training data.

Neural Network Blending

The binned data analysis in the previous section shows that neural networks are the best performing blending models. Here we investigate various number of neurons in the hidden layer and two hidden layers. Table 6.5 shows the outcome. We tried one and two hidden layer, the RMSEs with one hidden layer are slightly better. For one hidden layer we get the best results with 500 neurons (RMSE $=$ 0.873848). The performance can be enhanced by adding the $log(\#userVotes)$ as additional input, this lowers the RMSE to 0.873334 (generated by a 1-layer net with 100 neurons). Too many neurons in the hidden layer perform bad, for example with 500 neurons the net has the minimum 4-fold cross validation error on training epoch 69, maybe there is a need for a small weight decay, we have not tried this. The training time for the best individual net is $496 \cdot 16[s] = 7936[s]$, but the time for retraining is not included. Neural networks with the support as additional input achieve the lowest RMSE values in our experimental evaluation of blending techniques applied to collaborative filtering algorithms (single models).

Tree Blending

The analysis of the Bagged Gradient Boosted Decision Tree (BGBDT) model from Section 6.5 is done by varying the bagging size, the subspace size and the learning rate in the gradient boosting approach. We try to find the dependence of these parameters to the RMSE value of the blend. Figure 6.6 shows some learning curves of the BGBDT. The plot on the left shows the variation of the learning rate λ. Larger learning rate leads to longer training, which is obvious. The full line is the RMSE on the out-of-bag estimate ("oob" in the legend of Figure 6.6), the dotted line corresponds to the RMSE on the training set. Early overfitting occurs with large learning rates. Minimum out-of-bag RMSE with $\lambda = 0.02$ is on epoch=1126, with $\lambda = 0.01$ on epoch 1998 and with $\lambda = 0.005$ on epoch 4209. In the Figures in the middle and on the right we clamp $\lambda = 0.02$ and vary the subspace size (denoted with "size" in the legend). Subspace size is the number of features considered in each node to find the optimal split. The learning curves are nearly equal in the middle Figure, the right one shows the magnified region of interest. Subspace size of 4 works best on our experiments. This is close to the recommended subspace size of $\sqrt{\#features}$ from Breiman [8]. In Table 6.6 we list the detailed outcomes, when we either fix the subspace size or the learning rate. The influence of the bagging size N_{bag} is considered in Table 6.7, larger bagging size improves the prediction RMSE on the test set.

type of binning	model	no bins	2 bins	5 bins	10 bins	20 bins
support	linear	0.875277 (T:0.875516)	0.874877 (T:0.87517)	0.874741 (T:0.8750)	0.874744 (T:0.87499)	0.87485 (T:0.87513)
support	poly2	0.874911 (T:0.875167)	0.87444 (T:0.87475)	0.874309 (T:0.87459)	0.87434 (T:0.87463)	0.874567 (T:0.87495)
support	poly3	0.874754 (T:0.875002)	0.874299 (T:0.8746)	0.874188 (T:0.87448)	0.87426 (T:0.87454)	0.874562 (T:0.87498)
support	poly5	0.874812 (T:0.874991)	0.874306 (T:0.87459)	0.874227 (T:0.87452)	0.874322 (T:0.87467)	0.874713 (T:0.87519)
support	poly2-cross	0.874488 (T:0.874791)	0.87401 (T:0.87442)	0.873989 (T:0.87436)	0.87413 (T:0.87461)	0.87452 (T:0.87523)
support	NN-1HL:30	0.873934 (T:0.874244)	0.87365 (T:0.87399)	0.873828 (T:0.87424)	0.874199 (T:0.87458)	0.87462 (T:0.87524)
support	KRR gauss					0.87432 (T:0.87465)
date	linear		0.875212 (T:0.87545)	0.875195 (T:0.87541)	0.87527 (T:0.87544)	0.87537 (T:0.87558)
date	poly2		0.87482 (T:0.8751)	0.874816 (T:0.8751)	0.874924 (T:0.87514)	0.87512 (T:0.87546)
date	poly3		0.874685 (T:0.87492)	0.87471 (T:0.87489)	0.87488 (T:0.87509)	0.87521 (T:0.8755)
date	poly5		0.874713 (T:0.87492)	0.874771 (T:0.87496)	0.87499 (T:0.87522)	0.87544 (T:0.8758)
date	poly2-cross		0.87444 (T:0.87478)	0.874608 (T:0.87488)	0.87478 (T:0.87514)	0.87517 (T:0.87567)
date	NN-1HL:30		0.87415 (T:0.87446)	0.874325 (T:0.87481)	0.874736 (T:0.87507)	0.87533 (T:0.87572)
frequency	linear		0.87518 (T:0.87537)	0.87510 (T:0.87521)	0.87512 (T:0.8752)	0.87517 (T:0.87531)
frequency	poly2		0.874817 (T:0.87498)	0.874747 (T:0.87492)	0.874812 (T:0.87496)	0.874947 (T:0.87514)
frequency	poly3		0.874659 (T:0.87482)	0.874615 (T:0.87472)	0.87473 (T:0.87486)	0.874918 (T:0.87501)
frequency	poly5		0.874650 (T:0.8748)	0.87466 (T:0.87478)	0.87483 (T:0.875)	0.87512 (T:0.87525)
frequency	poly2-cross		0.874415 (T:0.87456)	0.874509 (T:0.87467)	0.874586 (T:0.87482)	0.874985 (T:0.87524)
frequency	NN-1HL:30		0.87408 (T:0.87427)	0.87425 (T:0.87448)	0.874488 (T:0.87475)	0.87476 (T:0.87501)

Table 6.4: RMSE on the test set from support/date/frequency binned blending. Models are trained on a 50% random half of the probe set, the other half is used as a test set. The reported values are RMSEs on the test set. Each particular model on each of the bins was optimized on 4-fold cross validation set. The small numbers in the brackets are the RMSEs on the cross validation set.

Our results suggest that smaller learning rates and larger bagging sizes improve the RMSE. The optimal subspace size dependens on the data, a good value to start with is the square root of the number of features.

neurons per layer	RMSE train (4-CV)	RMSE test	#epochs	#weights	time/epoch
1HL:20	0.874329	0.874019	497	421	4[s]
1HL:30	0.874285	0.873955	496	631	5[s]
1HL:100	0.874196	0.873916	498	2101	15[s]
1HL:200	0.874128	0.873931	498	4201	27[s]
1HL:500	0.874122	0.873848	498	10501	59[s]
2HL:20-20	0.874343	0.874056	490	841	6[s]
2HL:50-50	0.874335	0.874053	486	3601	22[s]
1HL:30 +support input	0.873737	0.873417	498	661	5[s]
1HL:50 +support input	0.873707	0.873385	496	1101	10[s]
1HL:100 +support input	0.873728	0.873334	496	2201	16[s]
1HL:500 +support input	0.874699	0.874731	69	11001	61[s]

Table 6.5: Results from different neural network blends. Training parameters: $\eta_{init} = 0.0005$, $\eta^{(-)} = 1e-6$, the tanh unit in the output layer is translated by: $out \cdot 3.6 + 3.0$. This output transformation is necessary, because target ratings have values from 1 to 5. Training was stopped when the RMSE on a 4-fold cross validation set begin to rise. The time/epoch denoted the time, which is needed to train one epoch in a 4-fold CV setup. The total number of weights in the net is #weights. Prediction on the test set are done with the retrained net. We used a Intel Core2 Duo machine with 2.53GHz in generating these results.

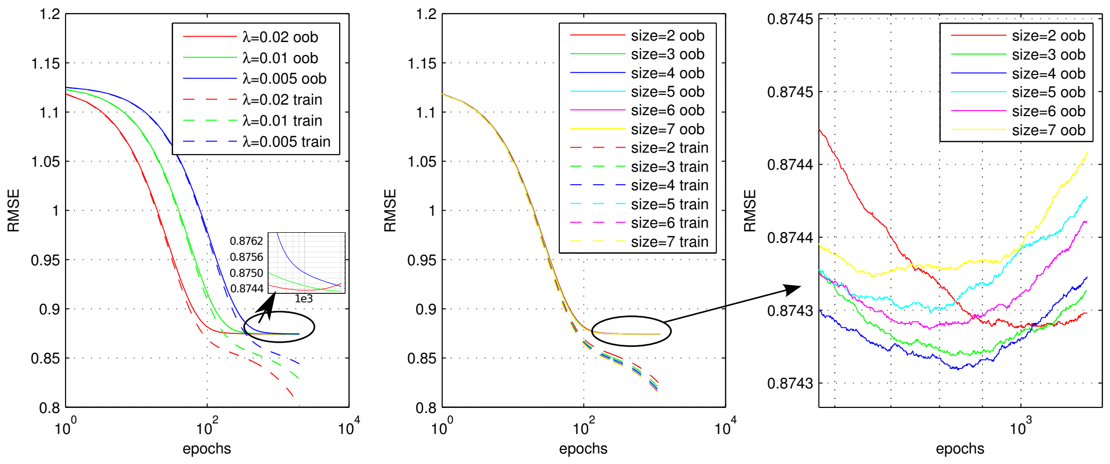


Figure 6.6: BGBDT - Bagged Gradient Boosted Decision Trees applied to blend CF predictions. "oob" means out-of-bag estimate. Bagging size is fixed to $N_{bag} = 32$. The left Figure shows the influence of the learning rate on the RMSE, feature subspace size is fixed to 2. In the Figure in the middle and right we fix the learning rate to $\lambda = 0.02$.

Kernel Ridge Regression Blending

Kernel ridge regression is not directly applicable to the training set of 700k samples. Based on the algorithm from Section 3.7, the gram matrix with size of NxN, where N is the sample size, must be inverted. A state of the art PC with 16GB of main memory can store and invert matrices up to $N = 60000$ (assume single precision floating point). We therefore use the following method: Training many KRR models on random subsets and average their predictions, the Figure 6.7 shows curves where subsets with 1% to 6% of the training set were used. The regularization constant and the width of the Gaussian kernel are optimized on a 4-fold cross validation set for every single model. Not very surprisingly, the outcome

	size=2 $\lambda = 0.02$	size=2 $\lambda = 0.01$	size=2 $\lambda = 0.005$	$\lambda = 0.02$ size=3	$\lambda = 0.02$ size=4	$\lambda = 0.02$ size=5	$\lambda = 0.02$ size=6	$\lambda = 0.02$ size=7
min. out-of bag RMSE	0.874338	0.874306	0.874258	0.874319	0.874309	0.874348	0.874337	0.874373
test RMSE	0.874036	0.874009	0.873992	0.874006	0.874020	0.874049	0.874073	0.874107

Table 6.6: BGBDT blending results on the test set. Here the learn rate (Table on the left, fix subspace size=2) and the random subspace size (Table on the right, fix learnrate 0.02) are varied. Bagging size is 32.

	bag=4	bag=32	bag=64	bag=128
min. out-of bag RMSE	0.874616	0.874273	0.874085	0.874063
test RMSE	0.874227	0.874009	0.873948	0.873945

Table 6.7: BGBDT blending results on the test set. Here, we vary the bag size. Feature subset size is 3. Learnrate is 0.02.

shows that more data is better. All models benefit from averaging over multiple runs with different data. This approach can be seen as a form of Bagging. An average of nine KRR models on 6% data achieves an RMSE of 0.8740 on the test set, which is significantly better than the linear regression baseline RMSE $= 0.8752$. The curves with 1% and 2% data show a saturation effect at about 100 averaged models, i.e. no improvement can be expected with an increase of the number of averaged models above 100.

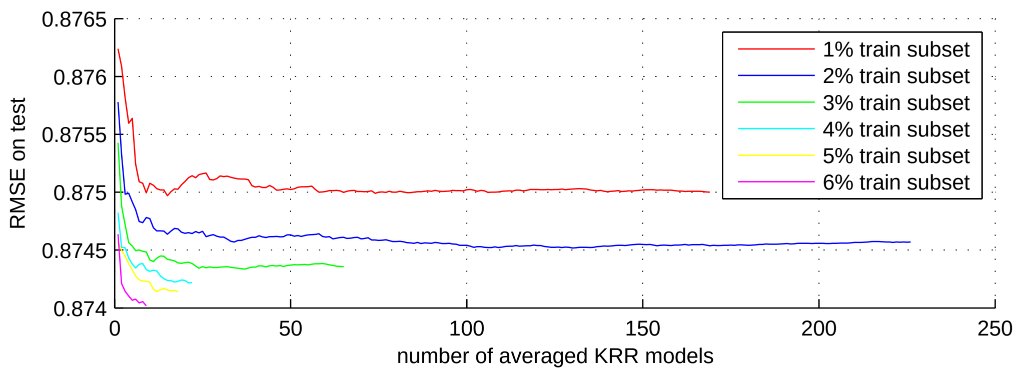


Figure 6.7: Kernel ridge regression applied to the blending of CF predictions. The KRR is trained on a random subset of the data (1%...6%). More data and more averaged models result in a lower RMSE.

K-Nearest Neighbors Blending

The runtime of the k-nearest neighbors algorithm is quadratic in N (the number of training samples). Training and metaparameters-tuning on all 700k data is too time consuming. We therefore try the same approach as in the KRR blending model. We investigate the effect of averaging many models, where each of them is trained on a random subset from of data. The results are shown in Figure 6.8. Again, more data and more averaged models are better. But the KNN shows very bad performance in terms of RMSE. The reported RMSEs are in the region of $0.885...0.884$. Recall that the linear regression baseline achieves RMSE $= 0.8752$ on the test set.

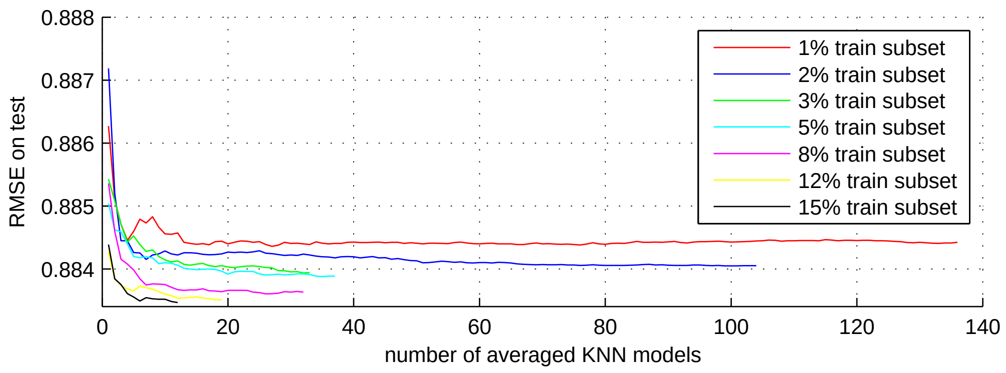


Figure 6.8: K-nearest neighbors applied to the blending of CF predictions. The KNN is trained on a random subset, we averaged the predictions from many random subsets. In each model, the neighborhood size k is optimized, typical values are $k = 200$.

Bagging with Neural Networks, Polynomial Regression and GBDT

We investigate the ability of bagging and blend stopping applied to many different models in order to blend the 18 predictions from Table 6.2. It results in a very accurate prediction in terms of RMSE. This was the motivation of this experiment. Blend stopping (from Section 4.7) was used to search for proper meta parameters or stop the training in gradient-descent based models. We use bagging [6] and the out-of-bag estimate (see Section 4.2) for the prediction of the training set. Blend stopping uses a simple linear regression of the predictors as combination strategy. The order of model training is the same as listed in Table 6.8. The models in the ensemble are neural networks, gradient boosted decision trees and polynomial regression. We use a bagging size N_{bag} of 128 (this can be run in parallel). Each individual model in listed in Table 6.8. The ensemble achieves an RMSE of 0.873093 on the test set, the prediction time was about 45 minutues for the size of the test set. The training time was about 3 days on a quad-core PC (Intel i7). This is an notable improvement compared to the best RMSE achieved by a single neural network (RMSE:0.873334, Table 6.5). The second-placed team from the Netflix Prize competions published a paper of their best blending technique [37]. They get an improvement to linear regression of about 0.0020 RMSE on the Netflix test set. We get about 0.0021 RMSE improvement compared to linear regression.

Model	oob RMSE	weight	parameters
const. 1	-	-0.016	-
NN	0.873535	0.191	1HL:100, $of = 3.0$, $sc = 3.6$, $\eta = 0.0005$, $\eta^{(-)} = 10^{-6}$, 500 epochs
GBDT	0.874835	0.089	subspace $= 3$, maxLeafs $= 100$, $\eta = 0.2$, 100 epochs, optSplitPoint
PR	0.874344	0.122	order=1, $\lambda = 0.001$, with cross interactions
PR	0.893329	-0.061	order=3, $\lambda = 0.05$, no cross interactions
NN	0.873513	0.278	1HL:100, $of = 3.0$, $sc = 2$, $\eta = 0.0005$, $\eta^{(-)} = 10^{-6}$, 500 epochs
NN	0.873463	0.332	2HL:30-50, $of = 3.0$, $sc = 2$, $\eta = 0.0005$, $\eta^{(-)} = 10^{-6}$, 500 epochs
GBDT	0.878749	0.049	subspace $= 3$, maxLeafs $= 10000$, $\eta = 1.0$, 10 epochs, noOptSplitPoint
lin. blend	0.873349 test:0.873093		

Table 6.8: Blend stopping applied to blend CF predictions from the Netflix Prize dataset. The second column reports the out-of-bag (oob) estimate during bagging. The third column shows the values of the linear blending weights. Accurate blending methods receive high weights.

