Machine Learning Engineer Nanodegree

Insurance Claim Prediction - Capstone Project

Maximilian Li
2016-11-01

1. Definition

Project Overview

This project is based upon a competition hosted by Kaggle

The insurance company guarantees the policyholder a payout if certain damages occur, like a house fire or car crash. For this insurance the insurance holder pays a monthly rate. To prevent abuse of the insurance policy, the insurance company has many levels of checks in between the insurance claim and the payout. This should prevent fraudulent claims like the policyholder laying fire to his own house or otherwise willfully damaging the insured property. In the past companies have used their data to automate the approval process of incoming claims (See: kaggle/bnp-paribas). In this project, the severance of the claim (amount of money lost for the company) shall be predicted and the application of Machine Learning in a Business Domain exercised.

Problem Statement

The kaggle-competition provider Allstate is searching for an automated way to predict the amount of money claimed by their insuree. For this purposes several machine learning models shall be trained on provided data, fine-tuned without overfitting the models and the best performing model determined.

Metrics

The model is evaluated on the mean absolute error (MAE) between the predicted loss and the actual loss on the test.csv dataset. The formula for the MAE is defined by:

$(1/n) * \sum|y_i - y'_i|$

with y_i being the prediction, and y'_i being the actual outcome of one event. This will compute the average "distance" of a prediction to the actual outcome.
https://www.kaggle.com/c/allstate-claims-severity/details/evaluation

2. Analysis

Data Exploration

The company Allstate provided the platform kaggle with an anonymized dataset. This dataset consists of: [Unique ID] [116 categorical features] [14 continuous features] [target feature: loss] with a total of 188.318 datapoints.

id	cat1	cat2	...	cont13	cont14	loss
1	A	B	...	0.822493	0.714843	2213.18
2	A	B	...	0.611431	0.304496	1283.60
5	A	B	...	0.195709	0.774425	3005.09

Statistical Analysis

Statistics for Insurance claims dataset:

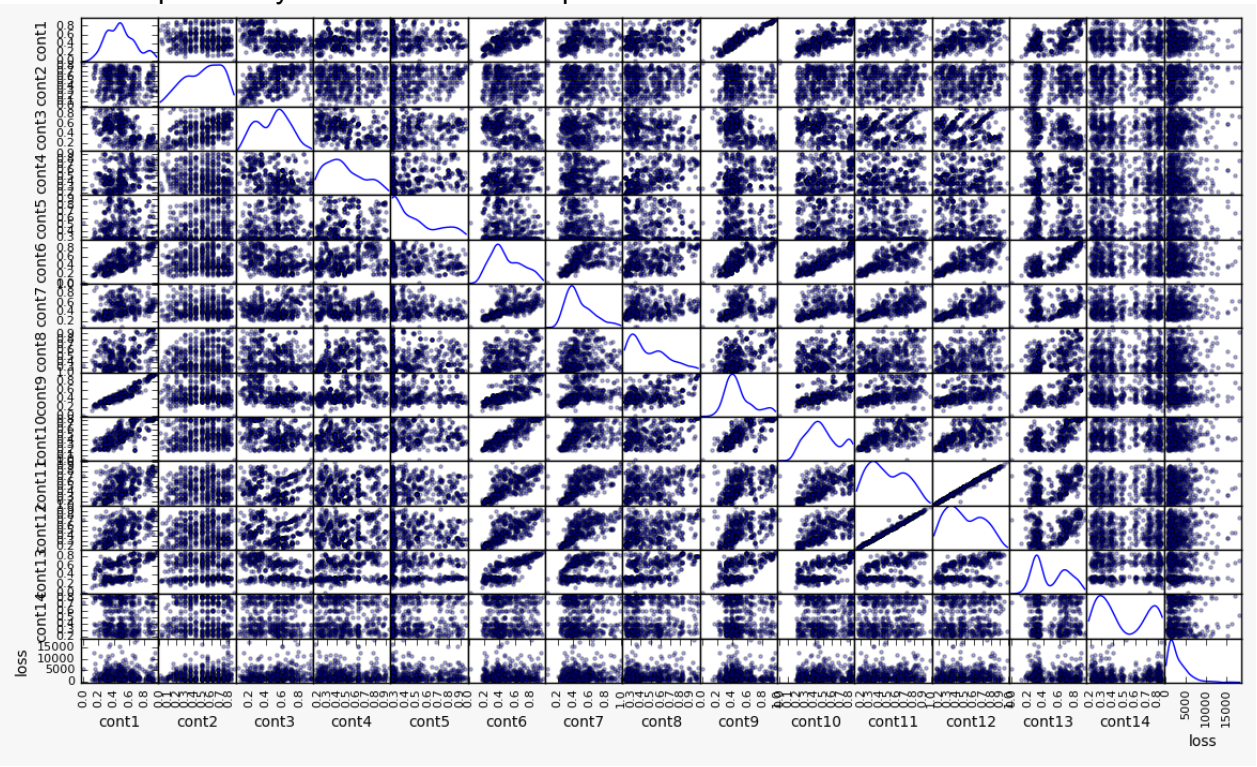
Minimum claim	$0.67
Maximum claim	$121,012.25
Mean claim	$3,037.34
Median claim	$2,115.57
25% percentile	$1,204.46
50% percentile	$2,115.57
75% percentile	$3,864.05
Interquartile Range	$2,659.59
Standard deviation of claims	$2,904.08

