Toward Successful Participation in Machine Learning Contests

Advanced Machine Learning – Project Report

Marcel Ackermann
TU Darmstadt
www.marcel-ackermann.com

Christoph Dann
TU Darmstadt
cdann@cdann.de

Tobias Plötz
TU Darmstadt
tobias.ploetz@gmail.com

All authors contributed equally.

***Abstract*—This paper is a report on our participation in the *Bond Trade Price Contest* and *Online Product Sales Contest* with two main contributions. First, we provide a detailed description of our approaches in both contests, which includes our expert model based on support vector machines in the Bond Trade Price Contest and our ensemble of gradient boosted trees and Gaussian processes in the Online Product Sales Contest. We also review briefly the approaches of other teams. Second, this paper presents general strategies that we found helpful in Machine Learning contests. These strategies are embedded in a holistic view on competing in such contests, which provides a valuable set of tools for successful participation for readers with basic knowledge about Machine Learning.**

I. Introduction

Each student has access to lectures on machine learning, in particular to the huge number of online videos and courses[1]. However, theoretical in-depth knowledge of concepts obtainable from such sources has to be complemented with practical hands-on experience in implementing and tuning algorithms for machine learning. Only then are major advances in research or successful applications possible. By participating in competitions, students can obtain experience in the engineering part of machine learning in a very short time. Online contests in particular, such as the ones hosted on the Kaggle website, have drawn a lot of attention lately, as they allow everyone to directly compete against world-class machine learning experts from all over the world. Hence, every student interested in related topics is advised to participate in such contests and enjoy the motivation of neck-and-neck races. With enough effort – and a bit of luck – a large amount of prize money, followed by world-wide fame and interesting job offers, comes for free as icing on the cake.

This paper is a report on the participation in the *Bond Trade Price Contest* and the *Online Product Sales Contest* during the Advanced Machine Learning Project at TU Darmstadt, Germany. We provide in-depth analyses of our submissions as well as a comparison to strategies of other top-scoring teams. Moreover, this paper should serve as a guide for contest participation in general. Readers should have basic knowledge of machine learning techniques, e.g., from an introductory lecture, but are provided with an overview of tools, good practice and other information required for successful participation. The variety of these topics only allows a summary of the main facts and ideas. However, pointers and references to further information are included.

The remainder of this paper is structured as follows. First, we present general advice for machine learning contests in Section II, including an introduction to contest settings and useful software packages. Section III covers the Bond Trade Price contest. We introduce the task, highlight major challenges and present a quantitative analysis of our approach as well as a short description of other models. Afterwards, Section IV follows the same pattern but for the Online Product Sales contest. The paper is concluded in Section V with a short discussion on the background of contestants.

II. General Contest Advices

In the following we discuss helpful advice for machine learning contests in general. While we focus on competitions hosted on the Kaggle website[2], the methodologies presented here are applicable to other contests as well. First, the setting of contests with online submission systems is presented briefly, followed by a discussion of standard procedures, which have proven themselves

[1]Popular websites with online lectures are for example Videolectures.net, Coursera or MIT OpenCourseware.

[2]http://kaggle.com

handy for most contestants. We discuss in particular how to evaluate the quality of your method in Section II-C, preprocessing in Section II-D, prediction models in Section II-E and building an ensemble in Section II-F. Finally, a selection of prominent software packages is presented.

A. Contest Setting

Kaggle hosts several contests, in which everyone can compete for free against machine learning enthusiasts all over the world. The contests are organized by companies, non-profit organizations or universities. The prizes range from money to job offers or simply "fame" for research tasks. Contestants can participate as individuals or in teams and merge teams during the competition. The duration is typically three months, but can vary between a day and a few years. For example, Kaggle hosted the $3 million Heritage Health Prize[3] running for 2 years, the Facebook Recruiting Competition[4] or the Million Song Dataset Challenge[5].

The core component of each contest is the data sets. The provided data is split into training and test set, which have the same attributes (often numerical or categorical, but text and structured data such as graphs are also possible). The prediction output, called label, is additionally provided for the training data. The goal is to generate predictions for the test set, which is usually a classification or regression problem. There are no restrictions on the methodology for obtaining a solution, even labeling manually is sometimes possible.

The predictions for the test data can be uploaded on a submission webpage, but only a limited number of times per day until the final deadline. The test set is split into a private and public part. The submission's score on the public part is immediately calculated and published on the public leaderboard, while the score on the private part determines the winner and is disclosed on the private leaderboard after the deadline. Both scores are calculated according to a publicly known evaluation measure. This splitting approach penalizes leaderboard overfitting, a common mistake avoidable by sound methodology (see Section II-C).

B. General Strategy

The process of generating predictions for the test set commonly consists of three main steps. First, the feature representation is transformed and data may be subsampled such that relevant patterns are emphasized and noise is reduced. In the second step, several independent machine learning algorithms generate proposal predictions for each test sample. The final prediction is then generated by merging the single proposals with an ensemble model. See Figure 1 for a visualization. Deviations from this scheme are possible, e.g., only one prediction model is used without an ensemble technique or prediction pipelines with more steps. However, the main steps presented here are part of most approaches. Participating in a contest actually means improving each component of this pipeline in order to obtain better prediction performance.

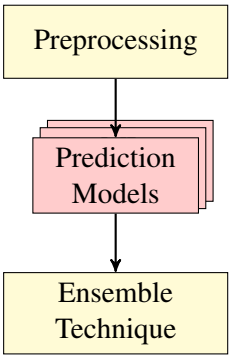


Fig. 1: Common prediction pipeline in contests: After preparing your data in a preprocessing step, several machine learning models generate independent predictions. Then the predictions are merged with an ensemble technique.

The key element for success is to identify the parts with room for improvement during the contest. It is important to understand the task and data well, i.e., inspecting the dataset is crucial. In general, the prediction problem should be decomposed into sub-problems. A common tool is for example a bias-variance analysis[6]. It is also reasonable to improve each component of the prediction pipeline individually. We provide a collection of helpful strategies for evaluation, preprocessing, employing prediction models and finally blending them in the following four sections.

C. Model Evaluation

The final goal is to minimize the prediction error on the private test set. As the private leaderboard is not available during the contest the actual performance can only be estimated. One might be tempted to use the public leaderboard score for evaluation. However, minimizing this score may cause overfitting on the public test set and yield inferior performance on the private test set. To circumvent this issue, we highly recommend

[3]http://www.heritagehealthprize.com/c/hhp

[4]http://www.kaggle.com/c/FacebookRecruiting

[5]http://www.kaggle.com/c/msdchallenge

[6]http://cs229.stanford.edu/materials/ML-advice.pdf

to use cross-validation [Mur12, Chapter 6.5.3] on the training set and occasional submissions to check that the public leaderboard score and the cross-validation score are consistent. In addition, the neck-and-neck race on the leaderboard is a great booster for motivation. If the computation of cross-validation scores on a large training set takes too much time, it is reasonable to use only one split, e.g., splitting the training set in an 80% split to train the models and a 20% validation split. If it still takes more than a few hours to evaluate the models, we recommend using only part of the training set. However, special care is necessary to ensure that the sample is representative by comparing statistics (e.g. the first and second moments of the target distribution).

Models tend to become more and more complicated during the contests, hence, significance tests are highly recommended (c.f. [Was07, Chapter 10]). For example, a one-sided paired T-test for cross validation scores can be used to decide whether a new model actually predicts more accurately or not. This methodology prevents the models from getting unnecessarily complex.

In general, the entire evaluation pipeline (preprocessing the data, training the models, training the ensemble, computing cross-validation scores) should be automated as much as possible to promote reproducibility and fast exploration of ideas for improving the prediction.

D. Preprocessing

Preprocessing is supposed to emphasize relevant patterns and minimize noise in the data. To this end, the feature representation of each sample is transformed (e.g. by adding expressive new features generated from the existing ones). It may also be beneficial to remove irrelevant features or omit outlying samples during training. As the feature representation is highly task dependent, domain knowledge is most helpful for identifying relevant features. A crucial point of the Online Product Sales Competition was for example to realize that the number of sales of a product is influenced by the day of the year it is launched (see Section IV-C). An important tool for recognizing patterns is to visualize the data including statistics such as the number of different values of a feature or its variance. Correlation plots of single features and targets can also point to principal dependencies. The graphical interface of Weka (see Section II-G) provides particularly easy access to standard visualizations.