Chapter 7

Ensemble Learning Framework Software - ELF

We use our own ensemble learning software for all experiments and reported error values, except for the BGBDT - bagged gradient boosted decision trees results in Table 6.6, Table 6.7 and Figure 6.6.

The "ELF" is written in C++ and is a supervised learning framework. The class inheritance diagram is shows in Figure 7.1. The `Scheduler`-class is responsible for the coordination of the training and prediction process. The `StandardAlgorithm`-class implements the basic class of all supervised learners. The `Data` class hold the data set with train and test set and the cross validation data. The `AutomaticParameterTuner` implements the structured coordinate search. Linear combination of models is done with the `BlendStopping` class. In the `DatasetReader` there are the methods for reading a particular data set. `NumericalTools` hold diverse equation solvers which are used in the framework.

Design Goals

The ELF can handle regression as well as classification problems. For classification problems the ELF has build in support for multiple domains, for example the KDDCup2009 data set has 3 binary targets on the same features. The framework was designed for efficient learning with the help of numerical optimized libraries from Intel's MKL and IPP. Many base models get a sustainable speedup via these libraries. The internal data representation is based on the `REAL*`-datatype (float pointers). Even matrices or multidimensional arrays are stored as a large array, where accession is implemented by pointer arithmetics. The `REAL` data type is a `typedef` and stands for single or double precision floating point accuracy. The whole framework can easily change its internal floating point accuracy, which enables a speedup of about 2x and a reduction of memory consumption (when switching from `double` to `float`). All implemented models are able to save the model's internal parameters. This property enables us to set an ensemble in a ready-to-predict mode in order to predict the targets of any feature vector. We use the class `StreamOutput` to replace the STL `cout` object. This has the advantage that we can grab the console output to a dsc file, which can be used to automatically record each model's output. The class `Framework` holds some global properties like mode, randomSeed, maxThreads or startupParameters in static members.

Usage

The training setup of the ensemble are specified in the `Master.dsc` file under the specific data set directory. The following structure is the minimum requirement for start the training process on a specific

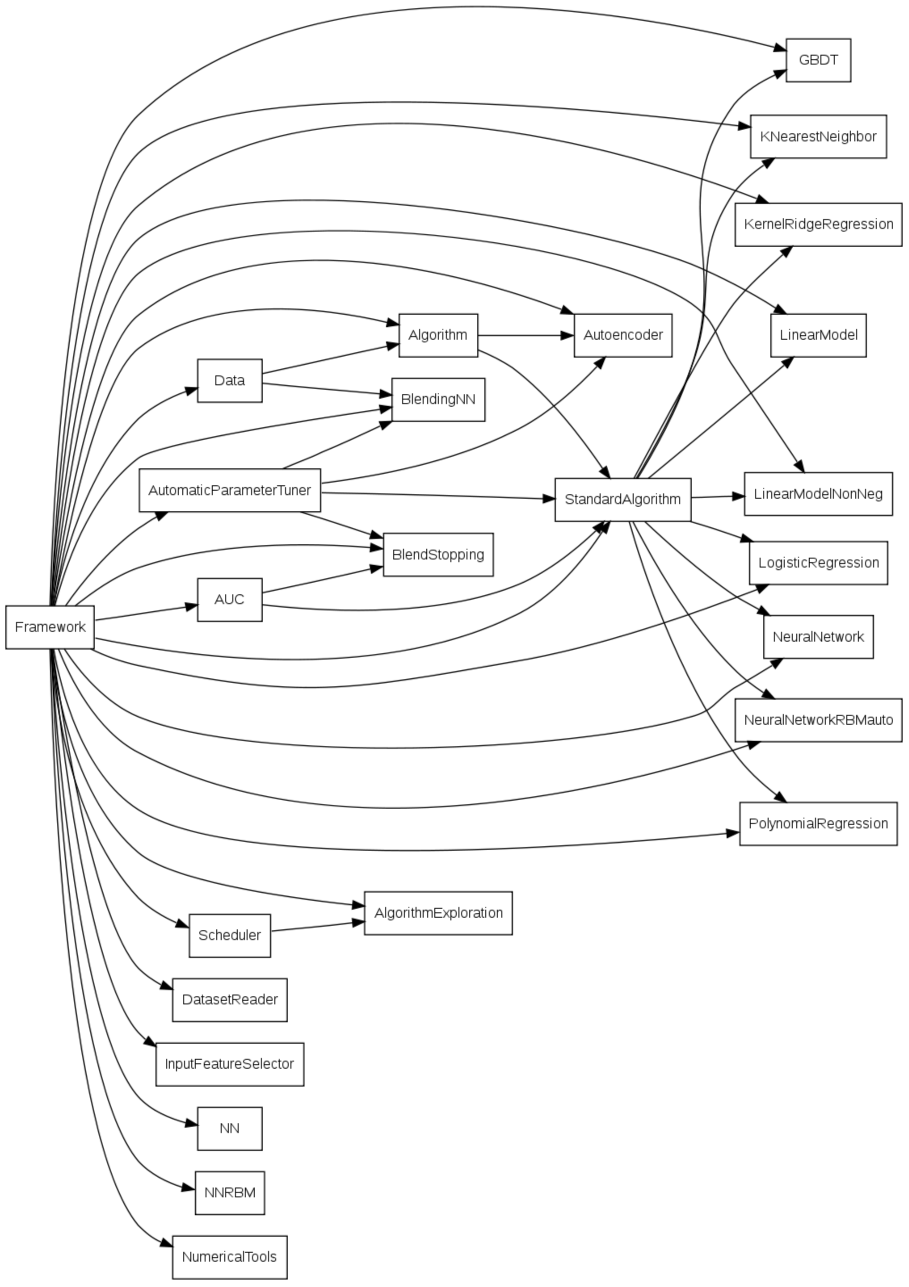


Figure 7.1: Class inheritance diagram of the ensemble learning framework (ELF).

data set. The file ending `.dsc` stands for description file, the format is plain text.
`DATASET/Master.dsc` : Master description file, global training options and a list of model dsc files
`DATASET/model-name0.dsc` : A model in the ensemble, e.g. `LinearModel-1.dsc`
`DATASET/model-name1.dsc` : A model in the ensemble, e.g. `NeuralNetwork-1.dsc`
`DATASET/DataFiles` : This directory contains the data set.
`DATASET/DscFiles` : The directory where the console output of the models are captured.
`DATASET/FullPredictorFiles` : The directory of the prediction of the train set.
`DATASET/TempFiles` : The directory, where combination weights and model weights are stored.

The data set name and [t,b,p] must be specified in the console in order to start the training or predic-

tion.
`$ ./ELF LETTER t` : start training
`$ ./ELF LETTER b` : start blending existing algorithms
`$ ./ELF LETTER p` : start prediction

The `Master.dsc` file

All global options and the order of the single models can be defined in the file. The first line must contain e.g. `dataset=LETTER`. The second line must contain `isClassificationDataset=[0,1]` to specify the type of training, 0 for regression, 1 for classification. The settings in Table 7.1 are optional. After that the section of model dsc files begin, the line with `[ALGORITHMS]` start this section. From now, each line contains the name of the specific dsc file of the single model description.

Implementation of the Proposed Ensemble Learning Techniques

Here we give a sketch of how we implement the ensemble learning techniques in the framework.

Linear combiner. This is a simple linear regression of all models in the ensemble, the set of models are defined in the `ALGORITHMS` section in the `Master.dsc` file in the data set folder. Each model minimized the linear combination by stop training or search for proper metaparameters. `blendingAlgorithm` must be set to LinearRegression. In each algorithm dsc file, the option `minimzeBlend=1` must be set.

Stacking. A linear regression with non-negative weights combines the algorithms without any interactions in the training. `blendingAlgorithm` must be set to LinearRegressionNonNeg. In each algorithm dsc file, the option `minimzeProbe=1` must be set.

Cascade Learning. The first model in the chain is trained in the standard way. The second one has an enlarged feature set by the cross-fold predictions of the train set of the first model and so on. `enableCascadeLearning` must be set to 1. `blendingAlgorithm` must be set to TakeLast in order to make final predictions only from the last model in the chain. In each algorithm dsc file, the option `minimzeBlend=1` must be set.

Training on Residuals. In each algorithm dsc file, there exist an option called `TRAIN_ON_FULLPREDICTOR`. When the value specifies a valid prediction file, the targets are subtracted by them. For example `TRAIN_ON_FULLPREDICTOR=NeuralNetwork_1.dat` would train on residuals from a neuronal network. `blendingAlgorithm` must be set to LinearRegression. In each algorithm dsc file, the option `minimzeBlend=1` must be set.

Training on Residuals + Cascade Learning. Options enabled by Training on Residuals and `enableCascadeLearning` must be set to 1.

Bagging. The experiments with bagging on the LETTER data set (Section 5.3) is done by calling the ELF with `./ELF LETTER x`. The bagging is implemented in the class `Scheduler` and `StandardAlgorithm`. The idea is to draw a bootstrap sample of the data set when the model gets retrained.

Boosting. The experiments with boosting on the LETTER data set (Section 5.3) is done by calling the ELF with `./ELF LETTER y`. The AdaBoost.M2 algorithm is implemented in the class `Scheduler`.

name	type	default value	description
`maxThreads`	`int`	2	Maximum number of threads used in MKL and IPP functions
`maxThreadsInCross`	`int`	2	Maximum number of parallel threads in the cross validation
`nCrossValidation`	`int`	6	The value of k in k-fold cross validation
`validationType`	`string`	Retraining	The type of validation. Possible values are Retraining, CrossFold-Mean and Bagging
`positiveTarget`	`float`	1.0	The value of positive class target
`negativeTarget`	`float`	-1.0	The value of negative class target
`randomSeed`	`int`	1234	Initial state of the random number generator, the string “time(0)” specifies a random init value
`nMixDataset`	`int`	20	Mix the data set by make n times the number of samples swaps
`nMixTrainList`	`int`	100	Mix the internal training index list by n times the number of samples swaps
`standardDeviationMin`	`float`	0.01	The minimum value for normalizing the input features
`blendingRegularization`	`float`	0.0001	The value of the ridge regression constant in the linear combiner
`blendingEnableCrossValidation`	`int`	0	This enables of disables the automatic search of the optimal ridge regression constant in the linear combiner
`blendingAlgorithm`	`string`	LinearRegression	The algorithm for the combination of the models. Possible values are LinearRegression, LinearRegressionNonNeg, TakeLast, NelderMead(for AUC optimization) and Average
`enablePostNNBlending`	`int`	0	When set to 1, it enables the a neural network (specified in the `NeuralNetwork.dscblend` file) to blend the existing predictions
`enableCascadeLearning`	`int`	0	When set to 1 the cascade learning mode is enabled
`enableGlobalMeanStdEstimate`	`int`	0	This enables uniform normalization over all input features by average mean and standard deviations
`enableSaveMemory`	`int`	1	This should be set to 1 in order to save memory when the size of `nCrossValidation` is large
`addOutputNoise`	`float`	0.0	Add prediction noise, this value specifies the standard deviation of a normal distributed noise
`enablePostBlendClipping`	`int`	0	This should be set to 0 when the dataset type is classification, it clips the predictions
`enableFeatureSelection`	`int`	0	The setting enables the option of a features selection process with a regularized linear model on double cross validation
`featureSelectionWriteBinaryDataset`	`int`	0	This works in combination with `enableFeatureSelection`, it makes a binary dataset from the feature selection process
`enableGlobalBlendingWeights`	`int`	1	Equal weights in the linear combination for all output variables
`errorFunction`	`string`	RMSE	Specified error function, possible values are RMSE, MAE or AUC
`disableWriteDscFile`	`int`	0	When set, no dsc files are written in the DscFiles directory
`enableStaticNormalization`	`int`	0	When set, the mean and standard deviations are taken from the next two options
`staticMeanNormalization`	`float`	0.0	Static mean value for normalizing the input features
`staticStdNormalization`	`float`	1.0	Static standard deviation value for normalizing the input features
`enableProbablisticNormalization`	`int`	0	Used in combination with `dimensionalityReduction`
`dimensionalityReduction`	`string`	no	Gives the ability of use an Autoencoder as dimensionality reduction. When Autoencoder is set, the file `Autoencoder.dsc` specify the neuronal net structure
`subsampleTrainSet`	`float`	1.0	Do a sub-sampling of the samples in the train set, the specified value corresponds to the percent
`subsampleFeatures`	`float`	1.0	Do a sub-sampling of the features in the train and test set, the specified value corresponds to the percent
`globalTrainingLoops`	`int`	1	Number of loops through the chain of models in order to refine the metaparameters

Table 7.1: All optional settings in the `Master.dsc` file.

Basic Regression Models

In this section we list all available regression models and list the implemented details. All learners are derived from the class `StandardAlgorithm`.

class LinearModel : This learner is based on linear regression with ridge regression. The linear regression is calculated for each target. No constant feature is added here, the constant must be included in the data set. Ridge regression constant can be automatically optimized. An example dsc file is shown in Table 7.2.

class NeuralNetwork : Multilayer neural network implementation trained with the backprop algorithm. Table 7.2 shows various options concerning the net structure and learn parameters. The number of training epochs are optimized with the parameter tuner.

class KNearestNeighbor : Online k-nearest neighbor algorithm. It is online, because for each prediction the set of k-nearest neighbors is calculated again. In other words we do not precalculate all correlations between all samples. The distance measure can be Pearson correlation or inverse euclidean distance. Tunable parameters is the neighborhood size k and some distance measure specific constants. Table 7.2 shows an example dsc file.

class KernelRidgeRegression : Implements the kernel ridge regression learn algorithm. Gauss, polynomial or tanh kernel can be selected. Tunable parameters are the kernel constants and the ridge regression constant. The computational expensive step is calculating the inverse of the Gram matrix, this is done with Cholesky factorization. Based our knowledge, this is the fastest way to solve a linear system of equations. This let us work with data sets up to size of 60k samples.

class PolynomialRegression : This is linear regression with an enlarged feature set, therefore very similar to the `LinearModel`.

class GBDT : An implementation of the gradient boosted decision tree algorithm. A single tree is build in a greedy fashion by split the largest node recursive, the stop criteria is the maximum number of leafs. The user can specify weather the split should be found optimal or a random split. A subspace size ca be set. The number of gradient boosting epochs are optimized automatically.

class LinearModelNonNeg : This is an experimental algorithm, the idea is to restrict a linear regression model with non-negative weights.

class LogisticRegression : This is an experimental implementation of logistic regression.

class NeuralNetworkRBMauto : The idea here is to train an autoencoder network in the first step, then train the neural network with the trained weights from the autoencoder. It is an experimental implementation.

Extensions and Ongoing Development

The ELF is a supervised learning tool and is applied to many data sets from the UCI database [3] as well as in the Netflix Prize competition in order to blend predictions. The framework implements a few ensemble learning techniques paired with solid base learners. It can be used to train from only a single model up to a long chain of coupled models. It was also applied by us on three other competitions, the KDDCup2009 [1], the Prudsys Data Mining Cup [2] and on the AusDM2009 Analytic Challenge [3]. There is always room for improvement of the framework's architecture or clean bugs in the code. We plan to implement new base learners and integrate bagging as model validation process. The parallelization of the validation process over a computer cluster is also planned. Furthermore the usability of some classes and the documentation are currently under work.

[1]`http://www.kddcup-orange.com/`
[2]`http://www.data-mining-cup.com/2009/Wettbewerb/`
[3]`http://www.tiberius.biz/ausdm09/index.html`

LinearModel_1.dsc

```
ALGORITHM=LinearModel
ID=1
TRAIN_ON_FULLPREDICTOR=NeuralNetwork_1.dat
DISABLE=0

[int]
maxTuninigEpochs=10

[double]
initMaxSwing=0.1
initReg=0.1

[bool]
tuneRigeModifiers=0
enableClipping=0
enableTuneSwing=0

# minimize probe/blend RMSE
minimzeProbe=0
minimzeProbeClassificationError=0
minimzeBlend=1
minimzeBlendClassificationError=0

[string]
weightFile=LinearModel_1_weights.dat
fullPrediction=LinearModel_1.dat
```

NeuralNetwork_1.dsc

```
ALGORITHM=NeuralNetwork
ID=1
#TRAIN_ON_FULLPREDICTOR=GBDT_1.dat
DISABLE=0

[int]
nrLayer=2
batchSize=1
minTuninigEpochs=10
maxTuninigEpochs=530

[double]
initMaxSwing=1.0

# output = outputNN * scale + offset
offsetOutputs=3.0
scaleOutputs=3.8

etaPosRPROP=1.05
etaNegRPROP=0.9
minUpdateRPROP=1e-8
maxUpdateRPROP=1e-2

initWeightFactor=1.0
learnrate=1.0e-3
learnrateMinimum=1e-6
learnrateSubtractionValueAfterEverySample=0.0e-12
learnrateSubtractionValueAfterEveryEpoch=2e-6
momentum=0.0
weightDecay=0.0e-2
minUpdateErrorBound=3e-7

[bool]
enableL1Regularization=0
enableClipping=0
enableTuneSwing=0
useBLASforTraining=1
enableRPROP=0

# minimize probe/blend RMSE
minimzeProbe=0
minimzeProbeClassificationError=0
minimzeBlend=1
minimzeBlendClassificationError=0

[string]
neuronsPerLayer=50,50
weightFile=NeuralNetwork_1_weights.dat
fullPrediction=NeuralNetwork_1.dat
```

KNearestNeighbors_1.dsc

```
ALGORITHM=KNearestNeighbor
ID=1
#TRAIN_ON_FULLPREDICTOR=LinearModel_1.dat
DISABLE=0

[int]
kInit=4
maxTuninigEpochs=1
evaluationBlockSize=2000

[double]
initMaxSwing=1.0
initScale=5.0
initScale2=1.0
initOffset=2.0
initGlobalOffset=-0.05
initPower=1.0
euclideanPower=1.0
euclideanLimit=0.01

[bool]
takeEuclideanDistance=1
#denormalize trainset with mean and std
dataDenormalization=0
optimizeEuclideanPower=1
enableAdvancedKNN=0
enableClipping=0
enableTuneSwing=0

# minimize probe/blend RMSE
minimzeProbe=0
minimzeProbeClassificationError=0
minimzeBlend=1
minimzeBlendClassificationError=0

[string]
weightFile=KNearestNeighbor_1_weights.dat
fullPrediction=KNearestNeighbor_1.dat
```

Table 7.2: Example dsc files. A linear regression, neural network and k-nearest neighbors. The #-sign starts a comment line.