This information helps to give context to future predictions. Also the Minimum of $0.67 and Maximum of $121,012.25, and their huge distance to the next quartile, are already showing that some outliers will have to be removed.

The interquartile range is smaller then I would have expected for this domain with $2,659,59.

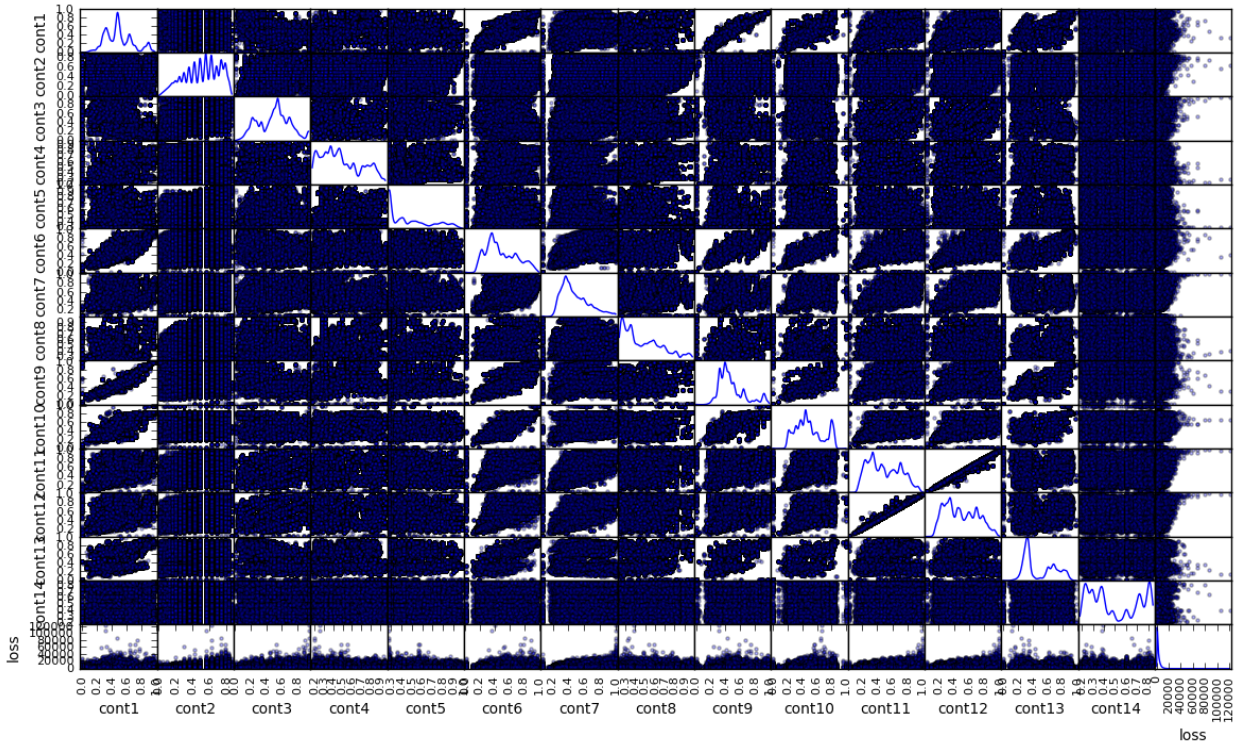
Exploratory Visualization

A scatterplot matrix helps to get a better understanding of the data. For performance reasons, the data is split to only contain the first 500 points.