In the end, preprocessing should help the prediction model to find patterns most efficiently. Due to the different nature of machine learning algorithms the optimal feature transformation depends on the subsequent algorithm. For example, kernel- or distance-based algorithms such as Support Vector Machines or k-Nearest Neighbor are most suited for features in continuous domain and comparable quality. The performance of such algorithms can be increased by normalizing the features such that each dimension has mean zero and variance one [VTS04]. As categorical features lack a natural order, either a specialized kernel has to be used or the features need to be encoded differently. A categorical feature with k different values can for example be replaced by k binary indicator features, where the i^{th} indicator feature is active if the original feature takes the i^{th} value (one-hot encoding, see [Mur12, Chapter 2.3.2]). Moreover, distance based algorithms are prone to irrelevant features as they diminish the quality of the distance measure[7]. In contrast, decision-tree-based approaches such as Random Forests can cope with large numbers of bad features well. They do not require feature normalization either, as linear transformations of features have no effect in decision trees. However, while kernel approaches usually can cope with missing values well[8], special treatment is necessary for decision tree approaches. Replacing missing values with mean or median is a common way. The feature representation can additionally be augmented by binary features which indicate whether a value was missing.

E. Prediction Models

Prediction models (or their implementation as prediction algorithms) are the key component of prediction pipeline. The choice of a suitable model for the task at hand as well as tuning its hyper-parameters is most crucial for the prediction quality. Whether an algorithm is suited for a particular problem or not depends on several properties such as the evaluation measure, feature representation, the training set size or the task type (structured prediction, classification, regression and ranking). In the following we provide a short list of algorithms successfully applied in contests. We limit ourselves to regression and classification algorithms for clarity. Other tasks such as ranking problems or structured output problems are often reduced to regression or classification and solved with the same algorithms. As this paper does not aim to explain Machine Learning methods in general, we only give short remarks indicating the most important properties of each algorithm in the context of competitions.

[7] for common kernels such as the Gaussian or exponential kernels. Alternatively, more sophisticated distance measures such as automated relevance determination forms can be used and their hyper-parameters be learned from data.

[8] Dimensions with missing values can be omitted during computation of the distance for example.

Algorithm	Task	Class	Data size
Linear Regression	regression		large
Logistic Regression	classification		large
Gaussian Processes	regression	distance	small
Support Vector Machines	both	distance	medium
Gradient Boosted Trees	both	tree	large
Random Forests	both	tree	large
Neural Nets	both		large
K-Nearest Neighbors	both	distance	medium
Naive-Bayes Classifier	classification		large

TABLE I: Overview of Machine Learning algorithms applicable to classification and regression problems. Whether an algorithm is distance-based or tree-based influences its suitability for specific tasks and the necessary preprocessing (c.f. Section II-D). The scalability information should be taken with a pinch of salt (highly dependent on the actual implementation) but can help to decide which method to choose for a problem at hand.

Linear Regression [Mur12, Chapter 1.4.5] is a simple model for regression which minimizes the mean-squared error on the training dataset. As they are prone to overfitting, the value of the hyper-parameter for the ℓ_2-regularization is crucial. The method is particularly suited for large datasets with many features, where a linear model is expressive enough.

Logistic Regression [Mur12, Chapter 1.4.6] extends linear regression by squashing the outputs through a sigmoid-function. The result is a *classification* model with probabilities for each class as output. It shares most properties with linear regression such as high scalability but limited expressiveness in low-dimensional data.

The **Naive Bayes Classifier** [Mur12, Chapter 3.5] is a probabilistic classification model similar to logistic regression. Because of its simplicity and scalability it is often used in practice. As shown by Jordan and Ng [JN02], it needs less data to converge than logistic regression, but usually yields worse predictions.

While Logistic Regression and Naive Bayes Classifiers have high bias, the **k-Nearest-Neighbor method** [Mur12, Chapter 1.4.2] is a non-parametric approach and predicts perfectly for infinitely many training data. A test point is predicted by assigning it to the predominant label of the k nearest training points. Therefore, the quality of the prediction largely depends on the distance measure. Sophisticated data structures such as kd-trees reduce the runtime for finding the neighbors [GCB97], but the algorithm is still limited to medium sized datasets by its computational effort and the requirement to store the entire training data. While the method is commonly used for classification, it is also applicable for regression[9].

Gaussian Processes [RW06] are powerful probabilistic models for regression tasks minimizing the mean squared error. They are very data efficient, non-parametric and their regularization can be accurately controlled by defining prior distributions. Hence, Gaussian Processes are one of the first choices for regression on continuous datasets. Unfortunately, training involves the inversion of a samples$\times$samples matrix, which limits the algorithm to small datasets. As the model relies on (infinite-dimensional) Gaussian distributions, the likelihood of its hyper-parameters such as the width of the Gaussian kernel can be computed analytically. Thus, hyper-parameters can be tuned with local optimization algorithms (e.g. L-BFGS [LN89]) much more efficiently than with standard exhaustive grid search.

Support Vector Machines are kernel-based methods just like Gaussian Processes. Yet, they are not derived with Bayesian statistics but with frequentist statistics. As they minimize the Hinge loss function, Support Vector Machines are powerful classification [CV95] and regression [SS04] algorithms particularly robust against outliers. They can be trained efficiently with stochastic sub-gradient descent even on large datasets for linear kernels. Using non-linear kernels such as the standard Gaussian kernel restricts training to medium sized datasets (c.f. Section III).

Neural Networks [Mur12, Chapter 16.5] are flexible, biologically inspired models for regression and classification, which can be visualized as a network of neurons. Since the number of nodes (neurons) as well as their overall structure can vary tremendously, neural networks can have large bias or large variance. The high flexibility of neural networks make them hard to use for novices. However, if designed properly, the models beat state-of-the art methods especially in tasks with low-level features such as pixels in images [CM12]. Neural networks have been a focus of research recently. These efforts yielded advanced training schemes such as drop-out training [HS12] and deep belief networks (extended neural networks trained with combination of supervised and unsupervised learning). Their capabilities, including high scalability, were demonstrated by the winners of the Merck Molecular Activity Challenge [Dah12].

Gradient Boosted Decision Trees (GBT) [Fri01, Fri02] are iteratively boosted decision trees. At each iteration a new tree is learned to reduce the training errors of the previous trees. As most boosting approaches, GBTs are very robust against overfitting (see Section IV-D) and irrelevant features. They have proven themselves as one of the most important models in competitions with regression tasks (c.f. Sections III-E and IV-H), but can also be employed for classification. While the trees can only be learned sequentially, training on large datasets is still possible when the training data is sub-sampled for

[9] assign the average of the values of the k nearest neighbors

each tree. Considering only a random subset of instances usually does not hurt the final accuracy.

Random Forests [Bre01] consist of several decision trees, too. In contrast to gradient boosted trees, each tree is trained independently on a subset of features and instances, which allows parallelizing learning of extremely large datasets (e.g. on computing clusters).

Hyper-Parameter Search. Setting good hyper-parameters for the prediction algorithm is as crucial as the algorithm choice itself, in particular for flexible models such as neural networks or gradient boosted decision trees. The performance of gradient boosted trees depends for example on the number of iterations, the learning rate, the tree size, the minimum number of training instances per leaf or the splitting criterion.

Finding optimal hyper-parameters can be considered as a non-linear (and in general non-convex) optimization problem. The function value to minimize is the cross-validation or held-out error which depends on the hyper-parameter values. Facilitating hyper-parameter search is a recent focus of research [HHLBM10, HHLB11, BBBK11, BB12]. The most naive way to find good hyper-parameters is to define a reasonable range of values for each parameter and compute the error for each possible combination of parameter values. Although evaluating the error at each point of a grid spanned in the parameter space (referred to as *exhaustive grid-search*) is computationally involved, it is still the most used approach due to its simplicity. We also employed grid-search in both contests we participated in. However, Bergstra and Bengio [BB12] showed that sampling hyper-parameters uniformly within a specified range is more efficient when some parameters are more important than others, which is often the case. As such a *random search* is just as easy to implement and parallelize as grid-search, we recommend its use in contests. In situations where the error or score of hyper-parameters (and their derivatives) can be computed analytically, e.g. their negative log-likelihood in Gaussian Processes, approximate Newton methods such as L-BFGS [LN89] can be applied much more efficiently than exhaustive search strategies. For score values that are not analytically tractable, gradient descent with finite-difference approximation can refine the parameters found by grid- or random search. Each strategy relies on the parameter ranges to be chosen reasonably. Unfortunately, the ranges vary substantially between different tasks in most cases. We recommend first exploring the effect of hyper-parameters manually by evaluating hand-set parameter values and estimate a suitable range. If necessary, the range should be adapted by iterating between parameter search and refining the ranges to search. Alternatively, a model of the score function can be learned with few evaluations (e.g. a Gaussian Process) which is then iteratively refined with new evaluations and minimized by a global optimization algorithm. This technique, referred to as *Bayesian Optimization* [BBBK11], is computationally expensive but still faster than other approaches when the error of hyper-parameters is costly to evaluate. Probabilistically motivated algorithms allow fully Bayesian approaches, where the hyper-parameters are marginalized out instead of only using one. Such models are also referred to as hierarchical Bayesian models [GCSR03, Chapter 5]. While marginalizing out hyper-parameters often yields significantly better performance, it is only tractable for a very simple models. Otherwise, extensive sampling is necessary, which is not feasible in competitions.