How to Build the Framework under Linux

We have tested the following instructions under Ubuntu Linux. This setup the Intel compiler and performance libraries for non commercial usage.

1. Download the Intel C++ compiler from `http://software.intel.com/en-us/articles/non-commercial-software-download/` and select the `Intel C++ Compiler Professional Edition for Linux`. Extract the 700MB tgz and follow the install instructions. This installs the compiler, the MKL and IPP to `/opt/intel/Compiler/...` on the system.

2. Edit the `/etc/ld.so.conf` file by adding the following lines. Replace the XX with the current version code of the Intel toolchain. After that run `ldconfig`.
   ```
   /opt/intel/Compiler/XX.X/XXX/lib/intel64
   /opt/intel/Compiler/XX.X/XXX/mkl/lib/em64t
   /opt/intel/Compiler/XX.X/XXX/ipp/em64t/sharedlib
   ```

3. Modify path to IPP and MKL headers and libraries in the Makefile.

4. Go to the directory of the ELF and do `make`. Done.

Chapter 8

Conclusion and Outlook

Ensemble learning techniques or simply "Ensembling" works well in many situations. The motivation of ensemble learning is that the accuracy of a set of models is superior compared to the best single model. This set can be assembled in many different ways (see Section "Introduction" 1.2 for a summary). The success of the ensemble depends on the diversity and the accuracy of each of the individuals, but this is a contradiction. Accurate models are always similar, they all model the same latent function. Different models have different prediction behaviour, for example a neural network's prediction should be different to the one obtained from a k-nearest neighbor algorithm (KNN is memory based and neural networks are not).

One of the simplest techniques for the construction of an ensemble is bagging from Breiman [6]. In bagging the type of model is fixed, each individual model is learned multiple times with a slightly modified version of the training set ("bootstrap sampling" of the data set, Section 4.2). Breiman shows significant improvement of the generalization error, when the base learner has unstable behavior. Decision trees are good candidates for bagging. Our results with bagging show a remarkable improvement (Section 5.3 and Section 6.8).

Another appoach for building the ensemble set of predictors is to run different kind of models. For example linear regression, neuronal networks, k-nearest neighbors and decision trees. Each of the four learners has different training mechanisms and therefore the predictions are not highly correlated. Less correlation and high accuracy are needed to build a good ensemble. The goal now is to combine them. We investigate four different techniques in order to obtain an accurate ensemble model. The four techniques are stacking, cascade learning, training on residuals and cascade learning with training on residuals. Stacking and cascade learning are well known in the literature, training on residuals has a relationship to boosting, because each of the models are learned on the residuals of the model before.

The outcome of the analysis on 14 small UCI data sets shows that **cascade learning** is the most successful technique to combine predictors. We investigate an ensemble of maximum 4 different kinds of models (linear regression, neural networks, kernel ridge regression and k-nearest neighbors). All possible configurations are explored when building the ensemble, because 3 of the 4 techniques are affected by the ordering of the individual models (only stacking is a loose coupling). When we compare the best single model versus the best ensemble (Table 5.6) we see that the ensemble never looses and improves performance significantly for 6 times out of 14 data sets. The winner, Cascade learning by Gama and Bradzil [17] is a technique which enlarges the feature space with the predictions of models in lower layers. This means that for every training example the current trained model has additional access to the predictions generated by all other models. This enables the current model to learn special structure out of the data. In the data set BALANCE [35] the approach of cascade learning significantly improves the accuracy on the test set.

Boosting was successfully applied on the LETTER data set, the results are shown in Figure 5.5. We use the Adaboost.M2 algorithm from Schwenk and Bengio [34], Adaboost.M2 does not a reweighting of the training examples (like Adaboost), instead it does resampling the training list based on the train error. Examples with high train error get a higher probability to be in the training set of the next round. We plot the accuracy over the boosting epochs and compare it to bagging. Adaboost.M2 outperforms bagging significantly. The obtained accuracy of 1.5% are well comparable to those in [34].

We apply stacking of 4 different models on the MNIST data set. The results are shown in Table 5.9. Stacking is designed to minimize the mean square prediction error by optimizing a non-negative combination of predictions. This declines the combined performance in terms of classification error. The best single model is kernel ridge regression with 1.23% classification error and an RMSE of 0.14336. The combination of all 4 models lowers the RMSE to 0.09948 but result in an increase of the classification error to 1.36%. This is a demonstrative example that minimizing the RMSE does not necessarily minimize the classification error.

Over all investigated single models **kernel ridge regression** is one of the strongest supervised learning algorithms. It has the drawback of bad scalability, training time rises with the third power of the number of training examples. Neural networks have also good performance in various experiments. Linear regression is used many times as a baseline model due to its simplicity.

The accuracy in recommender systems greatly improves with the help of ensembling techniques. The Netflix Prize data set is used here to generate the predictions. We used 18 different predictors obtained from k-nearest neighbors, SVD, asymmetric factor models, restricted boltzmann machines and global effect models. All of them are collaborative filtering models and trained on 100M ratings. Some of them are trained on the residuals of others. The complete list is under Table 6.2. This 18 predictors are now the ensemble of models, which needs to be combined. The evaluation of different blending algorithms are done on the probe set, which is a 1.4M subset of the whole Netflix data set. Furthermore we make a 50:50 split of the probe in a train and a test set. Linear regression with ridge regression acts as baseline and result in RMSE=0.8752 on the test set. We found that splitting the data based on the support criteria delivers best results. Linear regression with 5 bins gives RMSE=0.874741. A neural network with 500 neurons on the hidden layer achieve a RMSE of 0.873848 on the test set, when we add the $\log(support)$ as additional input the RMSE drops the 0.873334 (results from Table 6.5). This is the best result obtained from **neural networks** and the best result from all blending techniques investigated here. Tree blending with a GBDT achieves RMSE=0.873945. Additionally we tried kernel ridge regression and k-nearest neighbors on a random subset of the train set and report the performance when many models are averaged. Kernel ridge regression delivers good results, the RMSE drops to 0.8740, k-nearest neighbors are totally unsuitable for blending, the best achieved RMSE value was 0.8840. In section 6.8 we investigate the ability of bagging applied to 7 different blending models. Linear regression in combination with blend stopping was used to combine those 7 models. At the end we can show with a lot of computing power the RMSE on the test set can be lowered to 0.873093.

Almost all experiments were done with a own-developed framework for supervised ensemble learning. The details are sketched in Section 7. The main design goal was a fast implementation of the base learners with the help of optimized Intel libraries for numerical computations (MKL, IPP). This C++ based framework was not only developed for running experiments for this thesis, it can easily deployed to real world problems.

Possible Improvements

The ensemble learning techniques cascade learning and training on residuals learn models in a chain by a greedy reduction of the ensemble's RMSE. Parameters found in the first model need not to be optimal when the ensemble chain has finished training. A possible extention is to repeat training of the model chain by keeping all other model predicitons constant and re-optimize the current one. This can be understood as a refinement of the metaparameters when the ensemble is already build.

In stacking and training on residuals we use a linear combination as the combiner (non-negative weights for stacking). The linear combination can lead to overfitting on small data sets or on data sets with many predicted outputs (e.g. multi-class problems). A possible improvement is to add cross validation in order to optimize the ridge regression constant used for doing the linear regression (LR as combiner).

In the ensemble learning technique "ensemble selection" from Caruana et al. [11], one uses cross validation for parameter and model selection. When a prediction of a test sample is done one uses the mean of the k models in the cross validation set. In contrast to this approach, we use retraining of the model with best metaparameters. Averaging the predictions from all k models leads to a bagging-like effect, which helps to improve the accuracy. This approach can also be used during our evaluation. Furthermore, bagging can replace the cross validation process by optimizing on the out-of-bag estimate.

Appendix A

Additional Tables and Figures from the analysis of ensemble learning techniques

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**13.65**±3.0	**13.65**±3.0	13.92±3.0	13.92±3.0	N	14.34±2.7	14.34±2.7	**14.30**±2.7	**14.30**±2.7
K	**17.39**±3.2	**17.39**±3.2	17.51±3.1	17.51±3.1	R	14.81±2.8	14.81±2.8	**14.78**±2.8	**14.78**±2.8
L-N	14.56±2.8	14.69±2.8	**14.48**±2.9	14.74±2.7	L-K	13.85±2.6	**13.81**±2.6	14.45±2.9	
L-R	14.29±2.8	**14.06**±2.8	14.24±3.1		N-L	**14.17**±2.6	14.21±2.6	14.41±3.2	
N-K	**14.98**±2.8	15.03±2.7	15.00±2.8		N-R	14.55±2.6	14.58±2.7	**14.40**±2.6	
K-L	14.72±2.9	**13.87**±3.9	14.40±4.3	14.11±2.7	K-N	14.67±2.8	13.70±2.8	**13.59**±2.7	14.05±2.5
K-R	15.24±3.3	14.05±3.3	**13.50**±3.0	14.28±2.6	R-L	**13.97**±2.7	14.52±2.9	14.43±4.0	14.28±2.9
R-N	14.03±2.6	14.27±2.6	**13.81**±2.6	14.87±2.8	R-K	15.63±2.8	15.20±3.0	**15.13**±2.9	
L-N-K	13.95±2.6	13.92±2.5	**13.74**±2.7		L-N-R	14.90±5.8	15.08±7.4	**14.36**±3.1	
L-K-N	14.49±2.7	14.16±2.5	**13.80**±2.7		L-K-R	15.22±4.3	15.13±5.1	**14.68**±2.9	
L-R-N	14.45±3.9	14.72±4.6	**13.98**±2.8		L-R-K	13.76±2.8	13.67±2.8	**13.49**±2.6	
N-L-K	14.49±2.9	14.60±2.9	**13.62**±2.8		N-L-R	15.09±2.7	15.32±2.5	**14.92**±2.7	
N-K-L	14.76±2.9	14.57±2.8	**14.00**±2.8		N-K-R	14.83±2.9	14.87±2.9	**14.21**±2.5	
N-R-L	**14.85**±2.7	14.98±2.7	15.08±3.1		N-R-K	15.02±2.8	14.88±2.8	**14.34**±2.4	
K-L-N	15.43±2.9	14.73±3.1	**13.60**±2.6	14.26±2.7	K-L-R	15.65±2.7	**14.02**±2.5	14.25±2.3	
K-N-L	15.66±2.9	**13.86**±2.7	13.98±3.8		K-N-R	15.45±3.0	14.26±2.6	**13.78**±2.8	
K-R-L	15.46±3.2	14.19±3.1	14.77±4.7	**13.88**±3.1	K-R-N	14.65±2.9	13.90±2.7	**13.43**±2.9	13.93±2.6
R-L-N	14.62±2.8	14.76±2.8	**13.84**±2.4	14.25±3.8	R-L-K	14.90±2.4	15.17±2.4	**13.70**±2.9	
R-N-L	14.61±3.0	14.73±2.7	**14.44**±3.8		R-N-K	14.76±3.0	14.76±2.7	**13.52**±2.5	
R-K-L	15.18±2.4	15.08±2.6	**14.47**±3.1		R-K-N	14.73±2.5	15.00±2.7	**13.80**±2.5	
L-N-K-R	14.74±4.0	14.96±4.8	**14.51**±2.9		L-N-R-K	14.49±4.7	14.60±6.1	**13.86**±2.2	
L-K-N-R	15.35±6.2	15.55±8.9	**14.14**±2.7		L-K-R-N	14.05±2.8	13.87±2.5	**13.52**±2.1	
L-R-N-K	14.14±2.4	14.06±2.5	**13.73**±3.0		L-R-K-N	14.06±2.7	13.94±2.8	**13.78**±2.9	
N-L-K-R	15.03±2.8	14.83±2.7	**14.36**±2.7		N-L-R-K	14.20±3.0	14.50±2.9	**13.59**±2.5	
N-K-L-R	14.75±2.9	**13.84**±3.0	13.98±3.1		N-K-R-L	15.18±3.1	14.99±3.2	**14.34**±3.0	
N-R-L-K	15.05±2.7	14.89±2.7	**14.03**±2.9		N-R-K-L	15.02±2.6	14.97±2.9	**14.61**±3.7	
K-L-N-R	16.04±2.8	14.62±2.6	**14.10**±2.4		K-L-R-N	15.57±3.0	14.41±3.0	**13.51**±2.7	
K-N-L-R	15.98±2.8	14.14±2.9	**14.11**±2.6		K-N-R-L	15.89±3.0	14.39±2.4	**13.92**±2.4	
K-R-L-N	15.31±2.9	14.54±2.8	13.85±2.6	**13.78**±2.3	K-R-N-L	16.06±3.0	14.72±2.6	**14.64**±2.6	
R-L-N-K	15.12±2.7	15.37±3.0	**13.94**±2.7		R-L-K-N	15.50±2.5	15.56±2.7	**14.49**±2.6	
R-N-L-K	14.94±2.6	14.50±2.9	**13.61**±2.6		R-N-K-L	15.25±3.6	15.14±3.5	**14.13**±3.1	
R-K-L-N	15.57±3.2	15.74±2.9	**14.24**±3.1		R-K-N-L	14.69±2.5	14.43±2.6	**14.02**±2.8	

Table A.1: Results for the CREDIT dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

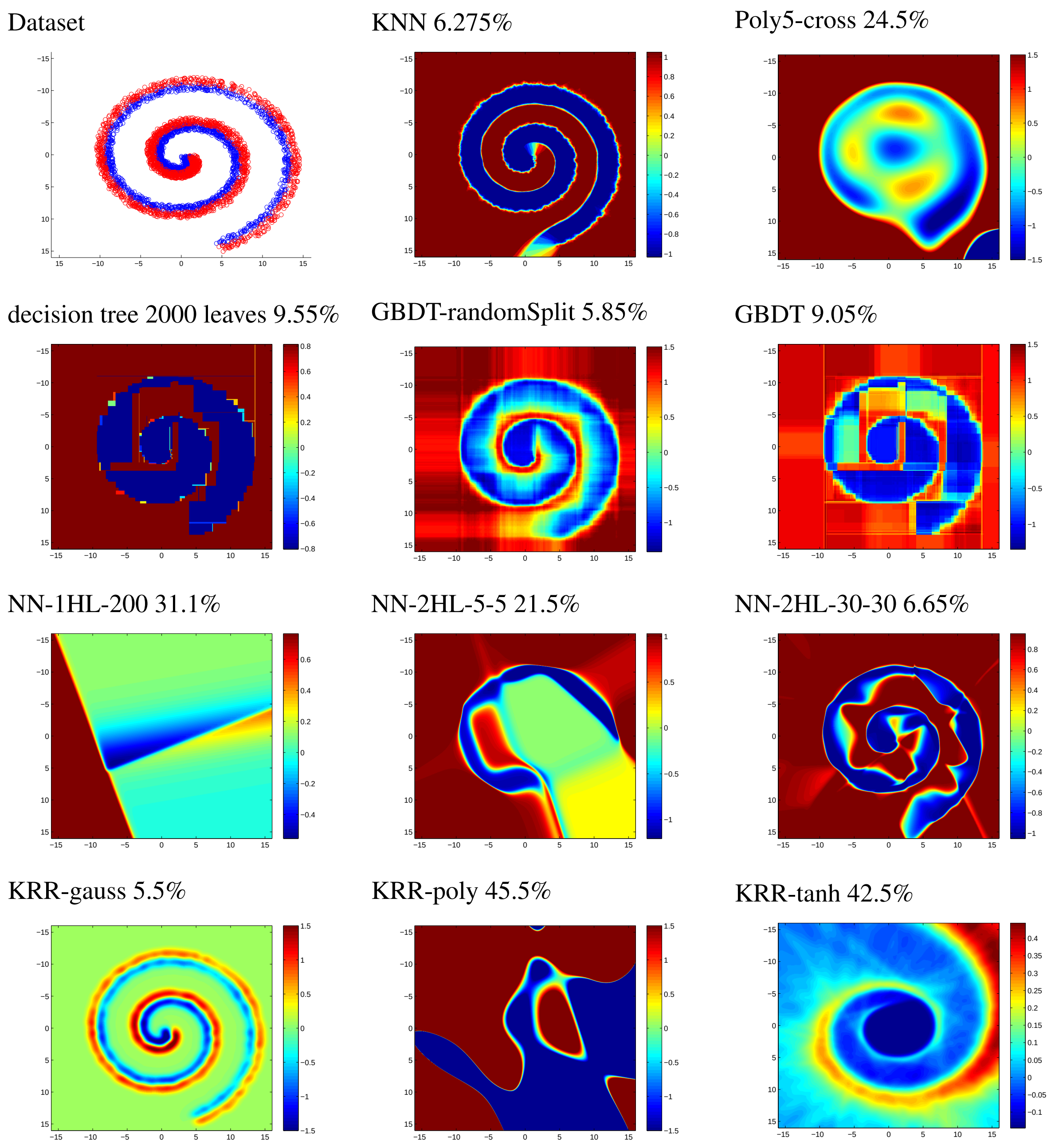


Figure A.1: Spiral dataset, comparision of the model prediction in feature space. We optimized each particulars model parameters in order to lower the error. The reported value (right beside the model name) is the classification error on a 4-fold cross validation set. KNN=k-nearest neighbors, PolyX=polynomial regression, GBDT=gradient boosted decision trees, NN=neural networks, KRR=kernel ridge regression. The trainig set has a size of 4000 samples. Generated with: `d=gen(spiral('m=2000','n=2.2','noise=1.0'));`