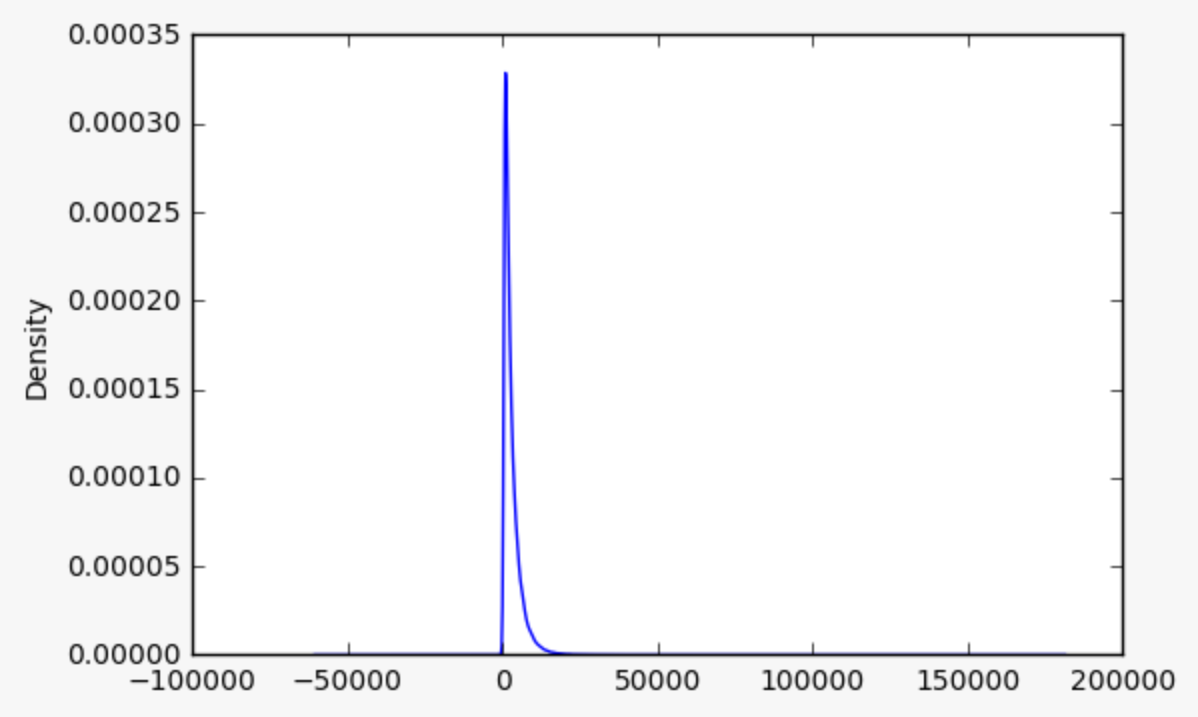


One can already see correlations, that can be used to trim down the dataset using feature selection. Most notably are *cont11* <-> *cont12*, which seem to correlate strictly linearly. Some other correlations can also be seen, but not as clearly, like *cont1* <-> *cont9*.

This is the scatterplot matrix over **ALL** the data points:



The scatterplot matrix also shows the right skewness of the target feature 'loss', which becomes more pronounced, if we plot it alone:



Algorithms and Techniques

The exploration of the dataset already showed some properties, which one has to consider:

- *Normalization* using Logarithm: The target feature needs to be normalized for the regression model to yield the best results possible, due to the right-skewness of the target feature.
- *Label-encoding:* The 116 categorical columns need to be Label-encoded. This turns the labels like “A”,”B”... into numerical values 0 and 1.
- *One-Hot-Encoding:* Expresses the exclusivity of the categorical values, so that the learning model understands, that 0 and 1 from the example above are exclusive categories, and not an ordering. To achieve this, the range of possible values [A,B] is transformed into a vector $[V_{1,} V_2]$ with a 1, if the corresponding value is stored for this datapoint.
- With the amount of features, and especially after the One-Hot-Encoding, one has to apply *Feature Selection/Dimensionality Reduction*.