F. Ensemble of Models

Most competitions are won by an ensemble of methods (e.g. the Netflix Prize [BKV08]), where outputs of several prediction models are blended together by a weighted average. Blending several predictors can for example be beneficial if errors are only mildly correlated and the individual errors counterbalance each other. While there are theoretical work on blending models in classification tasks, the effect of model averaging is not well investigated in regression tasks. Models do not necessarily need to be blended by a weighted average, but any algorithm can be trained with the output of the individual predictors as input (also referred to as Stacking [Mur12]). However, simple models such as linear regression (i.e. a weighted average) obtain best performance in practice. Complex ensemble algorithms tend to overfit, which can also be a problem of simple ensemble models with flawed training schemes.

We therefore recommend training the ensemble as follows. The training set is split into 5 or 10 folds. Each individual model is trained on all but one fold and predicts the outputs on the remaining fold. Repeating this scheme yields outputs of each model for all samples in the training set. Afterwards the training data is split again into a large portion on which the ensemble method is trained and a held-out set, which is used to evaluate the ensemble. The hyper-parameters of the ensemble method are chosen so that they minimize the error on the held-out set. Eventually, the ensemble model is trained with the optimal hyper-parameters on the entire training set[10]. As this technique involves retraining of the base prediction models, special care has to be taken if they have a high variance (e.g. due to a randomized learning procedure).

[10]Each individual model has to be trained again on the entire training set to predict on the test-set.

In this case, reducing the variance should be considered, e.g. by methods such as *Bagging* [Bre96a].

Although it is reasonable to optimize the preprocessing, the prediction models and the ensemble model individually, improving one part sometimes yields worse overall performance. We particularly observed this effect for the individual prediction models. For example, if an algorithm generates poor predictions for most samples but is able to tackle a small subset very well, it can improve the ensemble – even if it has a high overall error. Therefore, combining complementary models such as tree-based and distance-based algorithms should be considered in competitions.

G. Software Packages

Writing code for contests and research is quite different from software engineering in general. While robustness, maintainability, and security might be the most important properties of software in other domains, fast prototyping is most crucial in competitions. Minimal implementation effort for visualizing data, processing data and applying machine learning algorithms results in short iteration cycles, which allow exploring many ideas for improving the prediction performance. Several software tools proved themselves handy for producing fast and stable code during contests. We give an overview of the most prominent ones by highlighting their strengths and shortcomings in the following.

Weka [HFH+09] is a data mining framework written in Java with a very helpful graphical user interface for exploring and visualizing data. It has great capabilities when it comes to preprocessing and handling of meta data. For example, one can distinguish between numerical and ordinal attributes and track attributes along the preprocessing pipeline. However, the use case of Weka is very limited, as implementing machine learning algorithms in Java is cumbersome and only few bindings for other machine learning packages exist.

Matlab [MAT10] is a numerical computing environment. Its programming language features a rich syntax for matrix operations that makes implementing new algorithms easy. Hence, it is widely used in engineering and applied mathematics. However, as a special purpose language, Matlab is actually useful only for numerical data and slow in other tasks. We do not recommend it for building large modular machine learning pipelines, due to its lacking support of anything other than numerical operations. Moreover, Matlab is a commercial product with large costs while all other tools presented here are available for free.

R [R C12] is the favored tool for many statisticians. It provides a lot of functionality for manipulating numerical and categorical data and many machine learning algorithms are made available in separate packages. However, understanding the syntax requires investing a substantial effort.

In the last couple of years, there has been an emerging trend of using **Python for scientific computing**. A large set of libraries including the interactive shell **IPython** [PG07], **numpy** for numerical data, **matplotlib** [Hun07] for visualization and **scikit-learn** [PVG+11] for machine learning has been developed. The combination of a high-level general-purpose language and efficient libraries written in C provide the tools for automating the entire prediction pipeline in machine learning contests. However, Python only supports limited syntax for matrix operations and does not use multiple threads out of the box. The scikit-learn library provides a vast number of algorithms and has a clean interface, allowing the rapid integration of custom code into the framework. Moreover, other prominent machine learning libraries such as Shogun [SRH+10] have Python bindings. Python enables fast scripting and prototyping, but it may be slow on computationally hard problems. Scikit-learn is a very good choice for numerical data.

Clearly, every package has its strengths and weaknesses. To pick the cherries out of each one, wrapper libraries can be employed, which allow us to call R code within Python for example. For a beginner, we recommend Python as it is easy to learn and scikit-learn features many commonly used machine learning algorithms.

III. Bond Trade Price Contest

The *Benchmark Bond Trade Price Challenge* (*Bond Contest* for short) was our first contest. It was launched on January 27^{th} 2012 and ended three months later on April 30^{th} 2012. The task was to predict the trade price of a bond given general characteristics and information about the last ten trades of this bond. A bond is a certificate that a borrower, usually a state or company, has received money from a lender and that the borrower has to pay the money back at the date of maturity [OS05]. In the meantime, additional interests called coupons are paid by the borrower in regular intervals. Bonds may be traded at financial markets, similar to stocks. The pricing of a bond at the market reflects various factors such as the credibility of the borrower or the time until the bond matures, e.g., the shorter the time to maturity the less money can be earned from the coupons.

The large size of the training set with approximately 750,000 samples poses the main challenge of the contest and requires always taking the computational complexity

of different algorithms into consideration. In addition, modelling the last trades as time series is important for exploiting the given information most efficiently.

Our main goal for the Bond contest was to familiarize ourselves with applying machine learning techniques on real word problems, identifying general strategies and developing an efficient workflow. Eventually, we achieved the 33^{rd} place out of 265 competing teams.

In the following, we first introduce the main facts about the setting, then present different reformulations of the problems we used in the contest and afterwards discuss our preprocessing and prediction model. Finally, we briefly review the strategies of other teams.

A. Dataset and Evaluation

The dataset consists of roughly 750,000 samples with 60 features, which can be divided into information about the bond and information about the current and the last ten trades. They are listed in table II. The task was to predict the value of the trade price column p_0 given the values of the other columns such that the *Weighted Mean Absolute Error (WMAE)* is minimal. The WMAE is defined as follows. Given a set of true values $y_1, ..., y_n$, a set of predicted values $y'_1, ..., y'_n$ and weights $w_1, ..., w_n$, the weighted mean absolute error is calculated as

$$\text{WMAE} = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i |y_i - y'_i|. \qquad (1)$$

The training set contains an additional feature *bond_id* that indicates which particular bond is traded. This feature allows to reconstruct the full time series for each bond by combining all samples from the same bond. However, the feature *bond_id* is not present in the test dataset. Additionally, the competition host ensured, that there is no overlap in the samples of the test set, i.e., that any two samples from the same bond are at least 11 trades apart. Otherwise, the i^{th} last trade price p_i could have matched the unknown trade price p_0 of another sample in the test set.

Since evaluating models on the whole dataset is time consuming, we took subsets of the training set with non-overlapping samples to speed up the modelling process. For training and evaluation we generated two pairs of subsets with 10,000 resp. 50,000 samples, which we used to compare our modeling approaches during the contest.

B. Alternative Problem Formulations

Instead of modelling the trade price directly, we predicted the difference of the trade price to the price of the last trade. We give two arguments of why this might be a good idea. First, we believe that the last trade price greatly narrows the range in which the actual trade price may lie, so encoding this knowledge explicitly by only predicting the price differences saves the learning algorithm from modelling it. Second, we assume that most of the short term price dynamics is independent of the actual price level. For example, if the last trade was due to a customer selling a bond to a dealer and the current trade is the dealer selling the bond to another customer the price of the last trade will be significantly lower than for the current trade. These dynamics can be better expressed with price differences than with the actual price level.

We tried two ways to express price differences: *Absolute differences* Δp_A that are the price of the last trade minus the price of the current trade, i.e.

$$\Delta p_A = p_0 - p_1$$

and *relative differences* Δp_R that are absolute differences normalized by the price of the last trade, i.e.

$$\Delta p_R = (p_0 - p_1)/p_1$$

where p_0 is the current trade price and p_i is the trade price of the i^{th} most recent trade. The transformation to absolute differences leaves the error of an estimator invariant, i.e. $p_a - p'_a = p_0 - p'_0$, whereas this is not the case for the relative differences, i.e. $p_r - p'_r \neq p_0 - p'_0$. Hence, an estimator that minimizes some loss function on the relative differences will not necessarily be the estimator that minimizes the same loss function on the untransformed prices. Note, that a linear model will be invariant to the transformation to absolute differences.