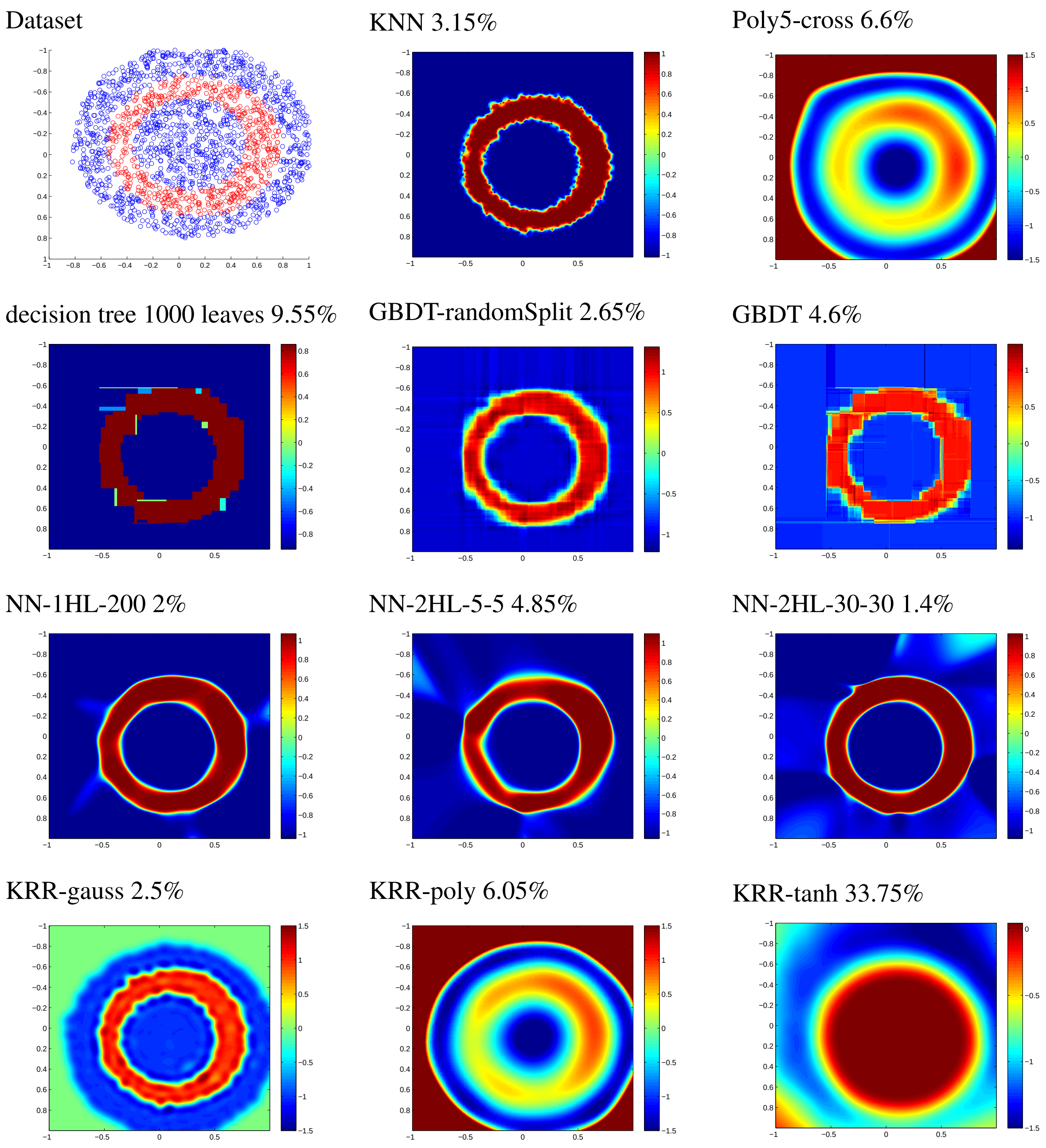


Figure A.2: Circle dataset, comparision of the model prediction in feature space. We optimized each particular's model parameters in order to lower the error. The reported value (right beside the model name) is the classification error on a 4-fold cross validation set. KNN=k-nearest neighbors, PolyX=polynomial regression, GBDT=gradient boosted decision trees, NN=neural networks, KRR=kernel ridge regression. The trainig set has a size of 4000 samples. Generated with: `d=gen(toy2d('2circles','l=2000'));`

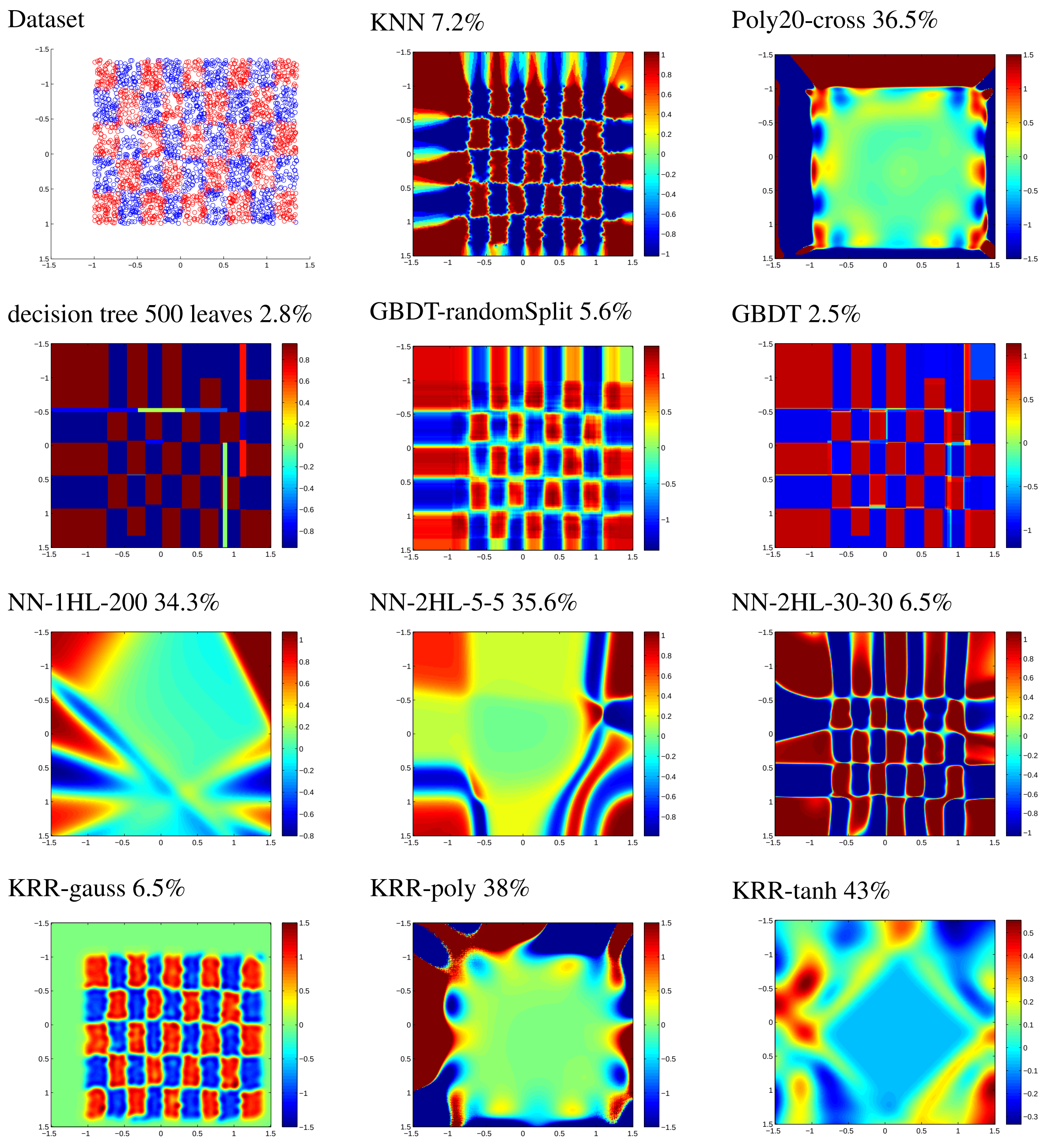


Figure A.3: Chess dataset, comparision of the model prediction in feature space. We optimized each particular's model parameters in order to lower the error. The reported value (right beside the model name) is the classification error on a 4-fold cross validation set. KNN=k-nearest neighbors, PolyX=polynomial regression, GBDT=gradient boosted decision trees, NN=neural networks, KRR=kernel ridge regression. The trainig set has a size of 4000 samples. Generated with: `d=gen(toy2d('chess','l=3000'));`

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**12.81**±2.6	**12.81**±2.6	**12.81**±2.6	**12.81**±2.6	N	**8.96**±2.1	**8.96**±2.1	**8.96**±2.1	**8.96**±2.1
K	10.00±2.5	10.00±2.5	**9.96**±2.5	**9.96**±2.5	R	**9.46**±2.1	**9.46**±2.1	9.51±2.1	9.51±2.1
L-N	9.92±2.1	10.87±2.6	9.76±2.5	**8.79**±2.1	L-K	10.01±2.7	**9.49**±2.8	10.00±2.8	
L-R	9.29±2.5	**8.53**±2.3	8.89±2.5		N-L	9.24±2.5	**4.68**±2.6	4.79±2.6	
N-K	8.95±2.3	**4.91**±2.3	5.74±2.3		N-R	8.75±2.5	**4.92**±2.5	5.11±2.5	
K-L	**10.05**±2.2	10.20±2.4	11.32±8.4	12.93±3.2	K-N	10.21±2.3	10.05±2.5	9.47±2.4	**9.25**±2.5
K-R	10.32±2.7	**8.94**±2.7	9.06±2.7	9.16±2.5	R-L	9.55±2.3	**9.36**±2.3	9.36±2.3	13.13±2.5
R-N	9.66±2.6	8.61±2.9	**7.33**±3.2	9.05±2.2	R-K	8.63±2.3	**8.28**±2.4	9.16±2.4	
L-N-K	9.46±2.3	8.00±2.5	**7.91**±2.5		L-N-R	10.23±2.8	7.54±2.8	**6.63**±2.4	
L-K-N	9.70±2.3	**9.20**±2.2	9.28±2.3		L-K-R	9.56±2.3	**8.67**±2.2	8.88±2.2	
L-R-N	9.53±2.3	8.67±2.6	**7.19**±3.0		L-R-K	8.20±2.5	7.42±2.1	**7.25**±2.5	
N-L-K	8.61±2.2	**4.40**±2.2	4.55±2.3		N-L-R	9.21±2.3	4.80±2.4	**4.52**±2.1	
N-K-L	8.82±2.4	**4.77**±2.4	4.94±2.1		N-K-R	8.17±2.2	**5.04**±2.3	5.14±2.5	
N-R-L	9.10±2.6	**4.63**±2.3	4.66±2.1		N-R-K	8.61±2.3	5.69±2.4	**5.01**±2.3	
K-L-N	9.76±2.6	9.48±5.4	9.00±2.9	**8.88**±1.9	K-L-R	10.54±2.5	9.45±4.1	**9.14**±2.5	
K-N-L	10.22±2.7	10.19±2.7	**8.90**±3.0		K-N-R	10.64±2.2	9.61±2.4	**8.68**±2.4	
K-R-L	10.66±2.4	9.59±2.4	**9.26**±2.5	12.71±2.8	K-R-N	9.95±2.6	9.00±2.6	**8.61**±2.5	9.15±2.3
R-L-N	9.18±2.4	**5.10**±2.2	5.41±2.5	9.00±2.2	R-L-K	8.74±2.7	**6.77**±2.6	7.22±2.7	
R-N-L	9.40±2.3	8.53±2.6	**5.71**±2.4		R-N-K	8.62±2.1	6.47±2.2	**6.25**±2.3	
R-K-L	8.61±2.5	**8.15**±2.4	8.54±2.8		R-K-N	8.22±2.4	7.84±2.3	**7.15**±2.4	
L-N-K-R	8.55±2.5	8.41±2.4	**7.37**±2.3		L-N-R-K	9.33±2.3	8.06±2.5	**7.43**±2.6	
L-K-N-R	9.52±2.5	8.42±2.5	**7.79**±2.4		L-K-R-N	9.63±2.4	**8.16**±2.3	8.64±2.5	
L-R-N-K	8.42±2.4	7.30±2.3	**6.65**±2.3		L-R-K-N	8.19±2.5	7.43±2.4	**6.58**±2.2	
N-L-K-R	8.28±2.2	5.01±2.2	**4.87**±2.1		N-L-R-K	8.79±2.6	4.58±2.4	**4.31**±2.2	
N-K-L-R	8.60±2.8	**5.15**±2.2	5.37±2.8		N-K-R-L	8.74±2.5	**4.82**±2.2	5.44±4.0	
N-R-L-K	9.30±2.4	5.88±2.3	**5.00**±2.0		N-R-K-L	8.48±2.3	**4.68**±2.1	5.27±7.3	
K-L-N-R	10.06±2.6	8.91±3.1	**8.07**±2.9		K-L-R-N	10.21±2.4	8.99±2.7	**8.59**±2.6	
K-N-L-R	10.65±2.5	9.46±2.5	**8.75**±2.5		K-N-R-L	10.32±2.4	9.00±2.3	**8.15**±2.1	
K-R-L-N	10.43±2.5	8.89±2.6	**8.67**±2.7	12.63±3.1	K-R-N-L	10.21±2.4	8.90±2.2	**7.13**±2.3	
R-L-N-K	8.61±2.5	5.55±2.2	**5.29**±2.3		R-L-K-N	8.11±2.6	6.44±2.2	**6.02**±2.3	
R-N-L-K	8.84±2.6	6.89±2.4	**5.32**±2.1		R-N-K-L	8.25±2.6	**6.38**±2.4	7.18±8.7	
R-K-L-N	7.88±2.4	7.87±2.4	**7.32**±2.2		R-K-N-L	8.38±2.0	7.93±2.3	**5.78**±2.6	

Table A.2: Results for the BALANCE dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**3.92**±1.7	**3.92**±1.7	4.09±1.8	4.09±1.8	N	**3.51**±1.5	**3.51**±1.5	**3.51**±1.5	**3.51**±1.5
K	**3.36**±1.4	**3.36**±1.4	3.37±1.4	3.37±1.4	R	**3.24**±1.3	**3.24**±1.3	3.24±1.3	3.24±1.3
L-N	3.71±1.6	3.66±1.5	**3.27**±1.4	3.29±1.5	L-K	3.15±1.5	3.05±1.6	**3.00**±1.5	
L-R	3.60±1.4	3.49±1.4	**3.24**±1.5		N-L	3.27±1.3	3.29±1.3	**3.25**±1.2	
N-K	3.62±1.4	3.66±1.4	**3.59**±1.5		N-R	3.47±1.3	3.44±1.3	**3.34**±1.3	
K-L	3.46±1.7	3.44±1.7	**3.39**±1.7	3.84±1.5	K-N	3.28±1.2	3.29±1.2	**3.24**±1.3	3.29±1.3
K-R	3.29±1.3	3.32±1.3	**3.26**±1.3	3.30±1.3	R-L	3.47±1.4	3.47±1.4	**3.33**±1.7	4.20±1.6
R-N	3.44±1.4	3.46±1.3	**3.16**±1.2	3.57±1.3	R-K	3.32±1.4	**3.27**±1.4	3.31±1.4	
L-N-K	3.62±1.4	**3.37**±1.4	3.41±1.4		L-N-R	3.49±1.3	3.43±1.2	**3.34**±1.3	
L-K-N	3.34±1.3	**3.30**±1.2	3.33±1.3		L-K-R	3.33±1.4	3.27±1.3	**3.16**±1.3	
L-R-N	3.36±1.5	3.33±1.5	**2.96**±1.4		L-R-K	3.51±1.3	3.64±1.4	**3.22**±1.3	
N-L-K	3.70±1.4	3.63±1.3	**3.63**±1.4		N-L-R	**3.11**±1.3	3.24±1.3	3.11±1.2	
N-K-L	3.32±1.3	3.36±1.3	**3.21**±1.3		N-K-R	3.54±1.3	3.59±1.4	**3.35**±1.3	
N-R-L	3.52±1.3	3.51±1.3	**3.42**±1.4		N-R-K	3.60±1.3	3.59±1.4	**3.53**±1.4	
K-L-N	3.39±1.6	3.35±1.6	**3.22**±1.5	4.23±1.7	K-L-R	**3.40**±1.4	3.42±1.4	3.40±1.4	
K-N-L	3.62±1.4	**3.56**±1.4	3.58±1.5		K-N-R	3.53±1.4	3.55±1.4	**3.51**±1.3	
K-R-L	3.49±1.3	3.45±1.4	3.44±1.3	**3.11**±1.3	K-R-N	3.60±1.4	3.61±1.4	3.49±1.4	**3.19**±1.3
R-L-N	3.52±1.4	3.55±1.5	**3.12**±1.4	4.07±1.3	R-L-K	3.68±1.4	3.57±1.2	**3.19**±1.3	
R-N-L	3.60±1.3	3.62±1.3	**3.30**±1.4		R-N-K	3.61±1.3	3.52±1.3	**3.22**±1.2	
R-K-L	3.62±1.4	3.54±1.3	**3.23**±1.3		R-K-N	3.84±1.3	3.71±1.3	**3.27**±1.3	
L-N-K-R	3.61±1.5	3.47±1.4	**3.34**±1.4		L-N-R-K	3.47±1.5	3.33±1.4	**3.10**±1.3	
L-K-N-R	3.65±1.4	3.58±1.6	**3.39**±1.5		L-K-R-N	3.65±1.4	3.40±1.4	**3.26**±1.4	
L-R-N-K	3.45±1.6	3.46±1.4	**3.20**±1.4		L-R-K-N	3.72±1.5	3.67±1.4	**3.16**±1.4	
N-L-K-R	3.70±1.3	3.72±1.5	**3.41**±1.4		N-L-R-K	3.37±1.3	3.45±1.4	**3.26**±1.3	
N-K-L-R	3.35±1.3	3.37±1.3	**3.17**±1.3		N-K-R-L	3.44±1.4	**3.32**±1.4	3.73±4.0	
N-R-L-K	3.46±1.4	3.45±1.4	**3.27**±1.2		N-R-K-L	3.72±1.5	3.77±1.5	**3.70**±1.5	
K-L-N-R	3.37±1.4	3.40±1.4	**3.28**±1.4		K-L-R-N	3.30±1.3	3.34±1.3	**3.16**±1.2	
K-N-L-R	3.71±1.4	3.73±1.5	**3.67**±1.5		K-N-R-L	**3.54**±1.4	4.50±9.1	4.16±4.6	
K-R-L-N	3.60±1.5	3.59±1.4	3.55±1.3	**3.25**±1.3	K-R-N-L	3.30±1.2	3.26±1.2	**3.18**±1.3	
R-L-N-K	3.87±1.4	3.90±1.5	**3.48**±1.4		R-L-K-N	3.81±1.4	3.81±1.4	**3.06**±1.3	
R-N-L-K	3.57±1.4	3.44±1.4	**3.13**±1.3		R-N-K-L	3.88±1.6	3.95±1.7	**3.36**±1.5	
R-K-L-N	3.63±1.5	3.73±1.5	**3.24**±1.4		R-K-N-L	3.64±1.4	3.63±1.4	**3.18**±1.4	