The models itself will be:

- *Decision Tree*
  - Pro: Lightweight
  - Pro: Simple
  - Contra: Easily overfits
  - Why this model: Very simple approach. If it is as good as the other models, I would prefer the simpler solution
- *Random Forest*
  - Pro: Similar strength like Decision Tree
  - Pro: Does not overfit as easily
  - Contra: Slower - sacrifices Speed for Accuracy
  - Why this model: All the advantages of a Decision Tree, but generally better scoring performance
- *SVM*
  - Pro: Good at handling large input spaces
  - Pro: Due to different kernels can handle nonlinear data well
  - Contra: Slow
  - Why this model: The Dataset has a very large input space (130 columns without any encoding work done). This is a strength of the SVM.
- *kNN*
  - Pro: Different approach than the other models: more like a database
  - Pro: All the training data is used
  - Pro: Works by nature through similarities in data
  - Contra: Slow
  - Why this model: We have a large Dataset with many entries (~190k). Maybe this model will be able to handle all this data in a timely fashion. Also I would imagine it is a good model for large Datasets like in a real-world business case and this is a good opportunity to work with it.

To create the training and testing sets sklearn's *train_test_split()* function will be used.

The models will be tuned using cross-validation like *GridSearch*.

As mentioned, the mean absolute error shall be used as a performance measure. As we also have to use a normalization on the target feature, and thus the prediction, I define my own scorer function, which "denormalizes" the inputs to better understand the performance in regard to the domain.

Benchmark

The benchmark model will always predict the mean of the training set. The performance is measured with the mean absolute error as requested by the Kaggle competition.

The implementation will be shown after the Preprocessing stage, as outliers will influence the mean absolute error very heavily.

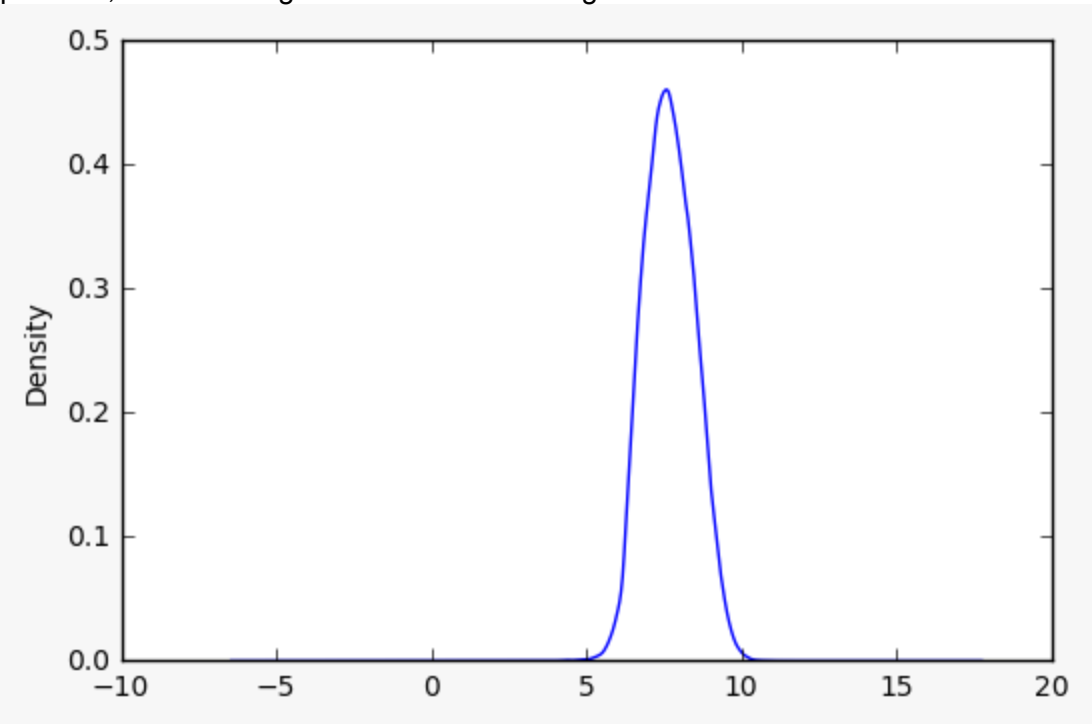
The benchmark, which always predicts the mean of the training set, has a baseline performance of 1758.93 .