	Original	**Abs. Diff**	**Rel. Diff**
Linear	0.932 ± 0.165	$\mathbf{0.932 \pm 0.165}$	0.933 ± 0.166
Tree	1.160 ± 0.172	0.917 ± 0.150	$\mathbf{0.916 \pm 0.151}$

TABLE III: Cross Validation WMAE and standard deviations using the original and the two derived learning problems for a linear model and a tree model.

In table III the WMAE of a linear model (Linear Regression without regularization) and a tree model (Gradient Boosting Trees) is evaluated using each of the three ways to formulate the learning problem. The results agree with our hypotheses. For the linear model, predicting the absolute difference does not change the WMAE and predicting relative differences is slightly inferior. For the tree model, however, reformulating the learning problem improves the performance. Both, predicting absolute and relative differences, resulted in significantly lower errors.

feature name	domain	description
id	$\mathbb{N}$	A unique identifier for the current sample
p_0	$\mathbb{R}$	The price for which the bond was traded. This variable is the label to predict.
w	$\mathbb{R}$	The weight given to this sample.
current_coupon	$\mathbb{R}$	The coupon on this bond (interest rate).
time_to_maturity	$\mathbb{R}$	The time until the bond is redeemed.
is_callable	$\{0, 1\}$	A flag indicating whether the bond can be called by the issuer.
reporting_delay	$\mathbb{R}$	The time between the trade actually happening and its apprearance in the reporting system.
p_i	$\mathbb{R}$	The price for the bond in this trade.
s_i	$\mathbb{N}$	The amount of bond certificates traded in this trade.
k_i	$\{2, 3, 4\}$	This feature indicates whether the trade represents a customer sell (2), a customer buy (3) or a trade between professional dealers.
e_i	$\mathbb{R}$	Long-term price estimate provided by the contest hosts.
t_i	$\mathbb{R}$	The time between the current trade and the last i^{th} trade taking place.

TABLE II: Features in the Bond Contest. The features having an i in their name are instantiated for $i = 0..10$ and represent the information given about the current trade and the last ten trades.

C. Preprocessing and Feature Extraction

In this section we describe which transformations we applied to the data and which new features we derived. To quantify the impact of each preprocessing step, we evaluate a linear regression model (called "Linear"), a GBT model on absolute price differences (called "Tree") and our final Treeclustering+SVR model on absolute price differences (called "TC+SVR") on the original feature set and on the processed feature set. Note, that we do not compare the performance of the different models, as their parameters are set in order to allow a fast evaluation.

a) Time Series Features: Each data vector contains information about the ten most recent trades before the current trade, i.e. a time series. Hence, it is natural to include information about the differences of consecutive trades in the feature set. To be precise, we included the time between two consecutive trades $t_{i+1} - t_i$ as well as difference in the trade price $p_{i+1} - p_i$ as features. We also took relative and absolute differences of the trade prices into account, analogous to the transformation of the target trade price p_0.

We hoped that the time difference allow a better modelling of why a trade happened. If two trades are issued at the same time, the later trade may be automatically issued in response to the first and hence price differences are intrinsic and can be modelled. If two trades are days apart, the later trade may be issued because of external factors such as bad news and there may therefore be no underlying deterministic model causing the price differences.

In table IV we show cross validation scores for different models in two settings: Once with the original features and once with the additional new time difference features. We could observe no improvements of the error scores. In two cases, adding the new features even hurt the performance, which leads to the conclusion that the time difference features alone have no additional predictive strength compared to the original feature set. Table V shows evaluation results for the price difference features. For the linear model, extending the feature set yields no performance gains, as the new features can be expressed as linear combinations of already existing features. For the tree model, however, adding the price difference features improves the error score.

	Original	**With Time Diff.**
Linear	0.932 ± 0.165	$\mathbf{0.932 \pm 0.166}$
Tree	$\mathbf{0.917 \pm 0.150}$	0.927 ± 0.150
TC+SVR	$\mathbf{1.015 \pm 0.152}$	1.027 ± 0.157

TABLE IV: Cross Validation WMAE and standard deviations comparing the original feature set to an extended set with time difference features.

	Original	**Abs. Diff**	**Rel. Diff**
Linear	0.932 ± 0.165	0.932 ± 0.165	$\mathbf{0.932 \pm 0.165}$
Tree	0.917 ± 0.150	$\mathbf{0.880 \pm 0.158}$	0.892 ± 0.156
TC+SVR	1.015 ± 0.152	$\mathbf{1.014 \pm 0.157}$	$\mathbf{1.014 \pm 0.157}$

TABLE V: Cross Validation WMAE showing the effect of the price difference features for different models.

b) Removing Features: At the end of the contest we experimented with removing features about the least recent trades from the feature set, as we believed that they do not carry useful information about the short term dynamics of the trade price and that the mid and long term dynamics are already incorporated in the estimated price features e_i. Table VI shows the cross-validation scores for excluding varying number of features. Removing the features has (in isolation) no impact on the linear model, while the tree model obtains its best score when features from the 7^{th} to 10^{th} most recent trades are removed. We also observe an improved error score for the tree clustered SVR model.

c) Logarithmic Transformations: The trade time and trade size related features show a highly skewed,

	Original	**10**	**9-10**	**8-10**	**7-10**	**6-10**	**5-10**
Linear	0.932 ± 0.165	**0.931 ± 0.166**	0.936 ± 0.167	0.936 ± 0.160	0.938 ± 0.159	0.938 ± 0.162	0.945 ± 0.162
Tree	0.917 ± 0.150	0.916 ± 0.149	0.917 ± 0.149	0.909 ± 0.150	**0.907 ± 0.145**	0.910 ± 0.152	0.907 ± 0.141
TC+SVR	1.015 ± 0.152	1.013 ± 0.152	1.008 ± 0.152	1.003 ± 0.153	1.000 ± 0.156	**1.000 ± 0.159**	1.001 ± 0.162

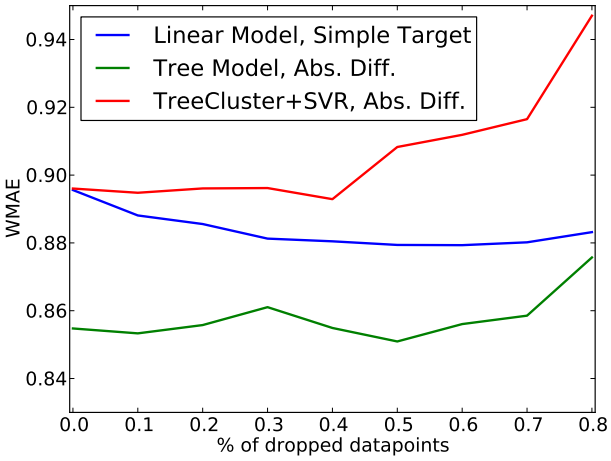
TABLE VI: Cross Validation WMAE for removing successively more features from the feature set. In the first column, only features from the 10^{th} most recent trade are removed, in the second column features from the 9^{th} and 10^{th} most recent trades are removed, etc.

scale free distribution. We applied logarithmic transformations[11] to these features and evaluated the effect of this transformation. We did not expect this transform to have a major impact on the performance of a tree based regression model as the logarithm is just a monotone transformation of the input feature. Hence, after the transformation the tree algorithm will split the input space between exactly the same two values of the training data, but the actual split point will differ. Yet, the transformations play a role for distance-based approaches such as our tree clustered SVR model, where we actually observed increasing performance from 1.015 to 1.003.