Table A.3: Results for the BREAST dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**23.10**±3.2	**23.10**±3.2	**23.10**±3.2	**23.10**±3.2	N	**23.43**±3.1	**23.43**±3.1	23.50±3.1	23.50±3.1
K	25.39±3.0	25.39±3.0	**25.31**±3.1	**25.31**±3.1	R	23.12±3.2	23.12±3.2	**23.12**±3.1	**23.12**±3.1
L-N	23.81±3.6	23.83±3.5	23.48±3.4	**23.44**±3.4	L-K	23.06±3.3	**22.95**±3.3	23.33±3.1	
L-R	23.69±3.2	23.57±3.2	**23.23**±2.9		N-L	**22.62**±3.4	22.74±3.5	22.79±3.4	
N-K	23.20±3.1	**23.09**±2.9	23.50±3.4		N-R	23.45±3.0	23.46±2.8	**22.92**±3.0	
K-L	24.49±3.5	23.51±3.3	23.39±3.3	**22.55**±3.1	K-N	24.21±3.8	23.04±3.3	23.39±3.3	**22.58**±3.4
K-R	24.36±3.4	23.30±3.3	23.50±3.3	**22.65**±3.0	R-L	23.17±3.2	23.25±3.2	**22.87**±2.9	23.11±3.3
R-N	**22.78**±2.9	22.93±3.1	22.92±2.7	23.19±3.0	R-K	23.14±2.9	**22.85**±2.5	23.46±2.7	
L-N-K	24.61±3.3	24.27±3.2	**23.79**±3.0		L-N-R	23.64±3.5	23.61±3.6	**23.34**±3.4	
L-K-N	24.01±3.0	24.05±3.2	**23.57**±3.0		L-K-R	24.30±3.2	24.49±2.9	**22.98**±2.9	
L-R-N	23.36±3.0	23.40±2.9	**23.26**±3.1		L-R-K	23.48±3.5	22.97±3.2	**22.66**±2.9	
N-L-K	22.98±2.9	22.98±2.9	**22.95**±3.2		N-L-R	23.26±3.2	23.25±3.1	**22.96**±3.2	
N-K-L	23.49±3.2	**23.29**±2.9	23.30±2.7		N-K-R	23.69±3.2	23.75±3.0	**23.30**±3.3	
N-R-L	**23.21**±3.5	23.50±3.7	23.59±3.9		N-R-K	23.88±3.0	23.69±3.2	**23.31**±3.3	
K-L-N	24.57±3.4	25.13±6.5	23.42±3.2	**22.85**±3.1	K-L-R	25.17±2.9	24.02±2.5	**23.24**±2.3	
K-N-L	24.72±3.0	24.12±3.0	**23.95**±2.7		K-N-R	24.57±2.7	23.48±2.9	**23.37**±2.9	
K-R-L	24.96±3.0	23.88±4.6	23.32±2.9	**22.95**±3.1	K-R-N	24.91±3.0	23.82±3.2	23.96±3.0	**22.69**±2.9
R-L-N	23.70±3.4	23.95±3.2	23.63±3.2	**22.70**±3.3	R-L-K	22.56±3.2	**22.29**±3.2	22.49±3.0	
R-N-L	**22.51**±2.9	22.68±2.8	22.56±2.7		R-N-K	22.79±2.9	22.83±3.0	**22.66**±3.0	
R-K-L	23.53±3.4	23.28±3.2	**22.79**±2.9		R-K-N	23.11±3.2	**22.59**±3.2	23.07±3.5	
L-N-K-R	24.46±2.9	24.10±3.0	**23.17**±2.8		L-N-R-K	23.80±3.3	23.77±3.3	**23.37**±3.3	
L-K-N-R	24.48±3.3	24.61±3.3	**23.72**±3.1		L-K-R-N	24.34±3.2	24.17±3.2	**22.79**±3.0	
L-R-N-K	24.18±3.4	23.93±3.3	**23.44**±3.4		L-R-K-N	24.43±2.9	24.22±3.0	**23.75**±3.2	
N-L-K-R	**22.84**±3.3	22.88±3.1	22.92±2.8		N-L-R-K	23.46±2.8	**23.18**±2.7	23.26±2.6	
N-K-L-R	23.82±3.2	23.42±3.3	**23.18**±3.2		N-K-R-L	23.58±3.2	23.90±3.6	**23.36**±3.1	
N-R-L-K	23.35±3.1	23.59±3.1	**23.00**±3.4		N-R-K-L	23.37±3.1	**23.27**±3.1	23.30±3.0	
K-L-N-R	24.96±3.0	25.09±7.5	**23.34**±2.9		K-L-R-N	24.55±3.4	23.93±3.2	**23.63**±3.3	
K-N-L-R	24.69±3.5	23.96±3.6	**23.58**±3.4		K-N-R-L	25.33±3.4	24.08±3.2	**23.92**±3.2	
K-R-L-N	25.05±3.1	24.40±4.0	23.83±3.2	**22.64**±3.1	K-R-N-L	24.88±3.7	**23.35**±3.3	23.45±3.3	
R-L-N-K	23.26±3.1	**23.20**±3.4	23.53±3.8		R-L-K-N	23.67±3.4	**22.91**±3.4	23.84±3.3	
R-N-L-K	23.13±3.1	**22.82**±3.0	22.94±3.1		R-N-K-L	23.71±3.5	22.92±3.3	**22.87**±3.0	
R-K-L-N	23.05±3.1	**22.89**±2.8	23.28±3.1		R-K-N-L	23.47±3.1	23.30±2.8	**23.13**±2.9	

Table A.4: Results for the DIABETES dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	24.43±2.8	24.43±2.8	**24.37**±2.8	**24.37**±2.8	N	24.83±3.2	24.83±3.2	**24.78**±3.2	**24.78**±3.2
K	**26.55**±3.0	**26.55**±3.0	26.69±2.9	26.69±2.9	R	23.59±2.8	23.59±2.8	**23.46**±2.8	**23.46**±2.8
L-N	25.95±2.7	26.39±2.7	25.83±3.0	**24.73**±2.9	L-K	**25.10**±3.3	25.23±3.1	26.01±3.2	
L-R	25.38±3.1	25.31±3.0	**23.81**±3.0		N-L	**25.72**±3.4	25.74±2.8	26.40±3.2	
N-K	**25.25**±2.8	25.36±2.9	25.72±2.8		N-R	26.06±2.7	25.83±2.8	**24.29**±2.7	
K-L	25.22±2.5	**24.38**±2.8	24.78±2.9	24.92±3.0	K-N	25.23±2.6	24.16±2.9	**23.96**±2.6	24.81±3.3
K-R	26.75±2.8	26.41±2.8	**23.78**±2.6	24.18±2.8	R-L	26.68±3.1	25.34±2.7	25.73±3.6	**24.76**±2.6
R-N	26.70±2.9	25.91±2.8	25.04±2.9	**24.62**±2.8	R-K	26.14±2.8	**26.01**±2.8	26.22±2.5	
L-N-K	25.81±3.0	26.68±3.1	**25.18**±2.8		L-N-R	25.50±2.7	25.94±2.4	**24.92**±2.4	
L-K-N	**25.87**±3.0	26.50±2.9	26.49±3.0		L-K-R	26.42±3.1	26.23±2.7	**24.00**±2.6	
L-R-N	26.35±3.2	26.71±3.4	**26.32**±3.3		L-R-K	**25.43**±2.9	25.80±2.9	25.84±2.9	
N-L-K	25.99±3.0	26.13±5.4	**25.70**±2.9		N-L-R	25.98±2.5	25.18±2.7	**23.78**±2.6	
N-K-L	25.76±2.8	**25.74**±2.8	26.16±3.1		N-K-R	26.41±3.1	25.50±3.0	**24.03**±2.9	
N-R-L	25.10±3.0	**24.48**±3.1	26.23±4.4		N-R-K	25.67±2.8	25.89±3.0	**25.22**±2.9	
K-L-N	26.16±2.8	26.16±3.1	26.31±2.8	**24.74**±2.7	K-L-R	25.96±3.5	26.40±4.7	**23.59**±2.8	
K-N-L	26.87±3.2	**25.68**±5.6	26.02±3.3		K-N-R	27.27±3.2	26.72±3.2	**24.05**±3.3	
K-R-L	26.43±2.7	25.33±2.8	25.33±3.7	**23.91**±2.7	K-R-N	26.59±2.5	25.63±2.3	25.90±3.3	**23.46**±2.7
R-L-N	25.68±3.3	26.28±3.4	25.53±3.4	**24.01**±2.8	R-L-K	27.03±3.3	25.74±5.8	**25.03**±2.9	
R-N-L	25.36±2.8	**24.50**±3.0	26.08±3.8		R-N-K	27.47±3.0	26.98±3.1	**25.52**±2.8	
R-K-L	27.38±3.2	26.36±3.1	**25.75**±3.4		R-K-N	26.88±3.2	26.00±3.1	**24.84**±3.0	
L-N-K-R	26.36±2.8	26.49±2.9	**24.67**±2.8		L-N-R-K	25.52±3.1	26.65±3.3	**24.82**±2.9	
L-K-N-R	25.73±2.9	26.92±2.9	**25.00**±3.0		L-K-R-N	**25.66**±3.1	26.28±3.0	25.90±3.2	
L-R-N-K	25.73±2.9	26.67±2.7	**24.79**±2.7		L-R-K-N	25.88±3.4	26.28±3.2	**25.71**±2.9	
N-L-K-R	26.14±2.6	26.10±2.7	**24.23**±2.6		N-L-R-K	25.88±2.9	26.06±2.9	**25.31**±3.2	
N-K-L-R	25.91±2.8	25.59±3.0	**23.97**±3.0		N-K-R-L	26.68±2.8	**26.38**±2.8	26.72±3.3	
N-R-L-K	25.92±2.9	**25.02**±2.8	25.13±2.9		N-R-K-L	25.95±3.1	**24.77**±2.8	26.19±3.8	
K-L-N-R	27.29±3.0	27.30±6.6	**25.24**±3.0		K-L-R-N	26.92±2.9	**26.20**±3.1	26.24±2.8	
K-N-L-R	27.45±2.8	26.06±2.8	**24.05**±2.9		K-N-R-L	27.58±3.3	**25.49**±3.1	26.46±3.4	
K-R-L-N	27.22±2.8	27.65±3.0	25.93±3.4	**24.22**±2.8	K-R-N-L	27.91±3.0	26.27±2.9	**26.15**±2.9	
R-L-N-K	26.27±2.9	25.93±2.5	**25.39**±2.7		R-L-K-N	27.23±3.1	25.97±3.1	**25.09**±2.9	
R-N-L-K	25.99±2.9	25.07±3.1	**24.85**±2.4		R-N-K-L	26.93±2.9	**25.12**±2.8	25.79±3.2	
R-K-L-N	26.19±3.2	26.31±3.3	**25.05**±3.3		R-K-N-L	26.34±2.6	**24.83**±2.6	26.45±4.1	

Table A.5: Results for the GERMAN dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	40.79±6.6	40.79±6.6	**40.61**±6.7	**40.61**±6.7	N	**38.77**±6.6	**38.77**±6.6	**38.77**±6.6	**38.77**±6.6
K	33.11±6.8	33.11±6.8	**32.90**±7.0	**32.90**±7.0	R	**30.41**±6.3	**30.41**±6.3	30.91±6.2	30.91±6.2
L-N	46.88±7.8	46.23±7.9	40.23±7.9	**39.38**±7.0	L-K	**32.72**±6.5	34.01±6.6	35.22±6.8	
L-R	39.45±9.2	34.10±8.1	**32.56**±6.8		N-L	40.34±6.7	**39.28**±6.5	40.11±6.2	
N-K	**32.36**±6.4	33.44±6.2	33.79±6.3		N-R	40.46±7.2	37.65±6.9	**32.19**±6.7	
K-L	33.89±5.8	32.79±6.1	**32.16**±6.0	39.61±7.7	K-N	33.15±6.0	32.88±6.2	**31.39**±6.6	39.13±7.2
K-R	36.77±7.8	34.70±6.6	33.09±6.6	**31.81**±6.9	R-L	30.56±6.8	<u>30.16±6.5</u>	30.59±5.8	40.87±7.3
R-N	31.29±5.9	**30.80**±5.9	31.43±5.5	38.45±6.4	R-K	33.84±5.9	**31.17**±6.6	31.91±7.1	
L-N-K	**33.48**±6.9	37.42±7.7	35.75±7.7		L-N-R	36.88±10.1	34.44±7.9	**32.93**±6.8	
L-K-N	**34.53**±7.2	36.42±7.3	36.03±6.5		L-K-R	34.48±5.9	36.44±6.4	**32.62**±6.3	
L-R-N	41.20±9.8	35.62±7.7	**34.05**±7.3		L-R-K	**32.89**±6.6	34.29±6.3	33.49±6.7	
N-L-K	**34.20**±6.5	34.21±7.1	35.02±8.2		N-L-R	40.80±7.3	38.61±7.6	**34.12**±6.7	
N-K-L	32.70±6.7	**32.66**±7.0	33.53±7.3		N-K-R	34.59±6.4	33.90±6.5	**33.67**±5.9	
N-R-L	39.25±6.7	36.04±7.3	**31.23**±6.4		N-R-K	33.27±6.5	33.72±5.5	**32.50**±5.3	
K-L-N	33.49±7.6	33.56±7.9	**32.84**±7.3	38.71±6.0	K-L-R	33.98±7.2	34.43±8.6	**33.27**±6.0	
K-N-L	34.40±5.9	**32.96**±6.1	33.90±5.7		K-N-R	33.89±7.0	32.90±7.0	**31.54**±6.7	
K-R-L	35.04±6.0	**31.49**±5.9	31.64±6.3	40.99±5.5	K-R-N	35.95±6.4	33.72±6.0	**31.84**±6.5	38.47±7.2
R-L-N	**30.94**±6.3	31.19±6.3	31.85±6.3	37.49±7.2	R-L-K	34.87±6.7	**32.75**±5.2	33.19±7.3	
R-N-L	32.62±7.1	**31.38**±7.0	33.67±7.0		R-N-K	32.81±6.2	31.08±6.5	**30.67**±6.8	
R-K-L	33.70±7.1	**31.04**±6.4	31.24±6.9		R-K-N	33.09±7.4	**30.53**±7.0	31.46±7.9	
L-N-K-R	35.27±7.1	38.22±6.8	**33.40**±6.3		L-N-R-K	**34.02**±6.5	35.48±7.4	34.44±7.2	
L-K-N-R	33.52±6.2	34.98±6.4	**32.57**±5.9		L-K-R-N	35.18±7.5	36.57±6.8	**35.07**±7.2	
L-R-N-K	**33.53**±5.4	35.19±6.6	33.58±6.5		L-R-K-N	34.94±7.6	34.98±7.4	**34.22**±6.4	
N-L-K-R	36.53±7.1	35.98±7.4	**33.25**±6.8		N-L-R-K	**33.69**±6.4	36.21±7.2	34.02±7.4	
N-K-L-R	32.82±7.2	34.10±7.6	**32.32**±8.3		N-K-R-L	34.92±7.0	**33.44**±6.3	34.16±7.1	
N-R-L-K	34.57±7.0	34.39±6.8	**32.61**±6.7		N-R-K-L	33.89±5.8	34.22±6.0	**33.10**±6.0	
K-L-N-R	34.55±6.9	33.80±6.5	**32.62**±7.0		K-L-R-N	33.18±6.8	32.75±6.9	**31.92**±6.5	
K-N-L-R	35.43±7.6	34.34±7.2	**34.29**±6.5		K-N-R-L	34.98±7.3	**33.28**±6.9	33.82±6.4	
K-R-L-N	34.48±6.0	34.17±6.4	**32.00**±5.7	38.49±6.6	K-R-N-L	35.58±7.0	**32.73**±6.9	34.25±7.3	
R-L-N-K	33.39±7.8	30.82±7.9	**30.70**±6.9		R-L-K-N	34.95±6.3	31.66±6.7	**31.14**±7.1	
R-N-L-K	34.07±6.4	**30.96**±6.4	31.98±7.3		R-N-K-L	32.92±6.2	**30.39**±6.7	32.02±6.1	
R-K-L-N	34.03±6.4	**32.16**±6.1	32.19±6.7		R-K-N-L	32.52±5.7	**30.42**±6.3	32.60±6.2	