3. Methodology

Data Preprocessing

Normalization

The target feature needs to be normalized for the regression model to yield the best results possible, due to the right-skewness of the target feature:



Outlier Detection

The big difference between Maximum and Minimum, and their distance to the nearest quartile already show some outliers need to be removed.

The following code will remove all the data points with a bigger distance to the nearest quartile than 1.5 * the interquartile range.

Q1	7.0938
Q3	8.2597
Step	1.7485

Example of outliers:

Row number (NOT ID)	log(Loss)
89	3.6481
470	5.2604
713	10.1647

The outlier removal process found 521 entries outside of 1.5 * the interquartile range to the nearest quartile. This is equal to 0.27% of the total dataset.

After this stage the dataset now holds 187,797 entries

New Statistics without Outliers:

Minimum claim	$210.42
Maximum claim	$22,184.84
Mean claim	$3,004.25
Median claim	$2,115.39
25% percentile	$1,206.31
50% percentile	$2,115.39
75% percentile	$3,855.56
Interquartile Range	$2,649.25
Standard deviation of claims	$2,700.49

Label Encoding

The Labels from the categorical columns are encoded.

id	cat1	cat2	cat3
1	0	1	0
2	0	1	0
5	0	1	0

Onehot Encoding

The exclusivity of the categorical values is expressed through the One - Hot - Encoder (See Algorithms and Techniques).
Unfortunately thus many new input dimensions are added to the dataset. The original 130 columns have been extended to 1148 dimensions by the One Hot Encoder.

Dimensionality Reduction

This 1148 dimensions need to be processed and reduced to improve the performance of the models. For this PCA is used.

By processing the input dimensions down to 100 dimensions, 84.4% of the variance in the dataset is explained. 100 dimensions seem to be a good tradeoff between precision and complexity, but might be tuned later.

From this point on forward the models will learn with a denser dataset consisting of 100 columns.

Creation of Training and Testing Sets

To avoid overfitting, the data is split into a testing and training set. For this *scikit-learn*'s *train_test_split()*-function is used. 25% of the data is used as the testing set.

Implementation

A Benchmark model is tested, and then the 4 learning models. For performance reasons, they will go through a training phase with 1,000; 5,000; 10,000 and 20,000 training samples.

1: Decision Tree Regressor

Number of Training Samples	**MAE Score**	**Compared to Benchmark** (lower is better)
1000	2005.61846986	+14.02%
5000	1905.58202612	+8.34%
10000	1864.64465207	+6.01%
20000	1849.54474106	+5.15%

2: Random Forest

Number of Training Samples	**MAE Score**	**Compared to Benchmark** (lower is better)
1000	1422.08665043	-19.15%
5000	1366.24618815	-22.33%
10000	1340.27246782	-23.80%
20000	1325.98813823	-24.61%

3: SVM

Number of Training Samples	**MAE Score**	**Compared to Benchmark** (lower is better)
1000	1258.64964054	-28,44%
5000	1202.59485141	-31.63%
10000	1192.06821899	-32.23%
20000	1176.22332253	-33.13%