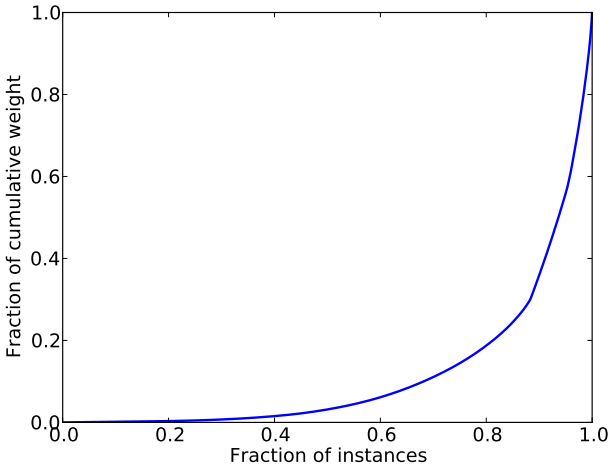
d) One-Hot Encoding of Trade Types: The trade types k_i are categorical features taking three values (see Table II). We added one-hot encoding features for each of them to make the their nominal nature explicit (c.f. Section II-D). This transformation leads to a substantial reduction of the error from 0.932 to 0.855 for the linear model and from 1.015 to 0.958 for the tree clustered SVR model. The tree model is unaffected by the newly introduced features, as they do not increase the expressiveness of the model (at least, if the tree depth is not limited).

e) Removing Trades With Low Weight: The weight w of each sample determines the influence of the sample on the overall error. All these weights exhibit a heavy tailed distribution, i.e., most training samples have a very small weight and few instances have a considerable high weight. See Figure 2b, where the sorted weights are summed up and plotted against the fraction of instances. Half of the instances account for only about 3% of the total weights. A regression algorithm, which assumed unweighted training data, alters its function prediction in favor of those instances, which are actually not relevant for the weighted error. This increased focus on instances with small weight may hurt the error on instances with high weight. Removing some of the training samples with low weight may hence improve the model. In contrast, removing these instances may hurt performance for a learning algorithm with a high variance. Figure 2a shows the performance of different models on multiple datasets with varying number of dropped instances. In addition, we used the final preprocessing pipeline. While the linear model benefits from removing instances, no such effect could be observed for the other models. Alternatively, we could have resampled based on the weight to balance the distribution or we could have incorporated the weights into the optimization problem of the SVR. We did not explore these options due to time constraints and the implementation effort, but we expect those to outperform dropping instances.

[11] As we are only dealing with integer values that may be zero, we actually apply the log1p function, i.e. $\log1p(x) = \log(x+1)$



(a)



(b)

Fig. 2: Figure a shows the performance of different models that are learned on datasets, where we drop an increasing percentage of the samples with the lowest weights. Figure b shows the fraction of samples with the lowest weights versus the fraction of total weight they account for. Standard errors are consistently around 0.03.

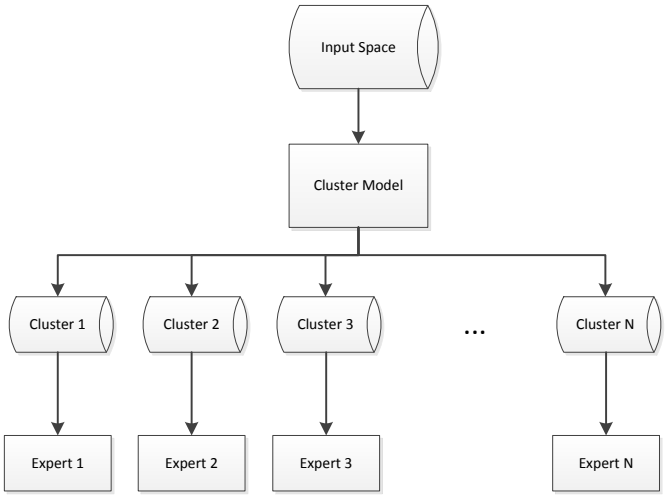


Fig. 3: The two layer expert model used in the Bond Contest. At the top layer a clustering model partitions the input space into disjoint subsets and hence the dataset into different clusters. A regression model is trained on each cluster to form an expert model for this part of the input space.

D. Prediction Model

When starting the contest, we decided to use Support Vector Regression (SVR) [SS04] as the base for our prediction model. At that time, we had no experience on prediction in heterogeneous datasets such as the Bond dataset and only limited knowledge about random forests and gradient boosted trees in general.

Training a support vector machine with non-linear kernels is not feasible on the Bond dataset with hundreds of thousands of samples. Hence, we decided to implement a two layer model: The first layer clusters the training set into batches of manageable sizes and the second layer consists of regression models (experts) fitted to one cluster each. At prediction time, new samples are first assigned to a cluster and then the expert of that cluster predicts the price for the sample. Figure 3 illustrates this approach. It is crucial that the clustering layer partitions the input space in such a way that the data in each region can be predicted more easily by the model class of the regression models. For example, we do not expect purely local methods such as a nearest neighbor regression model to benefit from this approach. In contrast, global models such as linear regression may actually improve.

There are many design choices in this model. Depending on the clustering algorithm, samples can be assigned hardly or softly to the clusters. For soft assignments, the predictions of the respective experts could be averaged depending on the assignment weights. Another option is to assign an expert to a cluster exclusively but to assign multiple clusters to each expert, again leading to an increased size of the expert's training sets. We did not explore these design choices but stuck to the simple model because of limited time until the end of the contest. We mainly considered two alternative models for the clustering layer: K-means and Regression Trees. Both models are described in the following paragraphs.

1) K-means: The k-means algorithm clusters a set of training samples into k clusters by minimizing the average squared distance to the assigned cluster centroid [Bis06]. We used this algorithm because an efficient implementation was already available for Weka. There are two important hyper-parameters: The number of clusters k and the distance measure imposed on the input space. Setting the number of clusters too low will produce clusters with many training samples, which renders learning the SVR model infeasible. If the training samples are partitioned into too many clusters with just a few training samples each, the SVR models would overfit and generalize poorly.

As k-means is an unsupervised clustering algorithm it may partition the data in a way that is not related to the target variable. During the contest, a distance measure that only takes the trade types and prices into account performed best. Table VII shows results for distance measures defined on different subsets of features. Calculating the distance only on the trade price yields the lowest error, but the tree clustering model presented below turns out to be superior.

2) Tree Based Clustering: Tree-based clustering uses a regression tree to partition the input space into clusters by assigning all samples of a leaf to a cluster. In contrast to k-means clustering, we can control the number of samples per cluster directly by specifying the minimum number of training samples per leaf. Moreover, the regression tree takes the label into account (supervised clustering), which yields clusters with low inter-cluster variance of the target variable (the tree is split such that the MSE of the leaf nodes is minimized). We found that learning a regression tree is much faster than learning a k-means clustering.

We evaluated the clustering model both with support vector and linear regression as experts and the entire preprocessing pipeline presented in the previous section. The results are shown in table VII. Tree clustering with SVRs outperforms the base line significantly, but, interestingly, using just linear regression as regressors works even better. Figure 4 shows the learned tree on a subset of the data[12]. The first splits occur on the trade types, as the price difference to the last trades

[12]The tree was learned with the regression tree implementation of WEKA. The evaluations in this paper are generated with the scikit-learn implementation.

Model	WMAE
Linear	0.842 ± 0.034
TreeCluster+SVR	0.799 ± 0.029
TreeCluster+Linear	$\mathbf{0.791 \pm 0.032}$
K-Means+SVR (all features)	0.844 ± 0.033
K-Means+SVR (trade prices)	0.837 ± 0.032
K-Means+SVR (trade types)	0.845 ± 0.031

TABLE VII: Cross Validation WMAE and standard deviations comparing a linear regression model to a tree clustering model with SVR and linear regression and to a k-means clustering model with SVR, where k-means used all features, only the trade prices or only the trade types for calculating distances. All preprocessing steps of our final pipeline were applied and prediction was done on the absolute trade price differences.

```
trade_type = 2
|   trade_type_last1 = 2
|   trade_type_last1 = 3
|   |   trade_size_last1 < 11.3
|   |   trade_size_last1 >= 11.3
|   trade_type_last1 = 4
trade_type = 3
|   trade_type_last1 = 2
|   |   trade_size < 11.18
|   |   trade_size >= 11.18
|   trade_type_last1 = 3
|   |   trade_price_delta1_1.0_last1 < 0.01
|   |   |   trade_size < 10.84
|   |   |   trade_size >= 10.84
|   |   trade_price_delta1_1.0_last1 >= 0.01
|   trade_type_last1 = 4
|   |   trade_size < 11.04
|   |   trade_size >= 11.04
trade_type = 4
|   trade_type_last1 = 2
|   |   trade_price_delta1_1.0_last1 < 0
|   |   trade_price_delta1_1.0_last1 >= 0
|   trade_type_last1 = 3
|   |   trade_price_delta1_1.0_last1 < 0.01
|   |   trade_price_delta1_1.0_last1 >= 0.01
|   trade_type_last1 = 4
```

Fig. 4: The tree was learned on a subset of the data. The first splits divide the data according to the current and last trade type.

strongly depends on whether they were between dealers or between dealers and customer.

We employed the preprocessing steps mentioned above with fine-tuned parameter settings for the final model. We also added higher order difference features on the trade prices and improved our model with boosting. In the end, we achieved a final score of 0.772 on the Kaggle leaderboard while the winning team got a score of 0.680 and thus outperformed our model significantly.

E. Strategies of Other Teams and Conclusion

The contestants are not required to publish their strategies and most keep their solution secret after the contest. In the Kaggle online forum discussion [Bon12] only few teams revealed their general strategies for the Bond contest. Hence, only a limited comparison to our approach is possible. We briefly present the best performing and disclosed strategies in the following.