Table A.6: Results for the GLASS dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	15.70±5.9	15.70±5.9	**15.62**±5.9	**15.62**±5.9	N	15.00±6.8	15.00±6.8	**14.93**±6.9	**14.93**±6.9
K	15.52±6.5	15.52±6.5	**15.40**±6.5	**15.40**±6.5	R	15.91±6.3	15.91±6.3	**15.80**±6.3	**15.80**±6.3
L-N	16.63±6.5	16.69±6.4	**15.37**±6.0	15.66±6.6	L-K	16.46±6.9	**15.66**±6.6	15.79±6.3	
L-R	15.99±6.7	15.67±6.2	**14.32**±6.0		N-L	16.18±6.5	**16.07**±5.1	16.97±5.7	
N-K	15.98±6.6	16.38±6.7	**15.62**±5.8		N-R	15.17±5.3	**15.11**±5.3	15.47±5.4	
K-L	15.99±6.2	15.83±6.3	16.17±5.9	**15.22**±6.1	K-N	16.05±7.0	15.44±6.5	**15.02**±6.9	15.12±5.2
K-R	16.72±7.1	16.69±7.1	**14.64**±6.3	15.39±5.8	R-L	17.22±6.5	18.08±6.6	16.88±7.4	**14.76**±6.2
R-N	16.39±6.3	15.85±6.1	**14.36**±6.0	15.22±6.6	R-K	17.93±6.0	16.92±6.3	**16.79**±4.7	
L-N-K	16.90±6.3	16.24±5.9	**15.60**±5.8		L-N-R	17.04±6.6	16.63±7.0	**15.38**±6.3	
L-K-N	17.10±6.0	16.50±5.8	**14.61**±6.1		L-K-R	18.03±6.6	16.15±6.2	**15.53**±6.0	
L-R-N	16.78±6.3	16.96±6.6	**15.33**±6.4		L-R-K	16.55±6.0	16.43±6.3	**15.97**±6.0	
N-L-K	17.19±6.8	17.42±7.0	**16.29**±7.0		N-L-R	16.33±5.7	18.43±6.3	**15.88**±6.0	
N-K-L	**15.98**±6.0	17.05±6.1	17.45±7.1		N-K-R	17.72±6.8	17.19±6.7	**15.96**±6.1	
N-R-L	**15.71**±6.4	16.82±6.7	18.23±7.1		N-R-K	15.90±6.5	**15.32**±6.3	15.39±6.2	
K-L-N	16.31±6.2	16.65±6.7	15.36±6.1	**14.85**±5.6	K-L-R	17.64±5.8	17.87±6.1	**16.07**±5.8	
K-N-L	**15.97**±5.7	17.35±9.6	17.89±6.5		K-N-R	16.18±6.0	15.37±5.9	**14.82**±6.1	
K-R-L	17.26±7.2	17.94±9.9	17.81±7.5	**14.86**±6.0	K-R-N	16.86±5.9	16.94±6.4	**15.48**±6.5	15.73±6.4
R-L-N	16.38±7.0	18.47±7.0	15.09±7.1	**14.81**±7.2	R-L-K	16.65±6.0	15.82±6.4	**15.30**±5.5	
R-N-L	18.62±6.8	**18.12**±6.4	18.50±5.8		R-N-K	17.57±5.7	17.21±6.4	**14.80**±5.8	
R-K-L	18.28±7.2	18.78±7.1	**17.69**±6.3		R-K-N	16.75±6.4	17.18±6.1	**14.87**±5.9	
L-N-K-R	17.44±6.7	16.35±7.2	**15.60**±6.4		L-N-R-K	17.80±6.2	17.89±5.8	**16.12**±5.7	
L-K-N-R	16.43±5.8	15.98±6.6	**15.38**±5.1		L-K-R-N	17.15±6.1	17.39±6.5	**15.49**±5.9	
L-R-N-K	17.70±7.0	17.99±7.3	**16.01**±7.1		L-R-K-N	17.65±6.8	17.88±7.5	**15.48**±6.1	
N-L-K-R	16.60±6.2	17.99±6.3	**15.73**±5.5		N-L-R-K	15.88±6.3	18.37±7.0	**15.78**±6.2	
N-K-L-R	15.61±6.3	17.94±6.9	**15.51**±6.1		N-K-R-L	**16.84**±6.5	16.85±6.5	18.21±7.7	
N-R-L-K	16.77±6.1	17.24±6.1	**15.97**±6.4		N-R-K-L	**15.54**±7.0	15.69±6.3	17.43±8.2	
K-L-N-R	16.82±6.8	16.78±6.6	**15.81**±7.1		K-L-R-N	16.96±7.0	17.30±7.8	**14.69**±6.9	
K-N-L-R	16.57±6.0	17.07±6.5	**15.32**±6.0		K-N-R-L	16.87±6.3	**16.72**±6.5	17.35±6.7	
K-R-L-N	16.78±6.0	18.32±7.4	15.17±6.5	**14.01**±5.7	K-R-N-L	**16.79**±5.9	17.51±6.1	18.18±6.5	
R-L-N-K	16.84±6.2	18.93±7.6	**15.88**±6.3		R-L-K-N	17.79±7.0	19.71±7.8	**15.81**±6.2	
R-N-L-K	16.78±6.3	16.65±6.1	**15.29**±5.5		R-N-K-L	16.66±7.2	**16.43**±7.2	18.89±9.1	
R-K-L-N	18.02±6.8	19.75±6.8	**17.77**±5.8		R-K-N-L	17.15±6.6	**16.90**±6.6	17.86±7.9	

Table A.7: Results for the HEPATITIS dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**12.28**±4.0	**12.28**±4.0	12.63±4.1	12.63±4.1	N	**11.21**±3.6	**11.21**±3.6	11.36±3.5	11.36±3.5
K	**15.04**±3.7	**15.04**±3.7	15.36±3.8	15.36±3.8	R	**8.15**±2.6	**8.15**±2.6	8.24±2.7	8.24±2.7
L-N	12.14±3.8	12.11±3.7	12.66±3.6	**11.88**±3.8	L-K	**13.06**±3.5	13.19±3.4	14.99±3.6	
L-R	11.39±3.7	11.66±3.8	**8.34**±3.3		N-L	12.30±3.5	12.11±3.6	**11.94**±3.6	
N-K	11.20±3.3	**11.13**±3.5	13.83±4.1		N-R	10.73±3.4	10.60±3.4	**8.72**±3.1	
K-L	**11.97**±3.9	12.44±3.7	12.12±3.7	12.95±3.5	K-N	11.24±3.9	11.78±4.0	**10.07**±3.5	11.74±3.7
K-R	11.19±4.4	11.20±4.7	8.59±3.0	**8.43**±3.0	R-L	8.29±3.1	8.00±2.9	**7.60**±2.8	12.78±3.7
R-N	7.32±2.9	**7.19**±2.7	8.02±3.0	12.12±3.6	R-K	**7.90**±3.2	7.90±3.2	12.63±4.1	
L-N-K	12.23±4.0	**12.04**±3.9	13.36±4.1		L-N-R	11.27±3.1	11.54±3.4	**8.73**±3.3	
L-K-N	11.15±3.9	11.63±4.1	**10.46**±3.7		L-K-R	12.53±3.6	12.97±3.6	**9.33**±3.5	
L-R-N	10.61±3.9	10.14±3.7	**9.25**±3.5		L-R-K	11.08±3.5	**11.03**±3.6	12.13±3.6	
N-L-K	12.50±4.1	**12.40**±4.2	13.32±4.3		N-L-R	12.34±3.8	11.90±3.8	**8.94**±3.5	
N-K-L	11.82±3.4	11.62±3.3	**11.42**±3.3		N-K-R	10.82±3.8	11.08±3.9	**8.88**±3.5	
N-R-L	11.52±3.8	11.25±3.9	**9.02**±3.8		N-R-K	**10.64**±3.2	10.75±3.1	12.54±3.6	
K-L-N	11.48±3.9	12.19±3.8	**10.29**±3.6	12.54±3.8	K-L-R	10.48±3.5	10.96±3.5	**8.31**±3.3	
K-N-L	12.21±3.8	12.41±3.9	**10.71**±3.6		K-N-R	10.49±3.6	10.88±4.0	**8.91**±3.5	
K-R-L	10.66±3.6	10.25±3.7	**8.32**±3.1	8.43±2.9	K-R-N	10.13±4.0	10.08±3.7	**8.04**±3.3	8.28±3.1
R-L-N	8.60±3.5	**7.53**±2.9	7.98±2.7	13.01±3.8	R-L-K	8.31±3.2	**7.86**±3.2	12.60±3.8	
R-N-L	**7.26**±2.9	7.27±3.1	7.30±3.0		R-N-K	7.28±2.8	**7.03**±2.8	11.90±3.7	
R-K-L	8.45±3.5	8.45±3.8	**8.26**±3.4		R-K-N	7.34±3.0	**6.79**±3.0	7.49±3.3	
L-N-K-R	11.70±4.0	12.24±4.3	**8.82**±3.4		L-N-R-K	12.46±3.8	**12.16**±3.8	13.80±4.4	
L-K-N-R	10.00±3.5	10.59±3.6	**8.95**±3.4		L-K-R-N	11.17±3.6	11.57±3.4	**8.95**±3.1	
L-R-N-K	9.63±3.5	**9.17**±3.2	12.14±4.2		L-R-K-N	10.40±3.6	10.40±3.7	**8.86**±2.9	
N-L-K-R	12.26±3.0	11.98±3.2	**9.14**±2.7		N-L-R-K	11.90±3.6	**11.43**±3.6	12.35±3.9	
N-K-L-R	11.58±3.7	11.26±3.8	**8.75**±3.5		N-K-R-L	11.79±3.3	11.59±3.1	**9.43**±3.4	
N-R-L-K	11.25±3.6	**10.81**±3.7	12.06±3.9		N-R-K-L	11.72±3.6	11.21±3.4	**8.71**±3.1	
K-L-N-R	10.68±3.8	10.93±4.4	**8.78**±3.3		K-L-R-N	9.95±3.6	10.31±3.7	**8.16**±2.9	
K-N-L-R	10.98±3.6	11.47±3.4	**8.84**±3.1		K-N-R-L	10.70±3.4	11.09±4.0	**9.16**±3.1	
K-R-L-N	11.04±3.9	10.91±4.0	**7.89**±3.1	11.13±3.4	K-R-N-L	10.89±3.7	10.67±3.7	**7.96**±2.8	
R-L-N-K	7.82±3.3	**7.05**±2.9	10.89±3.2		R-L-K-N	7.59±3.1	**6.95**±3.0	7.04±2.9	
R-N-L-K	8.14±2.9	**8.07**±2.9	11.16±3.6		R-N-K-L	7.83±3.1	7.98±3.1	**7.63**±2.9	
R-K-L-N	8.20±2.9	7.62±3.2	**7.31**±2.7		R-K-N-L	7.68±3.3	7.78±3.1	**7.59**±3.2	

Table A.8: Results for the IONOSPHERE dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	16.78±6.9	16.78±6.9	**16.63**±6.9	**16.63**±6.9	N	**4.79**±3.4	**4.79**±3.4	**4.79**±3.4	**4.79**±3.4
K	**4.23**±3.4	**4.23**±3.4	4.26±3.5	4.26±3.5	R	**4.72**±3.2	**4.72**±3.2	4.78±3.2	4.78±3.2
L-N	10.03±7.6	8.28±7.3	**4.72**±3.8	4.98±3.8	L-K	4.33±3.5	4.76±3.8	**4.17**±3.6	
L-R	**5.44**±4.1	5.46±4.0	5.49±3.8		N-L	4.78±3.3	3.27±2.6	**3.26**±2.6	
N-K	**4.72**±3.6	5.20±4.1	5.09±3.9		N-R	4.92±3.4	**4.66**±3.6	4.73±3.5	
K-L	**4.43**±3.2	5.47±9.6	5.78±10.6	16.32±7.2	K-N	3.52±3.1	**3.44**±3.2	3.84±3.3	4.70±3.4
K-R	4.07±3.5	4.07±3.6	**3.95**±3.7	4.72±3.6	R-L	4.67±4.0	4.59±4.0	**4.50**±4.0	18.10±6.6
R-N	**4.39**±3.3	4.72±3.2	5.01±3.3	5.17±3.9	R-K	6.04±3.9	5.50±3.7	**4.81**±3.6	
L-N-K	4.64±3.6	4.74±3.3	**4.42**±3.1		L-N-R	5.35±4.0	**4.36**±3.7	4.49±3.6	
L-K-N	5.06±3.4	5.58±3.6	**4.96**±3.4		L-K-R	5.29±3.9	5.09±3.4	**4.49**±3.2	
L-R-N	5.10±4.0	**4.42**±3.7	4.43±3.6		L-R-K	5.47±3.9	4.59±3.8	**4.08**±3.4	
N-L-K	4.41±3.7	**3.72**±3.4	3.73±3.4		N-L-R	5.50±3.7	**3.22**±2.9	4.00±3.3	
N-K-L	4.47±3.6	**4.20**±3.7	5.44±9.8		N-K-R	**4.18**±3.3	4.40±3.6	4.30±3.6	
N-R-L	5.40±3.5	4.51±3.5	**4.25**±3.2		N-R-K	5.01±3.9	**4.26**±4.1	4.47±3.9	
K-L-N	**4.09**±3.5	5.33±10.0	4.61±4.3	5.36±4.0	K-L-R	4.54±3.8	5.49±9.3	**4.49**±3.9	
K-N-L	4.17±3.6	**3.94**±3.7	4.64±7.0		K-N-R	4.54±3.4	4.64±3.6	**4.50**±3.7	
K-R-L	4.74±3.4	**4.72**±3.7	5.21±7.4	17.06±6.1	K-R-N	4.22±3.0	**4.15**±3.1	4.37±3.2	4.98±4.0
R-L-N	4.06±3.5	**3.73**±3.3	4.40±3.6	4.20±3.6	R-L-K	5.39±3.6	5.05±4.0	**4.50**±3.7	
R-N-L	4.49±3.7	**4.31**±3.5	4.45±3.5		R-N-K	5.49±3.6	4.94±3.4	**4.80**±3.5	
R-K-L	5.65±3.7	4.98±3.4	**4.39**±3.3		R-K-N	5.43±3.3	5.23±3.6	**4.97**±3.5	
L-N-K-R	5.28±3.7	4.87±3.6	**4.47**±3.4		L-N-R-K	6.01±4.1	5.19±4.1	**4.89**±3.4	
L-K-N-R	4.92±3.9	5.29±3.4	**4.35**±3.3		L-K-R-N	5.11±4.0	4.65±3.6	**4.53**±3.9	
L-R-N-K	5.35±3.8	4.15±3.9	**4.00**±3.2		L-R-K-N	5.76±3.7	4.69±3.6	**4.23**±3.7	
N-L-K-R	4.55±3.7	**3.48**±3.1	3.78±3.4		N-L-R-K	5.01±3.5	**3.41**±3.3	4.17±3.2	
N-K-L-R	4.79±3.4	**4.20**±3.7	4.73±4.1		N-K-R-L	4.87±3.1	**3.95**±3.0	6.39±11.0	
N-R-L-K	5.29±4.3	**3.90**±3.7	4.35±3.7		N-R-K-L	5.03±3.9	**4.03**±3.6	4.48±5.6	
K-L-N-R	**4.15**±4.0	5.06±9.3	4.70±4.9		K-L-R-N	4.52±3.4	4.80±4.3	**4.31**±3.4	
K-N-L-R	**5.01**±3.4	5.19±5.2	5.02±3.8		K-N-R-L	**3.68**±3.2	4.47±7.0	3.79±3.6	
K-R-L-N	4.40±3.5	5.34±9.2	**4.28**±3.4	17.26±7.1	K-R-N-L	4.41±3.2	4.22±3.3	**3.95**±3.2	
R-L-N-K	5.63±3.8	5.47±3.9	**4.99**±3.4		R-L-K-N	5.10±3.6	4.24±3.5	**4.05**±3.2	
R-N-L-K	5.04±3.4	**4.27**±3.4	4.41±3.4		R-N-K-L	6.02±3.7	5.42±3.7	**4.79**±3.0	
R-K-L-N	5.32±4.1	**4.91**±3.9	4.96±3.6		R-K-N-L	5.76±3.8	5.28±3.8	**4.63**±3.6	