4: KNN

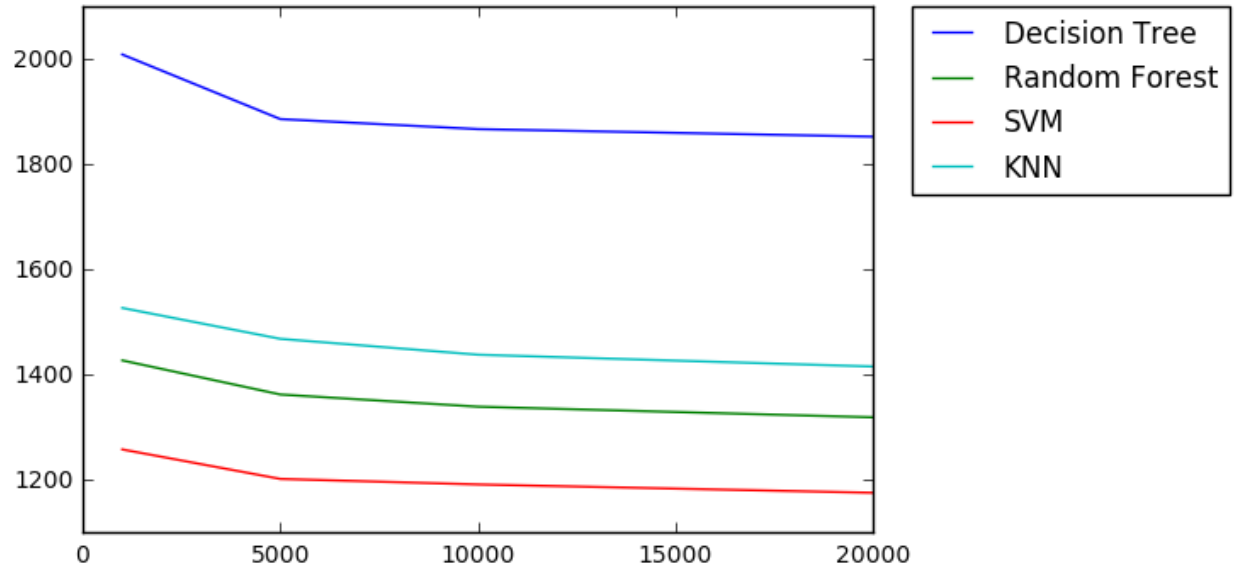
Number of Training Samples	**MAE Score**	**Compared to Benchmark** (lower is better)
1000	1527.73184284	-13.14%
5000	1468.98012290	-16.48%
10000	1438.80938107	-18.20%
20000	1416.32361958	-19.48%

5: First Reflection

The implementation of the models themselves was relatively straightforward. Because the computation of especially the SVM and KNN Regressors took quite a while, I ended up implementing the *trainTestRegressor()* - function to avoid duplicate code and restrict the amount of data used for training. This was then refined to make multiple runs with performance measured for each *Model-Num_of_Samples* pair and the resulting information was used in the Refinement chapter.

Refinement

The following plot shows the performance increase with datapoints added (lower is better):



All models gain a relatively big increase in performance, by training with 5000 samples. Especially the Decision Tree will improve with even more data points, but as the tuning of the model parameters is computationally quite expensive, I will use 5000 samples as the training set for the parameter tuning. This is a preferable tradeoff between performance and scoring.

A GridSearch implementation searches for the best parameters for all 4 of the above defined models:

1: Decision Tree Regressor

The GridSearch tries to tune the *max_depth* parameter of the decision tree.

MAE Score	**Compared to Benchmark** (lower is better)	**Compared to old Model** (lower is better)
1675.36308342	-4.75%	-22.18%

2: Random Forest

The GridSearch tries to tune the *max_depth* and *n_estimators* parameter of the Random Forest.

MAE Score	**Compared to Benchmark** (lower is better)	**Compared to old Model** (lower is better)
1332.71812298	-24.23%	-2.45%

3: SVM

The GridSearch tries to tune the *C* and *kernel* parameter of the Random Forest.

MAE Score	**Compared to Benchmark** (lower is better)	**Compared to old Model** (lower is better)
1202.59485141	-31.63%	±0.00%

4: KNN

The GridSearch tries to tune the *n_neighbors* and *algorithm* parameter of the Random Forest.

MAE Score	**Compared to Benchmark** (lower is better)	**Compared to old Model** (lower is better)
1450.47106324	-17.54%	-1,26%

The SVM from the GridSearch-Optimization scores the same MAE as the untuned model. Despite this it is still the best model of the 4 models evaluated.

4. Results

The final solution chosen is a Support Vector Machine Regression with the default parameters by scikit-learn 0.17. It might take a longer time to compute than the Decision Tree, but is able to handle the big input space much better and achieves a better score than all the other models.

With 20000 training samples it achieved a MAE of 1176.223, which is 66.87% of the MAE made by the Benchmark model and less than half a standard deviation of the target feature without outliers.