Second Place [Bon12, Post #14]: The second placed team of Sergey Yurgenson and Bruce Cragin used an ensemble of Random Forests (RF) [Bre01] and Gradient Boosted Trees (GBT) ([Fri01]). They only required little preprocessing effort to obtain competitive scores.

Seventh Place [Bon12, Post #12]: Glen Koundry also used an ensemble of Random Forests and Gradient Boosted Trees to which he added a model based on locally weighted regression. As part of his preprocessing, he only used features from the four most recent trades, a strategy that we also employed.

Ninth Place [Bon12, Post #4, #8, #21]: Anil Thomas used Random Forests as a base regressor. Just as we did, he experimented with different problem formulations and predicted price differences as well as differences between the known prediction of the long term model and the actual trade price. In contrast to us, he did not pick the best-working approach but blended all models for a better score.

Two main differences between the best performing models and our approach are apparent. First, most good teams concentrated on random forests or gradient boosted trees. These models are more robust to hyper-parameter settings than our model and can be learned faster. Second, many contestants used a variety of models and blended them together to get a final ensemble (c.f. Section II-F), which outperforms each individual model. For our second contest, we adopted both methodologies. Moreover, we had familiarized ourselves with important software tools, which allowed us to approach the next contest more goal-oriented (e.g. avoiding manual work by automating the evaluation pipeline, see also Section II-B).

IV. Online Product Sales Contest

The Online Product Sales (OPS) Contest started on May 9^{th}, 2012 and ended on July 3^{rd}, 2012. The contestants should predict the number of online sales of a product for each month of the first year after product launch.

The main challenges of this contest are the large number of features, many missing values, unknown meaning of attributes and the small training set size. In total 366 teams participated in the contest and our team scored 7^{th} place.

In the following, we will first give some more details about the contest setting including the dataset and the evaluation process. Afterwards, we describe the main components of our prediction method. Subsequently, approaches are presented which have a reasonable motivation but turned out to be not fruitful for this particular

problem. Eventually, some key points of other teams' strategies are discussed.

A. Dataset and Evaluation

All teams are provided with the test set containing 519 samples with 546 attributes and a training set of 751 instances (products), which contains up to 12 labels for each product in addition. The labels correspond to the number of monthly sales of the product in the first year after product launch rounded in steps of 500. For some products, the sale numbers for the last months are not available. We assume that products with incomplete sale numbers either left the market within a year or have been released recently so that the sales numbers are not available yet. Missing labels are omitted during training and evaluation[13].

The performance of each team is measured with the *Root Mean Squared Logarithmic Error*

$$\text{RMSLE}(y, \hat{y}) = \sqrt{\text{MSLE}(y, \hat{y})} = \sqrt{\frac{1}{12n} \sum_{i=1}^{n} \sum_{j=1}^{12} \left[\log(y_{ij} + 1) - \log(\hat{y}_{ij} + 1) \right]^2} \quad (2)$$

where y_{ij} is the prediction of sales of product i in month j and $\hat{y}_{ij}$ the corresponding true sale number. Equation (2) is the (root) mean squared error of the labels in log-space. Hence, standard algorithms for least squares solutions can be used, if the labels are transformed a-priori by

$$\tilde{y} = \log(y + 1) \quad (3)$$

and the predictions are transformed back afterwards. In addition, the mean squared logarithmic error MSLE decouples into equally weighted terms for each month and product which allows to treat the task as independent one-dimensional regression problems.

The competition hosts obfuscated the meaning of attributes. A description is available only for two features: (1) the date when the product was announced and the advertisement started and (2) the date of the actual product sale launch. For a detailed discussion of those two dates consider Section IV-C. The remaining features are divided into 513 categorical (`Cat1`,...) and 31 numerical attributes (`Quan1`,...). Yet, the given partition is questionable, as some categorical features show characteristics of continuous data (e.g. almost all values differ) and vice versa. Almost all features contain a large number of missing values.

[13]While the entire instances are omitted for the line-fitting model, months with labels are kept for the Gaussian processes and the boosted trees.

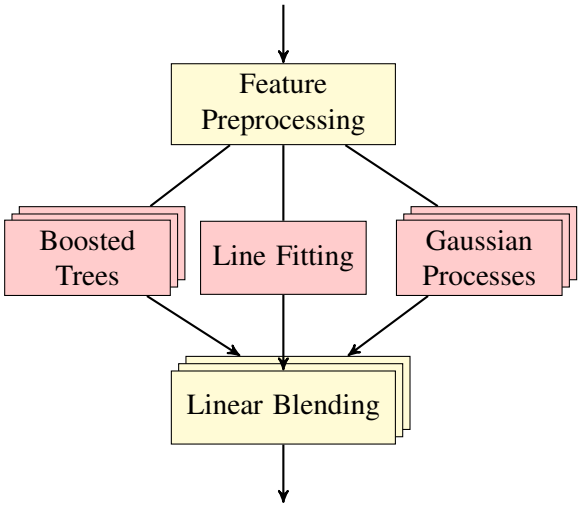


Fig. 5: Overview of our final model in the OPS contest: After preprocessing the features, they are fed into three kinds of prediction models: boosted trees, a linear fit with trees and Gaussian processes. The individual predictions are then merged by a weighted average and minor post-processing. The boosted trees, Gaussian processes and the final blending consist of single models for predicting the label of each month.

B. Overview of our Approach

Our final model consists of three individual models which are blended by a linear model [Bre96b]. See Figure 5 for an overview. The main component of our approach are gradient boosted regression trees. We chose them as a starting point since tree-based models can handle many features of varying quality and characteristics well. During the contest, we increased their performance by hyper-parameter optimization and further randomization to reduce their bias (c.f. Section IV-D for the details). Besides improving the boosted trees, we developed two alternative prediction models which themselves perform inferior. However, as the errors of the single models are only mildly correlated, we could decrease the overall error by computing a weighted average of the predictions.

Tree-based models generate piecewise constant regression functions, i.e., they cover the input space with axis aligned tiles of constant value each. To counterbalance the tendency towards axis-aligned solutions, we employed a Gaussian process as a second model. Gaussian processes [RW06] generate smooth regression functions and their output for a test point is determined by its distance to training points. Hence, their nature is quite different to decision trees. Details can be found in Section IV-F. Both models consist of separate predictors for each month, i.e., we trained a gradient boosted tree model and Gaussian process individually for each of the 12 months. Implicitly, we thereby assumed independence of the sales of a product for each months. Obviously, this

assumption does not hold as the sales are correlated over time. To circumvent this shortcoming, we added a third model which couples the predictions for each month. We chose to simply fit a line to the sales of each product throughout the year and then learn the parameters of this line. Again, we relied on boosted trees to do so. Details are presented in Section IV-E.

To find the optimal weights for the final blending of all models, we applied standard linear regression with the outputs of the three models as input. The outputs of each model were generated by 10-fold cross validation on the training dataset (always take the predictions on the test split). Afterwards we trained the weights for the blending by minimizing the training error. While this methodology may lead to overfitting, it yielded more robust results than evaluation on held-out data points, because of the limited number of training examples and low complexity of the linear model. Each month was blended with separate weights.

Apart from composing a good prediction model and tuning its hyper-parameters, preprocessing the input data had the largest effect on the final performance. While each model required its own input transformations to work well, some processing steps were beneficial for all models. We present those in Section IV-C.

C. General Feature Preprocessing

Feature engineering and preprocessing is hindered by the fact that the feature names were obfuscated. The only features with clear semantics are the dates when a product was announced and when it was launched. These are given as the number of days since some date in the past. Figure 6 shows a scatter plot of the release date as day of the year versus the sales record of the second month. There is a trend that sales are higher when the product is released around day 320, which suggests that there is a "Christmas" effect. Transforming the two date features modulo 365 results in a significantly smaller prediction error. We also experimented with other transformations of the date features, e.g. taking the month of the year or just the year, with no additional benefits to the overall model.

Aside of the date transformation, we also apply a logarithmic transform on features for which it is justified by an increasing predictive performance (see also Section III-C). The missing values are replaced with the median of all existing values. In addition, we omit features that were either constant or binary with one strongly predominant value. In Tables X and IX we see that while preprocessing is more crucial for the Gaussian process model, it also decreases the error for the gradient boosted tree model substantially.