Table A.9: Results for the IRIS dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	23.10±6.2	23.10±6.2	**23.05**±6.4	**23.05**±6.4	N	21.72±5.7	21.72±5.7	**21.57**±5.7	**21.57**±5.7
K	**16.49**±5.3	**16.49**±5.3	16.66±5.5	16.66±5.5	R	**13.12**±5.2	**13.12**±5.2	13.26±5.3	13.26±5.3
L-N	25.71±7.4	24.85±6.6	**20.95**±6.1	20.98±5.6	L-K	**14.83**±6.0	15.16±6.2	16.89±6.7	
L-R	14.16±5.2	14.18±5.3	**13.25**±5.2		N-L	22.31±6.7	**21.80**±6.3	22.41±6.3	
N-K	17.55±6.9	17.54±7.1	**16.50**±6.2		N-R	17.41±6.6	16.79±6.6	**13.13**±5.5	
K-L	16.88±6.3	16.82±6.2	**15.12**±5.8	21.55±6.4	K-N	17.05±5.8	17.03±5.8	**16.64**±6.0	20.47±6.2
K-R	14.48±5.2	14.36±5.0	**11.86**±5.1	12.72±5.5	R-L	**13.51**±5.0	14.03±5.4	13.59±4.8	22.71±6.3
R-N	**14.38**±4.9	14.71±4.8	16.71±6.1	21.67±6.0	R-K	**14.60**±5.5	14.79±5.5	16.20±5.9	
L-N-K	15.93±6.6	**15.29**±6.0	16.28±5.9		L-N-R	14.35±4.9	14.49±4.5	**14.25**±4.9	
L-K-N	17.30±6.3	17.08±5.9	**16.96**±5.5		L-K-R	15.19±5.3	15.68±5.3	**11.90**±4.7	
L-R-N	**15.86**±5.5	15.95±6.0	16.22±5.6		L-R-K	**15.13**±5.0	15.13±5.3	16.20±5.4	
N-L-K	17.66±6.2	17.93±6.1	**16.25**±5.9		N-L-R	17.54±6.4	17.57±6.5	**13.86**±5.6	
N-K-L	18.83±6.6	**17.80**±5.6	18.63±5.5		N-K-R	17.94±5.8	18.44±6.4	**12.97**±5.7	
N-R-L	19.11±6.0	18.00±6.1	**16.19**±5.2		N-R-K	17.37±5.6	16.88±6.1	**15.49**±5.7	
K-L-N	17.95±6.4	17.20±6.3	**15.34**±5.2	20.90±5.5	K-L-R	16.55±5.8	15.99±5.9	**11.29**±3.9	
K-N-L	17.59±6.5	17.49±6.5	**17.42**±6.5		K-N-R	16.80±6.1	16.36±5.6	**12.68**±5.0	
K-R-L	16.80±5.4	16.50±5.8	14.14±5.0	**13.74**±4.8	K-R-N	15.82±6.4	15.55±5.9	14.17±4.6	**13.71**±5.0
R-L-N	14.93±5.6	**13.97**±5.5	14.98±6.1	22.71±6.5	R-L-K	15.39±5.9	15.36±5.7	**15.22**±5.1	
R-N-L	**15.17**±5.4	15.78±5.5	16.28±5.8		R-N-K	16.07±5.6	15.01±6.0	**14.63**±6.1	
R-K-L	15.44±6.0	15.36±5.8	**14.78**±5.2		R-K-N	16.03±6.2	15.38±5.7	**14.95**±5.7	
L-N-K-R	17.82±6.4	17.86±6.1	**13.62**±5.6		L-N-R-K	16.03±5.6	15.89±6.1	**15.80**±5.1	
L-K-N-R	17.55±5.6	17.08±5.5	**13.15**±4.9		L-K-R-N	17.06±5.8	17.39±5.9	**15.27**±4.9	
L-R-N-K	16.34±5.5	16.62±6.1	**15.97**±5.9		L-R-K-N	16.30±5.0	16.57±5.7	**14.89**±5.1	
N-L-K-R	18.33±6.1	18.88±6.2	**14.17**±5.3		N-L-R-K	16.38±5.3	16.55±5.3	**15.65**±5.6	
N-K-L-R	18.16±5.9	17.23±5.4	**12.84**±5.0		N-K-R-L	18.08±5.8	17.59±5.9	**16.20**±5.7	
N-R-L-K	18.78±6.6	18.27±6.3	**15.32**±5.4		N-R-K-L	18.09±5.8	17.31±5.9	**15.64**±6.9	
K-L-N-R	17.81±6.0	17.86±5.5	**11.87**±5.1		K-L-R-N	17.89±5.5	17.81±6.0	**14.99**±5.1	
K-N-L-R	17.24±5.6	17.55±5.4	**13.03**±4.7		K-N-R-L	16.79±6.0	16.63±6.0	**15.89**±6.2	
K-R-L-N	16.98±5.9	16.80±6.3	**14.20**±5.1	18.97±5.9	K-R-N-L	16.70±6.1	16.94±6.2	**16.09**±6.8	
R-L-N-K	15.82±5.6	14.86±5.8	**14.28**±5.8		R-L-K-N	16.52±6.6	15.28±5.8	**14.82**±4.6	
R-N-L-K	15.26±5.3	15.88±5.8	**14.78**±5.7		R-N-K-L	15.48±5.4	16.03±5.9	**15.33**±5.6	
R-K-L-N	16.42±5.4	15.91±4.6	**14.60**±5.3		R-K-N-L	15.58±5.4	15.31±6.0	**15.03**±5.7	

Table A.10: Results for the SONAR dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**26.08**±4.9	**26.08**±4.9	**26.08**±4.9	**26.08**±4.9	N	25.41±5.1	25.41±5.1	**25.37**±4.9	**25.37**±4.9
K	28.57±5.3	28.57±5.3	**28.09**±5.2	**28.09**±5.2	R	26.15±4.8	26.15±4.8	**26.00**±4.8	**26.00**±4.8
L-N	26.05±4.9	25.38±4.7	25.20±5.3	**24.78**±5.1	L-K	26.25±5.2	**26.10**±5.2	26.71±5.2	
L-R	**26.06**±5.0	26.25±5.0	26.88±5.5		N-L	**25.07**±4.3	25.58±4.7	25.93±4.5	
N-K	**25.48**±5.1	25.60±5.1	27.01±5.4		N-R	25.77±5.5	**25.51**±5.4	27.49±6.2	
K-L	27.57±5.0	26.66±5.0	25.90±5.1	**25.72**±5.1	K-N	28.35±5.2	27.05±6.1	26.53±5.6	**24.59**±4.5
K-R	28.03±5.0	28.20±5.4	28.60±6.0	**26.35**±6.0	R-L	27.36±5.4	26.91±5.3	26.31±5.3	**25.90**±5.2
R-N	26.01±5.1	25.90±4.8	**25.24**±4.9	25.33±5.0	R-K	**25.13**±5.3	25.59±5.3	27.37±5.7	
L-N-K	26.34±5.1	**25.42**±5.3	25.93±4.6		L-N-R	26.19±5.4	**25.82**±5.6	26.66±5.8	
L-K-N	25.39±4.5	**25.17**±4.4	25.74±4.4		L-K-R	26.64±5.2	**26.05**±5.3	28.18±5.8	
L-R-N	25.89±5.3	**25.70**±5.3	25.96±5.1		L-R-K	26.54±4.7	**25.63**±4.8	26.64±5.0	
N-L-K	**26.04**±4.8	26.51±4.8	27.28±5.0		N-L-R	**25.93**±4.3	26.56±4.6	27.59±4.7	
N-K-L	**26.05**±5.3	26.64±5.4	26.32±5.5		N-K-R	26.41±5.0	**26.12**±5.3	27.80±5.1	
N-R-L	**24.66**±4.7	24.86±4.4	27.10±5.4		N-R-K	26.05±5.0	**26.01**±4.9	27.42±5.1	
K-L-N	27.33±5.0	28.87±11.7	**25.23**±4.9	25.40±4.8	K-L-R	27.32±5.6	**26.98**±6.8	28.17±4.8	
K-N-L	26.92±5.4	**26.04**±5.3	26.48±5.7		K-N-R	27.91±5.8	**26.32**±5.5	28.54±5.2	
K-R-L	26.74±5.6	25.94±5.4	26.24±5.0	**25.79**±5.2	K-R-N	26.87±5.6	26.83±5.5	27.67±5.3	**26.04**±5.2
R-L-N	26.45±5.0	26.24±4.8	**25.79**±4.8	25.85±5.8	R-L-K	25.79±5.5	**25.64**±5.4	26.83±5.3	
R-N-L	**26.11**±5.0	26.21±5.3	27.04±5.4		R-N-K	**27.03**±4.9	27.10±5.0	28.04±5.0	
R-K-L	25.60±5.3	25.88±5.4	**25.28**±5.6		R-K-N	**25.47**±5.2	25.78±5.0	25.82±5.1	
L-N-K-R	26.73±5.2	**25.87**±5.1	27.62±5.1		L-N-R-K	27.32±5.6	**26.47**±4.7	27.49±5.4	
L-K-N-R	26.72±4.0	**26.51**±4.3	28.55±5.3		L-K-R-N	25.92±5.0	**25.90**±4.8	27.02±5.1	
L-R-N-K	25.86±5.0	**25.81**±5.2	27.13±5.3		L-R-K-N	26.17±6.1	**26.01**±6.0	26.37±6.0	
N-L-K-R	25.71±4.6	**25.67**±4.4	27.57±5.2		N-L-R-K	**25.64**±4.7	26.14±5.0	26.91±4.9	
N-K-L-R	**26.62**±4.5	26.76±4.8	28.25±5.7		N-K-R-L	**26.93**±5.1	26.95±5.6	28.26±5.3	
N-R-L-K	25.39±5.4	**25.39**±5.4	27.13±5.0		N-R-K-L	**25.78**±4.6	26.43±4.9	27.26±5.1	
K-L-N-R	**25.90**±4.8	27.28±7.5	27.61±5.4		K-L-R-N	26.94±5.2	**26.46**±7.4	27.67±5.4	
K-N-L-R	**26.55**±5.2	27.06±6.9	28.46±5.9		K-N-R-L	26.84±5.2	**26.13**±5.6	27.32±6.3	
K-R-L-N	27.10±5.3	26.75±6.6	27.51±5.5	**24.84**±4.8	K-R-N-L	28.31±5.2	**27.36**±5.4	28.14±5.0	
R-L-N-K	**25.16**±5.3	25.56±4.9	26.76±5.5		R-L-K-N	**25.76**±4.9	26.62±6.8	26.26±5.0	
R-N-L-K	**26.19**±5.0	26.36±5.2	27.57±5.6		R-N-K-L	**25.99**±5.6	26.50±5.6	26.72±5.5	
R-K-L-N	26.11±4.6	**26.04**±4.6	26.84±4.8		R-K-N-L	**25.44**±4.9	25.74±4.5	25.94±4.9	

Table A.11: Results for the SURVIVAL dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**23.91**$\pm$2.8	**23.91**$\pm$2.8	23.94$\pm$2.8	23.94$\pm$2.8	N	**19.09**$\pm$2.9	**19.09**$\pm$2.9	**19.09**$\pm$2.9	**19.09**$\pm$2.9
K	28.36$\pm$3.4	28.36$\pm$3.4	**28.26**$\pm$3.4	**28.26**$\pm$3.4	R	21.63$\pm$2.8	21.63$\pm$2.8	**21.59**$\pm$2.8	**21.59**$\pm$2.8
L-N	21.23$\pm$2.8	20.61$\pm$2.9	**18.88**$\pm$2.9	19.25$\pm$2.8	L-K	22.10$\pm$3.1	**21.32**$\pm$3.1	22.92$\pm$3.2	
L-R	19.53$\pm$3.0	19.06$\pm$2.8	**18.86**$\pm$2.8		N-L	19.72$\pm$2.8	**18.42**$\pm$3.2	18.46$\pm$3.1	
N-K	19.24$\pm$3.0	**19.13**$\pm$3.1	22.20$\pm$3.1		N-R	19.86$\pm$2.7	19.28$\pm$2.8	**18.61**$\pm$2.8	
K-L	25.18$\pm$3.2	**21.44**$\pm$3.0	21.60$\pm$3.0	24.38$\pm$3.1	K-N	25.71$\pm$3.0	23.01$\pm$2.8	20.54$\pm$2.6	**19.13**$\pm$3.0
K-R	27.43$\pm$3.9	**20.88**$\pm$3.0	21.75$\pm$3.0	21.78$\pm$2.8	R-L	19.67$\pm$2.7	19.10$\pm$2.7	**18.89**$\pm$2.7	24.29$\pm$3.2
R-N	21.36$\pm$2.7	20.89$\pm$2.9	19.24$\pm$2.5	**18.85**$\pm$3.0	R-K	21.86$\pm$3.4	**21.43**$\pm$3.0	23.53$\pm$3.1	
L-N-K	20.84$\pm$3.4	**20.22**$\pm$3.1	20.67$\pm$3.1		L-N-R	21.49$\pm$3.3	**19.13**$\pm$3.1	19.24$\pm$2.9	
L-K-N	22.79$\pm$3.3	20.61$\pm$3.0	**19.44**$\pm$3.2		L-K-R	23.61$\pm$3.3	20.67$\pm$3.3	**18.38**$\pm$2.7	
L-R-N	19.09$\pm$3.1	18.47$\pm$2.6	**18.03**$\pm$2.8		L-R-K	19.67$\pm$2.9	**19.60**$\pm$3.0	20.50$\pm$2.8	
N-L-K	19.28$\pm$3.0	**18.24**$\pm$2.5	20.66$\pm$3.1		N-L-R	19.08$\pm$2.7	18.25$\pm$2.5	**17.97**$\pm$2.7	
N-K-L	19.62$\pm$3.2	**18.76**$\pm$2.7	18.89$\pm$2.5		N-K-R	19.93$\pm$2.8	18.98$\pm$2.6	**17.64**$\pm$2.5	
N-R-L	19.36$\pm$3.3	18.66$\pm$2.9	**18.45**$\pm$2.6		N-R-K	19.89$\pm$3.2	**19.60**$\pm$3.1	21.05$\pm$2.8	
K-L-N	25.69$\pm$3.4	22.43$\pm$3.1	20.02$\pm$2.9	**18.83**$\pm$3.0	K-L-R	26.33$\pm$3.7	20.09$\pm$3.1	**19.94**$\pm$2.9	
K-N-L	25.65$\pm$3.0	21.07$\pm$3.0	**19.17**$\pm$2.9		K-N-R	28.17$\pm$3.1	22.56$\pm$3.1	**19.71**$\pm$2.8	
K-R-L	25.07$\pm$3.6	19.15$\pm$2.8	**18.57**$\pm$2.8	24.15$\pm$3.1	K-R-N	25.92$\pm$3.4	21.15$\pm$3.2	19.08$\pm$3.2	**18.52**$\pm$3.0
R-L-N	20.27$\pm$2.8	19.69$\pm$3.0	19.54$\pm$3.1	**18.96**$\pm$2.9	R-L-K	20.72$\pm$2.9	**19.63**$\pm$2.9	22.51$\pm$3.1	
R-N-L	21.06$\pm$2.7	19.85$\pm$2.6	**19.69**$\pm$2.9		R-N-K	22.77$\pm$3.0	**21.49**$\pm$3.1	23.12$\pm$3.2	
R-K-L	20.74$\pm$2.7	**19.98**$\pm$2.8	20.07$\pm$3.0		R-K-N	22.29$\pm$2.9	20.25$\pm$3.0	**19.38**$\pm$2.8	
L-N-K-R	23.02$\pm$2.8	20.14$\pm$3.0	**18.91**$\pm$2.7		L-N-R-K	21.09$\pm$2.8	**19.97**$\pm$3.2	20.48$\pm$2.6	
L-K-N-R	24.34$\pm$3.2	21.07$\pm$3.1	**19.37**$\pm$3.2		L-K-R-N	23.92$\pm$2.8	20.81$\pm$3.2	**19.05**$\pm$2.8	
L-R-N-K	19.91$\pm$2.7	**19.16**$\pm$2.8	20.32$\pm$3.0		L-R-K-N	19.93$\pm$3.0	**19.25**$\pm$3.1	19.27$\pm$3.0	
N-L-K-R	20.12$\pm$2.8	18.76$\pm$2.7	**17.85**$\pm$3.0		N-L-R-K	19.49$\pm$2.9	**18.57**$\pm$2.9	20.37$\pm$3.3	
N-K-L-R	20.63$\pm$2.6	18.43$\pm$2.6	**18.19**$\pm$2.8		N-K-R-L	20.91$\pm$2.9	18.64$\pm$2.5	**17.55**$\pm$2.5	
N-R-L-K	19.78$\pm$2.7	**18.71**$\pm$2.7	20.37$\pm$2.8		N-R-K-L	20.23$\pm$3.1	19.09$\pm$3.0	**18.46**$\pm$2.9	
K-L-N-R	26.65$\pm$3.6	20.33$\pm$3.2	**19.62**$\pm$3.4		K-L-R-N	26.79$\pm$3.4	20.44$\pm$3.0	**19.53**$\pm$3.0	
K-N-L-R	27.24$\pm$3.6	21.49$\pm$2.8	**20.24**$\pm$2.8		K-N-R-L	27.39$\pm$3.4	21.44$\pm$3.0	**19.34**$\pm$2.7	
K-R-L-N	24.70$\pm$3.3	19.30$\pm$2.9	19.19$\pm$3.0	**18.99**$\pm$2.6	K-R-N-L	25.62$\pm$3.3	19.20$\pm$2.5	**19.08**$\pm$2.6	
R-L-N-K	21.00$\pm$2.6	**19.75**$\pm$2.6	22.08$\pm$3.1		R-L-K-N	20.43$\pm$2.8	20.04$\pm$2.9	**19.91**$\pm$3.1	
R-N-L-K	21.42$\pm$3.1	**19.99**$\pm$3.3	21.44$\pm$3.3		R-N-K-L	20.88$\pm$3.1	19.72$\pm$3.2	**19.35**$\pm$2.9	
R-K-L-N	20.36$\pm$2.9	**19.49**$\pm$3.0	20.34$\pm$3.3		R-K-N-L	21.05$\pm$2.7	20.40$\pm$2.9	**19.65**$\pm$2.7	