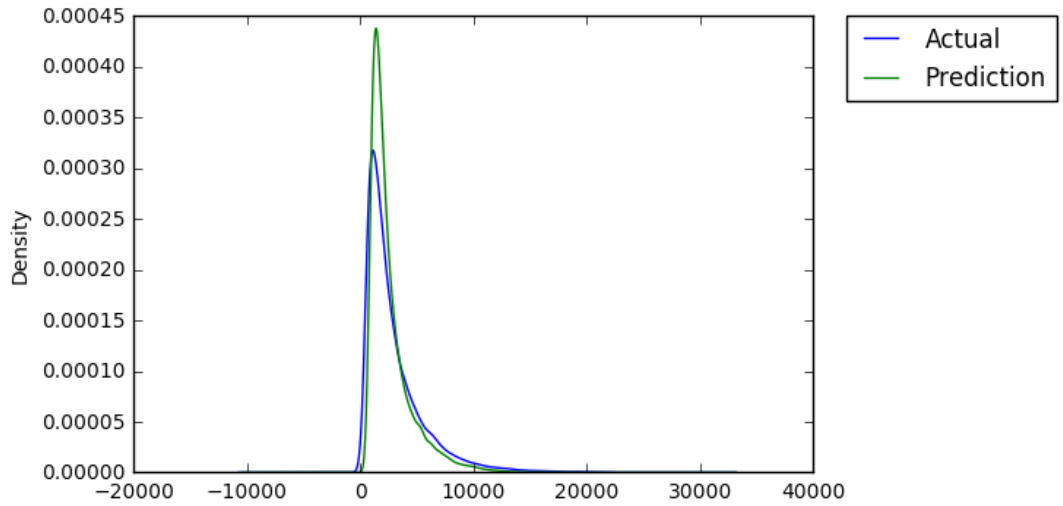
As this model has been trained with the first 20,000 data points in the X_train set generated by the *train_test_split()* function, we can do a simple robustness test with the remaining data points. For this we make several runs with 20,000-samples-sized chunks of the remaining training data and compute an average MAE score for the model.
The model's average performance is 1182.6008. This is about 6 points higher than the

performance score from the testing phase, but still about 600 points lower than the Benchmark. This deviation is small enough, that I would consider this model robust.

5. Conclusion

Free-Form Visualization

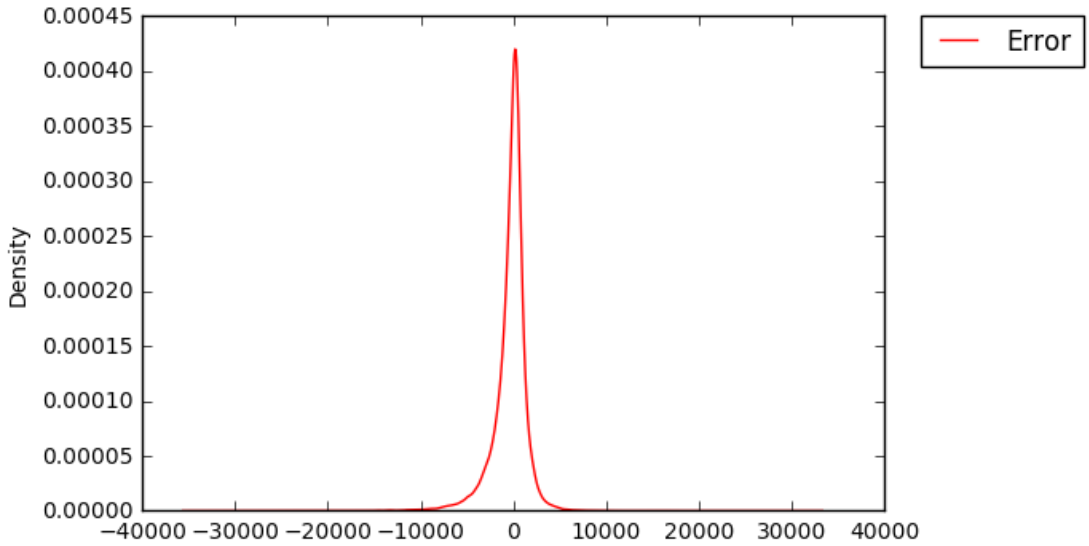
This visualization shows the distributions of the model's predictions and the actual values:



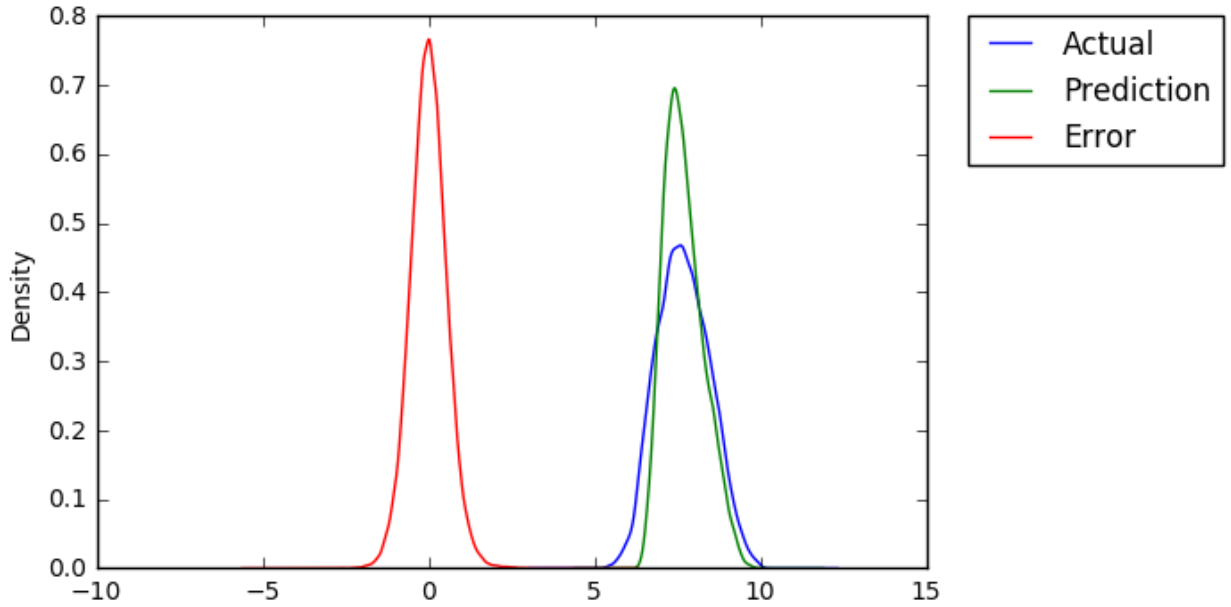
One can see, that the most distributed value (the highest peak) is for both sets roughly the same value. But the prediction is somewhat hesitant to predict the values outside of this peak, the distribution is concentrated around this peak.

This is further shown by two graphs:

1. The Distribution of the Error: Most of the Predictions have an error of 0. The prediction is most likely to predict less than the actual data shows, the graph is more pronounced on the negative side:



2. The Distribution of the Values with normalization: If one plots the normalized data and errors, the meaning of the peaks becomes clearer:



One can now see, that the peaks of the distributions is also the mean for the distribution. Thus the error is most likely 0.

Reflection

This project developed a model to predict the severance of insurance claims, given 116 categorical and 14 continuous inputs.

The data was normalized and, using a statistical analysis, outliers removed.

The categorical values were processed using Label- and One-Hot-Encoding. This resulted in 1134 input dimensions, which were reduced to 100 dimensions using Principal Component Analysis. These 100 dimensions explain 84.4% of the variance in the data.

Using this data, 4 different Models, a Decision Tree, a Random Forest, a Support Vector Machine and a kNN-Regressor were trained and evaluated against a Benchmark model predicting the statistical mean of its training data.

This evaluation found that the SVM was the best performing model of the chosen 4.

Two aspects of this project I found rather difficult:

1. **The Anonymized data:** Because the dataset was heavily anonymized, it was difficult to apply any domain knowledge one might have. This would have made the feature selection and interpretation of PCA a lot easier.
2. **Size of the data:** It was more difficult than I expected at the beginning of the project to deal with this dataset. For a real-world business scenario optimization for multi threading/parallel programming or distributed systems would have been mandatory, but I decided these were features out of scope for this project.

Improvement

There is one big issue with this implementation: The provided dataset holds aprox. 190k data points. This was a dataset to big to handle efficiently for this implementation. One would need to improve the preprocessing stage or utilize multithreading, learning the whole dataset with a model took a very long time (as a fact I was not able to finish the .fit() phase with the whole dataset). Using this complete dataset, a better solution might be found.

The next possible improvement exists in the selection of the model: This project compared 4 models against a benchmark model. With a different selection a better model might be found.

Also a thing I would have liked to try out: With v0.18 of scikit-learn, Decision Trees (and the Random Forest Model) are now able to split using the MAE as a metric. Until now only MSE (Mean Squared Error) was available. I would have liked to see, if the models would have improved, as we also use the MAE to rate the overall performance of the models.