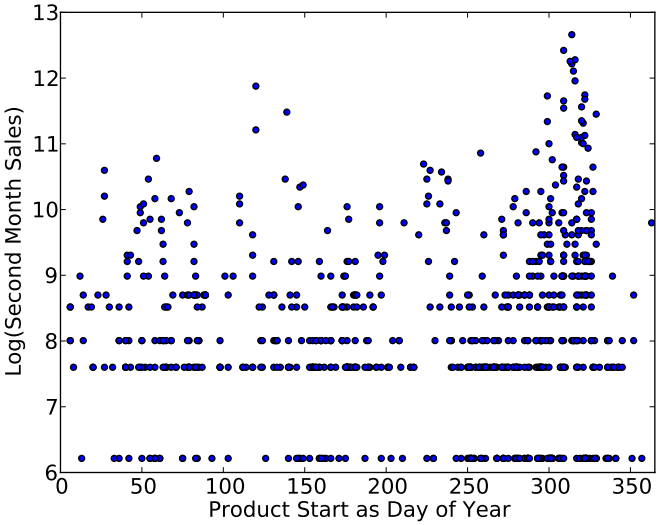


Fig. 6: Dependency of the release day in the year on product sales in the second month.

D. Gradient Boosted Trees

We decided to use decision trees for prediction to cope with the problem of many features of unknown quality. The most prominent regression models based on trees are Random Forests [Bre01] and Gradient Boosting Trees (GBT) [Fri01, Fri02]. As we observed consistently better off-the-shelf performance with the latter ones, we took them as a base model for further exploration. Gradient Boosting Trees allow to directly minimize different loss functions by iteratively fitting tree models. For the squared error loss this amounts to fitting each new tree to the residuals of the current model. In practice, it is necessary to learn each tree only on a subsample of the training data to prevent overfitting. We also follow the advice of Friedman in [Fri01] to limit the depth of each tree to a small value of six. As Figure 7 illustrates, the resulting GBT model is very robust to overfitting.

As gradient boosted trees basically perform gradient descent, they are prone to get stuck in local optima. One way to find different local optima is to add randomization to the optimization procedure [LBOM98]. The standard boosted tree model is already randomizated by subsampling the training instances for each new tree and is hence doing mini-batch gradient descent. We extended this idea and learn each tree on a random subsets of features and samples (*Gradient Boosted Random Trees* (GBRT)). This approach reduces the RMSLE from 0.593 to 0.576 (see Table IX). We had assumed that GBRT have lower variance than GBT, and our post-contest evaluation revealed, that it also reduces bias. Table VIII shows the squared bias and the variance of the GBT and GBRT model. Note, that the bias and variance of

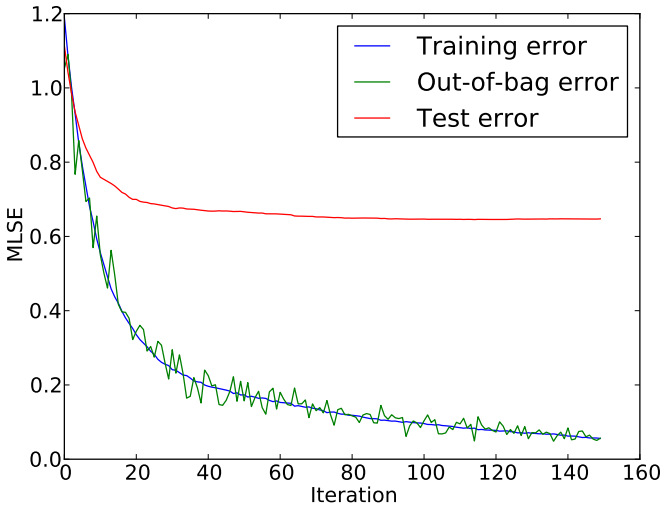


Fig. 7: This figure shows a typical evolution of test and training errors of a GBT for an increasing number of learned trees (iterations). As the training error converges to zero, the test error levels at a certain value and does not increase again.

	Bias^2	**Variance**
GBT	0.365	0.047
GBRT	0.364	0.040
RF	0.428	0.021

TABLE VIII: Squared bias and variance of the GBT model, Gradient Boosted Random Trees (GBRT) and Random Forests (RF). The introduction of feature subsampling in the GBRT model reduces the variance as well as the bias. Using RF yields a reduction of variance at the cost of an increased bias. Bias and variance are estimated by 100 iterations of 3-fold cross validation.

Model	**CV Score**	**Std. Dev.**
Our GBRT Model	0.576	±0.032
Our GBT Model w/o feat. subs.	0.593	±0.032
Our GBRT Model without prep.	0.598	±0.034
Split-randomized GBT	0.572	±0.034
Linear Model 1 Line	0.694	±0.038
Linear Model 2 Lines	0.637	±0.027
Blended Models (1 Line)	0.568	±0.035
Blended Models (2 Lines)	0.569	±0.033

TABLE IX: 10-fold Cross-Validation Scores of Different Tree-Models.

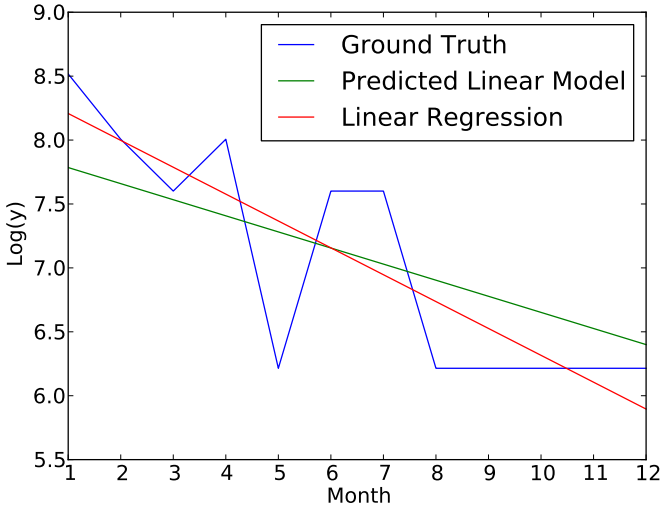


Fig. 8: The blue curve shows the log sales for a single training instance plotted over all twelve months. The red line shows the optimal linear regression model and the green line shows the line with predicted slope and intercept.

the model are directly related to the mean squared error (MSE) by the equation MSE $=$ bias^2 $+$ variance.

E. Line Fitting with GBTs

We include a model in our ensemble, that predicts the sales of all months jointly, to account for the correlation of the sales in different months. Instead of using individual GBRTs to estimate the sales for each month, we assume that the sales change only linear over time and predict the intercept and slope of this linear progress. See Figure 8 for an illustration. As the assumption of linearity does not hold on the training examples, we estimated their slope and intercept by a least-squares line-fit. Table IX shows that this approach yields error of 0.694, which larger than estimating the sales for each months individually. But as it differs sufficiently from the other models, it is useful as a component in the final ensemble.

We also experimented with fitting two line segments, which assumes that the temporal change of the sales of 12 months can be explained by two lines (6 months each). While fitting multiple lines improves the model on its own, it harms the performance of the final ensemble (see Table IX). We believe that this is due to the changed model being more similar to the model with individual GBRT for each month and their errors hence being more correlated.

F. Gaussian Processes

The main challenge of using Gaussian processes on the OPS dataset is to design a good distance measure, i.e., preprocess the attributes and use an appropriate kernel to obtain distances relevant for the task. Gaussian processes work particularly well in continuous spaces and smooth target functions. However, most nominal attributes in the OPS dataset only take less than 10 values and do not have a natural order of the values. We therefore excluded features which take less than 60 unique values on the training set. The cross-validation results in Table X (c.f. the entries with different min. val. settings) show that this threshold is the optimal trade-off between reducing noise by omitting harmful features and including helpful features. The cross-validation scores in

the table are slightly biased towards lower thresholds as the set used for training during cross-validation is smaller than the true one. Yet, we found it not to be significant by checking against the public leaderboard score.

Besides selecting the right subset of features, normalization and the kernel choice play an important role in optimizing the model. While the Gaussian kernel is the best distance measure in many applications, we experienced that the *absolute exponential kernel*

$$k(x, y) = \exp\left(\sigma^{-1}\|x - y\|_1\right) \tag{4}$$

decreases the error significantly (c.f. Table X), where $\sigma = 10$ is the length-scale. This observation is consistent with the good performance of decision-tree based models as both approaches prefer axis-aligned regression functions. As shown in Table X, the attributes need to be normalized, e.g. with mean zero and variance one, to obtain reasonable prediction performance.

Independent Gaussian processes are trained for each response variable but we also tried to include the month as an additional feature and use a single Gaussian process for all predictions. While the performance is on par with the separate models (c.f. All-in-one GP Model vs. Single Monthly GP Model in Table X), the memory consumption increases drastically as there are 12 times as much training samples which go in quadratically instead of linear as for separate models trained in parallel. As the memory requirement of 8 *GB* prevent computing cross-validation scores in parallel and thus slows down the evaluation process, we followed the individual model approach.