Table A.12: Results for the VEHICLE dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	4.44±1.8	4.44±1.8	**4.43**±1.8	**4.43**±1.8	N	**4.21**±2.3	**4.21**±2.3	4.24±2.4	4.24±2.4
K	8.10±2.3	8.10±2.3	**8.07**±2.3	**8.07**±2.3	R	4.23±2.2	4.23±2.2	**4.22**±2.2	**4.22**±2.2
L-N	4.74±2.1	4.72±2.2	**4.02**±1.8	4.05±1.8	L-K	4.63±1.9	**4.61**±2.0	6.53±2.2	
L-R	4.68±1.8	4.65±1.8	**4.31**±1.9		N-L	**4.20**±2.0	4.21±2.1	4.32±2.1	
N-K	**4.15**±1.6	4.18±1.7	6.67±2.3		N-R	4.11±2.0	4.20±2.0	**3.72**±1.9	
K-L	5.40±2.3	5.01±2.4	**4.56**±2.0	4.71±1.9	K-N	5.37±2.1	4.76±2.0	4.52±2.1	**4.18**±2.1
K-R	5.20±2.2	4.63±2.2	**4.34**±2.2	4.34±2.1	R-L	4.86±2.1	4.84±1.9	4.98±2.8	**4.59**±1.8
R-N	4.64±2.1	4.91±2.2	**4.13**±2.0	4.17±1.7	R-K	4.39±2.0	**4.29**±1.9	7.08±2.2	
L-N-K	**4.71**±2.1	4.78±2.2	5.99±2.3		L-N-R	4.79±1.7	4.81±1.8	**3.99**±1.8	
L-K-N	4.80±1.9	4.75±2.0	**4.55**±1.8		L-K-R	4.51±2.2	4.48±2.2	**4.43**±2.2	
L-R-N	4.74±1.9	4.84±2.0	**3.90**±1.9		L-R-K	4.25±1.8	**4.21**±1.8	5.49±1.9	
N-L-K	4.30±1.9	**4.19**±2.0	5.18±2.3		N-L-R	4.19±1.8	4.31±1.9	**3.98**±2.0	
N-K-L	**4.14**±2.1	4.21±2.1	4.43±2.1		N-K-R	**3.97**±2.0	4.18±2.0	4.03±2.0	
N-R-L	**4.16**±1.8	4.31±1.9	4.47±2.1		N-R-K	**4.33**±1.8	4.66±1.9	5.31±2.1	
K-L-N	6.93±8.8	6.32±6.3	4.67±2.1	**4.54**±2.1	K-L-R	5.64±2.2	5.46±1.9	**4.46**±2.0	
K-N-L	5.93±2.5	4.93±3.3	**4.55**±1.8		K-N-R	5.34±2.4	5.22±2.3	**3.91**±2.1	
K-R-L	5.79±2.3	5.02±2.1	5.10±2.2	**4.17**±2.2	K-R-N	5.91±2.5	5.07±2.2	**4.46**±1.9	4.49±2.0
R-L-N	5.40±2.3	5.45±2.1	4.82±2.1	**4.61**±1.9	R-L-K	**4.46**±1.9	4.51±2.1	5.72±2.3	
R-N-L	4.80±2.3	4.42±2.1	**4.28**±2.1		R-N-K	**4.52**±2.0	4.92±1.9	5.80±2.1	
R-K-L	4.56±2.1	**4.44**±2.1	4.93±2.3		R-K-N	5.00±2.4	4.94±2.4	**4.70**±2.0	
L-N-K-R	4.91±2.2	4.98±2.1	**4.48**±2.0		L-N-R-K	**4.54**±2.0	4.76±2.2	5.34±2.3	
L-K-N-R	4.64±2.1	4.69±2.1	**4.44**±2.2		L-K-R-N	4.67±2.0	4.76±2.0	**4.58**±2.2	
L-R-N-K	4.67±1.8	**4.56**±1.8	5.20±2.2		L-R-K-N	5.17±2.5	5.18±2.5	**4.97**±2.3	
N-L-K-R	3.98±1.9	4.14±1.9	**3.82**±2.0		N-L-R-K	**4.43**±2.0	4.65±2.0	5.36±2.0	
N-K-L-R	3.99±1.8	4.24±2.0	**3.79**±1.7		N-K-R-L	**3.96**±2.0	4.16±2.0	4.39±1.9	
N-R-L-K	**4.39**±1.9	4.49±2.0	5.23±2.1		N-R-K-L	**3.93**±2.0	4.13±1.9	4.50±2.6	
K-L-N-R	5.75±2.4	5.10±2.5	**4.12**±1.9		K-L-R-N	6.08±2.1	6.75±7.7	**4.49**±2.1	
K-N-L-R	5.55±2.3	4.51±2.1	**3.90**±2.0		K-N-R-L	5.87±2.4	**4.94**±2.1	4.99±2.9	
K-R-L-N	5.91±2.4	5.43±5.3	4.53±2.1	**4.02**±1.9	K-R-N-L	5.40±2.4	**4.29**±2.7	4.30±2.6	
R-L-N-K	**4.93**±2.2	5.29±2.1	5.73±2.2		R-L-K-N	4.61±2.0	4.76±2.0	**4.59**±2.0	
R-N-L-K	**4.42**±2.2	4.55±2.1	5.53±2.4		R-N-K-L	**4.36**±2.0	4.48±2.0	4.46±2.1	
R-K-L-N	4.81±2.1	5.07±2.0	**4.60**±2.1		R-K-N-L	**4.51**±2.0	4.65±2.0	4.83±1.9	

Table A.13: Results for the VOTES dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Setup	Res	Res+C	Cas	Stack	Setup	Res	Res+C	Cas	Stack
L	**1.56**±2.0	**1.56**±2.0	**1.56**±2.0	**1.56**±2.0	N	**1.79**±2.2	**1.79**±2.2	**1.79**±2.2	**1.79**±2.2
K	**3.83**±2.9	**3.83**±2.9	**3.83**±2.9	**3.83**±2.9	R	**2.00**±2.1	**2.00**±2.1	**2.00**±2.1	**2.00**±2.1
L-N	**1.10**±1.5	1.28±1.6	1.53±1.7	1.55±1.7	L-K	2.22±2.6	**2.03**±2.4	3.09±2.8	
L-R	1.12±1.7	**1.05**±1.5	1.48±2.0		N-L	2.00±2.0	2.17±2.1	**1.96**±2.1	
N-K	**1.78**±1.9	1.80±2.0	2.19±2.3		N-R	2.46±2.3	2.44±2.2	**2.22**±2.1	
K-L	2.89±2.7	3.83±10.0	3.34±6.6	**1.43**±1.9	K-N	2.71±2.6	2.73±2.7	2.34±2.4	**1.93**±2.0
K-R	2.79±3.0	2.85±2.9	2.42±2.6	**1.81**±2.3	R-L	1.96±2.3	1.77±2.2	1.33±2.0	**1.25**±1.8
R-N	2.06±2.5	1.91±2.5	1.79±2.2	**1.67**±1.9	R-K	**1.96**±2.1	2.26±2.4	2.41±2.6	
L-N-K	1.76±2.2	**1.71**±2.2	1.91±2.2		L-N-R	1.26±1.8	**1.19**±1.9	1.64±2.1	
L-K-N	2.57±2.6	**2.33**±2.6	2.44±2.5		L-K-R	2.56±2.6	2.31±2.5	**1.97**±2.2	
L-R-N	1.69±2.1	**1.41**±1.8	1.93±1.9		L-R-K	1.94±2.2	**1.67**±2.0	2.36±2.6	
N-L-K	2.18±2.4	2.32±2.3	**2.14**±2.5		N-L-R	2.00±2.1	1.93±1.9	**1.92**±1.8	
N-K-L	2.65±2.5	2.44±2.3	**2.32**±2.5		N-K-R	2.07±2.3	2.11±2.3	**1.75**±2.0	
N-R-L	2.59±2.7	2.59±2.5	**2.41**±2.4		N-R-K	1.91±2.3	**1.90**±2.2	2.30±2.4	
K-L-N	2.81±2.9	2.64±2.8	2.33±2.6	**1.95**±2.0	K-L-R	3.23±2.9	3.16±2.9	**2.42**±2.9	
K-N-L	2.76±2.9	2.83±2.9	**2.34**±2.3		K-N-R	3.80±3.5	3.52±3.0	**2.29**±2.5	
K-R-L	3.10±2.8	3.17±2.9	2.89±2.9	**1.33**±1.8	K-R-N	2.91±2.5	2.75±2.4	2.53±2.4	**2.02**±1.9
R-L-N	1.52±2.0	**1.48**±1.7	1.70±2.1	2.07±2.1	R-L-K	**2.08**±2.2	2.57±2.2	2.38±2.3	
R-N-L	**1.56**±1.9	1.64±2.1	1.70±2.1		R-N-K	**1.52**±2.1	1.86±2.1	1.67±2.0	
R-K-L	**2.13**±2.6	2.25±2.4	2.29±3.1		R-K-N	2.09±2.3	2.10±2.3	**1.91**±2.1	
L-N-K-R	2.80±2.5	2.27±2.2	**2.13**±2.1		L-N-R-K	1.94±2.1	**1.57**±2.0	1.94±2.2	
L-K-N-R	2.50±2.8	**1.87**±2.0	2.05±2.2		L-K-R-N	2.33±2.7	**2.05**±2.4	2.06±2.2	
L-R-N-K	1.61±2.0	**1.57**±1.9	1.73±2.0		L-R-K-N	1.71±2.1	**1.56**±2.0	1.94±2.1	
N-L-K-R	2.15±2.4	2.10±2.5	**2.05**±2.3		N-L-R-K	2.18±2.4	**2.02**±2.2	2.06±2.2	
N-K-L-R	2.10±2.5	2.29±2.3	**1.80**±2.5		N-K-R-L	**2.53**±2.7	2.62±2.9	2.74±2.5	
N-R-L-K	2.33±2.5	2.32±2.4	**1.99**±2.3		N-R-K-L	2.25±2.4	2.32±2.5	**2.04**±2.3	
K-L-N-R	3.73±3.0	4.48±10.1	**2.90**±2.5		K-L-R-N	3.41±2.9	3.48±2.9	**2.66**±2.2	
K-N-L-R	3.36±2.5	3.22±2.8	**2.54**±2.5		K-N-R-L	3.39±3.0	3.33±3.0	**2.80**±2.9	
K-R-L-N	2.86±2.9	3.05±2.8	2.30±2.3	**1.13**±1.6	K-R-N-L	3.58±3.2	3.53±3.0	**2.80**±2.5	
R-L-N-K	2.29±2.7	2.06±2.8	**1.94**±2.4		R-L-K-N	2.36±2.8	2.32±2.7	**2.19**±2.6	
R-N-L-K	2.29±2.3	2.30±2.7	**1.92**±2.6		R-N-K-L	2.42±2.6	2.47±2.8	**1.81**±2.4	
R-K-L-N	2.18±2.5	2.27±2.5	**2.12**±2.5		R-K-N-L	**2.27**±2.4	2.39±2.3	2.30±2.7	

Table A.14: Results for the WINE dataset, errors are reported in mean classification error over 100 test splits. L=linear regression, N=neural network, K=k-nearest neighbors, R=kernel ridge regression. Red values are per-column ensemble winners. Large red and underlined is the minimum mean value.

Bibliography

[1] *In The Handbook of Brain Theory and Neural Networks, Second edition*, chapter Ensemble learning. Cambridge, MA, 2002. (Cited on page 1.)

[2] K. M. Ali and M. J. Pazzani. Error reduction through learning multiple descriptions. In *Machine Learning, Springer*. (Cited on page 1.)

[3] A. Asuncion and D. Newman". "UCI machine learning repository", "2007". `"http://archive.ics.uci.edu/ml/datasets.html"`. (Cited on pages 35, 44 and 71.)

[4] J. Bennet and S. Lanning. The netflix prize. KDD Cup workshop, 2007. `"http://www.netflixprize.com"`. (Cited on pages 2, 6, 9, 50, 51 and 55.)

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited on pages 5, 16 and 45.)

[6] L. Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996. (Cited on pages 2, 23, 58, 65 and 73.)

[7] L. Breiman. Stacked regressions. *Mach. Learn.*, 24(1):49–64, 1996. (Cited on page 30.)

[8] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. (Cited on pages 17 and 61.)

[9] R. P. Brent. Algorithms for minimization without derivatives, 1973. (Cited on page 19.)

[10] G. Brown. *Diversity in Neural Network Ensembles*. PhD thesis, University of Birmingham, 2004. (Cited on pages 3 and 4.)

[11] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *In Proceedings of the 21st International Conference on Machine Learning*, pages 137–144. ACM Press, 2004. (Cited on pages 2, 4, 30, 44 and 75.)

[12] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1995. (Cited on page 24.)

[13] Y. Freund and R. E. Schapire. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999. (Cited on pages 3 and 24.)

[14] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6:161, 1991. (Cited on page 44.)

[15] J. Friedman. Greedy function approximation: A gradient boosting machine. Technical report, Salford Systems, 1999. (Cited on pages 17 and 58.)

[16] J. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 2002. (Cited on pages 17 and 58.)

[17] J. Gama and P. Bradzil. Cascade generalization, 2000. (Cited on pages 27, 35, 40, 44 and 73.)

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. (Cited on pages 5, 6, 33 and 45.)

[19] L. Herlocker, Jon, A. Konstan, A. Joseph, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002. (Cited on page 50.)

[20] L. Herlocker, A. Konstan, G. L. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004. (Cited on page 50.)

[21] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006. (Cited on page 14.)

[22] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining ICDM*, 2008. (Cited on page 51.)

[23] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. In *Neural Computation*, pages 79–87, 1991. (Cited on page 3.)

[24] Y. Koren. The BellKor solution to the Netflix Grand Prize, 2009. (Cited on pages 2 and 51.)

[25] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9:112–147, 1998. (Cited on page 19.)

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. (Cited on pages 13 and 45.)

[27] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and K. Muller, editors, *Neural Networks: Tricks of the trade*. Springer, 1998. (Cited on pages 13 and 14.)

[28] P. Melville and R. J. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 505–510. (Cited on page 4.)

[29] M. Piotte and M. Chabbert. The Pragmatic theory solution to the Netflix Grand Prize, 2009. (Cited on pages 2 and 51.)

[30] J. R. Quinlan. Simplifying decision trees. *Int. J. Hum.-Comput. Stud*, 51:497, 1999. (Cited on page 35.)

[31] G. Raetsch. Robust multi-class boosting, 1993. (Cited on page 25.)

[32] R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007. (Cited on page 54.)

[33] R. E. Schapire. The strength of weak learnability, 1990. (Cited on page 24.)

[34] H. Schwenk and Y. Bengio. Boosting neural networks. *Neural Comput.*, 12(8):1869–1887, 2000. (Cited on pages 3, 25, 44, 73 and 74.)

[35] R. Siegler. Dataset balance, generated to model psychological experiments, three aspects of cognitive development. In *Cognitive Psychology*. (Cited on page 73.)

[36] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Tech. Dig*, vol. 10:262–266, 1989. in. (Cited on page 37.)

[37] J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-weighted linear stacking. *arXiv:0911.0460v2*, 2009. (Cited on page 65.)

[38] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Johns Hopkins APL Technical Digest*, 10:262–266, 1988. (Cited on page 36.)

[39] A. N. Tikhonov and V. Y. Arsenin. Solution of ill-posed problems, 1977. (Cited on page 13.)

[40] K. M. Ting and I. H. Witten. Stacking bagged and dagged models. In *In Proc. 14th International Conference on Machine Learning*, pages 367–375. Morgan Kaufmann, 1997. (Cited on pages 2, 23 and 44.)

[41] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999. (Cited on pages 23, 26 and 35.)

[42] A. Töscher, M. Jahrer, and R. M. Bell. The BigChaos solution to the Netflix Grand Prize, 2009. (Cited on pages 2, 15, 30, 51, 53, 54, 55 and 58.)

[43] A. Töscher, M. Jahrer, and R. Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *KDD Workshop at SIGKDD 08*, August 2008. (Cited on page 53.)

[44] J. Weston, A. Elisseeff, G. Bakir, and F. Sinz. The spider machine learning toolbox for matlab. Web, 2006. `http://www.kyb.tuebingen.mpg.de/bs/people/spider`. (Cited on page 33.)

[45] W. Wolberg and O. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology,. In *Proceedings of the National Academy of Sciences*, pages 9193–9196, Dec 1990. (Cited on page 36.)

[46] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992. (Cited on page 26.)