The prediction error can be reduced further with Bagging proposed in [Bre96a]. Instead of training one Gaussian process per month, the prediction of 200 GP models are averaged. Each model is trained on a random subset of the training data, consisting of 11 features[14] each and 70% of the instances. In the end, the Gaussian process model consists of $12 \cdot 200 = 2400$ individual models and yield a cross-validation score of 0.607.

G. Unfruitful Ideas

While developing our final model, we experimented with several things which turned out not to improve the predictive performance of the model. In this section we shortly discuss the ideas which did not work out as expected. As an alternative to GBT, we also tried out Random Forests (RF). Random forests are bagged decision trees, i.e., the average of independently trees trained on a random subset of the data. Therefore, random forests have usually lower variance than GBT at the cost of having a higher bias. A bias-variance decomposition of our GBRT model and random forests shown in Table VIII support this statement. Since the bias is the dominating source of error in the OPS contest, random forests yield inferior performance.

[14]Each feature still need to take at least 60 different values on the training set

Model	CV Score	Std. Dev.
Bagged Monthly GP Model	0.607	±0.031
Single Monthly GP Model	0.621	±0.032
Monthly GP Model no date feat.	0.650	±0.033
Monthly GP Model no log trans.	0.623	±0.031
Monthly GP Model Squared Exp.	0.833	±0.052
Monthly GP Model unnormalized	1.014	±0.047
All-in-one GP Model	0.623	±0.031
Monthly GP Model minval=3	0.679	±0.037
Monthly GP Model minval=30	0.619	±0.029
Monthly GP Model minval=40	0.618	±0.031
Monthly GP Model minval=60	0.618	±0.031
Monthly GP Model minval=200	0.790	±0.031

TABLE X: 10-fold Cross-Validation Scores of Different GP-Models.

As all sales numbers are rounded to multiples of 500 and the sales of all products are low in the last months, the target values of these months take only a few values in the training set. Hence, it can also interpreted as a classification task, which might be easier to learn. However, the caveat of this approach is the complex loss function between the different values as the RMSLE in the original formulation yields different penalties for each pair of classes. We had no implementation of a classifier available that could directly minimize such a loss function and the short timespan of the contest did not allow to implement one ourselves. Therefore, we resided to classifiers minimizing the standard 0/1 loss, which lead to poor results.

Using a plain linear regression model also turned out to yield a bad performance. As the dataset had many features but only a very limited amount of data, we tried Lasso Regression as it selects the relevant features due to its ℓ_1-regularization. However, fitting a linear model resulted in poor error scores because the linearity assumption does not hold on the given features, i.e. extensive preprocessing would have been necessary.

H. Strategies of Other Teams and Conclusion

As in the Bond contest, we can only rely on sparse information about strategies of other teams from the Kaggle online forum discussion [OPS12]. In the following we briefly present an overview of the best performing and disclosed strategies.

First Place [OPS12, Post #6]: The winner Peter Prettenhofer employed only a single gradient boosted tree model with similar pre-processing as ours. He

reduced the variance of the model by sub-sampling the features during learning for each split of the decision trees, while we sub-sampled the attributes similarly but only once per decision tree. Moreover, he put more effort in tuning the hyper-parameters such as tree depths, learning rate or minimum number of instances in the tree leafs by exhaustive grid-search which required large computational power. Based on this information, we tried to reproduce his results but only obtained better results than our GBRT model (see Table IX).

Second Place [OPS12, Post #3]: Xavier Conort relied on gradient-boosted regression trees and random forests. The single models were combined by a generalized additive model [HT86] with a cubic spline smoother. While we trained GBRT models for each month individually, he included the month to predict as a feature and trained only a single model.

Fourth Place [OPS12, Post #16]: Shea Parkes also considered the prediction for all month as a single regression problem and predicted with a linear combination of random forests and neural nets. To reduce the variance, he subsampled the features during training of the neural net (bagged neural nets [HTF08]).

In general, models based on decision trees and in particular boosted decision trees seemed to be particularly suited for this contest due to their robustness to irrelevant features. The transformation of the date feature was another key element to accurate predictions for most teams.

As the comparison to the first place shows, a single perfectly tuned model would have sufficed to win the contest. Influenced by our experience from the Bond contest, we focused more on creating a bundle of good methods for the ensemble instead of tuning the hyper-parameters excessively. To keep the computational costs manageable, we did not fully explore the hyper-parameter space. For example, we also tried to randomly subsample attributes per split instead of tree for the GBT model, but did not experience any benefit for the small number of trees in our evaluation. Nevertheless, the lessons from the Bond contest, e.g. fully automating the evaluation process, relying on boosted trees and building an ensemble, definitely paid off. For future contests, we would keep the methodology but allow more computational effort for tuning the hyper-parameters of each model.

V. Conclusion

We have shown in this report that a strong theoretical background as well as practical experience and a sound methodology is needed to be on par with the high performing teams in machine learning contests. We want to emphasize one more thing that cannot be conveyed by thorough analysis: Participating is a lot of fun! Creating ideas and putting them to work, seeing your team rising and falling on the leaderboard, exchanging thoughts within your team and with other competitors in the forum generates a highly motivating atmosphere while the deadline is inevitably approaching. In the end, you find yourself spending a lot more time on the contest than you expected. We encourage everyone who is interested in machine learning and has not yet gained practical experience to take the chance these contests offer.

References

[BB12] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb), 2012.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, 2011.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BKV08] Robert M. Bell, Yehuda Koren, and Chris Volinsky. TheBellKor 2008 Solution to the Netflix Prize. Technical report, Statistics Research Department at AT&T Research, 2008.

[Bon12] Kaggle Discussion Forum – Benchmark Bond Trade Price Challenge – Congratulations. https://www.kaggle.com/c/benchmark-bond-trade-price-challenge/forums/t/1833/congratulations, April 2012.

[Bre96a] Leo Breiman. Bagging Predictors. *Machine learning*, 24(2), 1996.

[Bre96b] Leo Breiman. Stacked Regressions. *Machine learning*, 24(1), 1996.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1), 2001.

[CM12] Dan Cires and Ueli Meier. Multi-column Deep Neural Networks for Image Classification. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3), 1995.

[Dah12] George Dahl. Deep Learning how I did it: Merck 1st place interview. http://blog.kaggle.com/2012/11/01/deep-

learning-how-i-did-it-merck-1st-place-interview/, November 2012.

[Fri01] Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5), 2001.

[Fri02] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4), 2002.

[GCB97] Patrick J. Grother, Gerald T. Candela, and James L. Blue. Fast implementations of nearest neighbor classifiers. *Pattern Recognition*, 30(3), 1997.

[GCSR03] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis, Second Edition (Texts in Statistical Science)*. Chapman & Hall/CRC, 2003.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1), 2009.

[HHLB11] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization*, 2011.

[HHLBM10] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Kevin Murphy. Time-Bounded Sequential Parameter Optimization. In *Learning and Intelligent Optimization*, 2010.

[HS12] Geoffrey E. Hinton and Nitish Srivastava. Improving neural networks by preventing co-adaptation of feature detectors. *eprint arXiv:1207.0580*, 2012.

[HT86] Trevor Hastie and Robert Tibshirani. Generalized Additive Model. *Statistical Science*, 1(3), 1986.

[HTF08] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2008.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3), 2007.

[JN02] Michael I. Jordan and Andrew Y. Ng. On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, 2002.

[LBOM98] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: tricks of the trade*. Springer, 1998.

[LN89] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3), 1989.

[MAT10] MATLAB. *version 7.10.0 (R2010a)*. Natick, Massachusetts, 2010.

[Mur12] Kevin Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[OPS12] Kaggle Discussion Forum – Online Product Sales – Congrats to the winners. http://www.kaggle.com/c/online-sales/forums/t/2135/congrats-to-the-winners, July 2012.

[OS05] Arthur O'Sullivan and Steven M. Sheffrin. *Economics: Principles in action*. Prentice Hall, 2005.

[PG07] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Computing Science and Engineering*, 9(3), 2007.

[PVG+11] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, and David Cournapeau. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2011.

[R C12] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2012.

[RW06] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[SRH+10] Sören Sonnenburg, Gunnar Raetsch, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio De Bona, Alexander Binder, Christian Gehl, and Vojtech Franc. The SHOGUN Machine Learning Toolbox. *Journal of Machine Learning Research*, 11(Jun), 2010.

[SS04] Alexander J. Smola and Bernd Schölkopf. A Tutorial on Support Vector Regression. *Statistics and computing*, 14(3), 2004.

[VTS04] Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. A primer on kernel meth-

ods. In *Kernel Methods in Computational Biology*. MIT Press, 2004.

[Was07] Larry Wassermann. *All of Statistics*. Springer, 2007.