Exploration

Probing the outcomes of different Train-Test Splits

Marco A. Wedemeyer - July 2020

One of the courses offered in the Data Science program at Tilburg University is an introduction to data mining. We were taught the basic vocabulary and techniques of machine learning, for instance, the difference between supervised and unsupervised classification, model parameters and hyperparameters and also best practices for training these models. The core pillar of proper training was to not test your model's performance on the training data but rather on a seperate test set. The underlying algorithms are designed to minimise prediction error and thus have a tendency to over optimise the solution to the data at hand. Testing the models ability to generalise what it has learned by seeing how well it performs on unseen data is key to making useful predictions.

As a data scientist one is mostly limited to a particular data set. Thus, the practical solution is to split the given data set into separate train and test sets[1]. The obvious question is how to perform the split. What proportion of the data should be used for testing? What are the consequences of changing this parameter? How consistent will my results be if I rerun the test?

In our data mining course we were given the rule of thumb of 80-20 (train-test), however, advice online ranges from 50-50 to 90-10. Rules of thumb are designed to give the best average result for the most common cases and thus do not provide sufficient guidance in unconventional situations. To further my understanding of how machine learning algorithms see the world, I decided to explore how different factors affected the reported error by model.

During our first tutorium we were introduced to the python machine learning package sklearn and the Boston Housing data set. Sklearn provides the function `train_test_split`, which performs the desired train-test split on provided data. It also takes as input a desired `test_size` and a `random_state` parameter for replicability.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=j, random_state=i)
```

I started simple by adjusting the test_size parameter in steps of 0.01 between 0 and 1. As the Boston Housing data set has a continuous target, I used the `LinearRegression` class provided by sklearn to model the data. The results of this initial testing can be seen in Figure 1. The rule of thumb of 80-20 is marked with a black dot. One can see that the reported *root mean squared error* (RMSE)[2] changes depending on the proportion chosen. Interestingly, if the value had been just a little smaller (0.15) the reported RMSE would have been 25% smaller.

[1] It is common to split the data three ways to generate a validation set used for model tuning, however, this set will not be needed in this exploration.

[2] The performance of a regression model can be measured using the root mean squared error, which is the average distance between predicted values and the actual target values.

This result indicates that the choice in `test_set` proportion can have a significant impact on the reported error of the model. The graph shows a particular trend but a rather noisy one. In order to understand the more general relationship underlying this trend I decided to simulate many more of these trials. In order to achieve different results I looped over 100 different `random_state` values. The resulting spaghetti graph is shown in Figure 2.

Although no individual line follows the overall relationship, we can still infer a general pattern. Like a funnel the lines start off with a wide spread in RMSE when the test set proportion is close to zero. As the test set proportion increases the variability of the lines reduces. This is due to the increasing probability that the test is representative of the underlying relationship. This trend continues until shortly after a 15-85 split when the RMSE sharply increases. With such little data the model is unable to learn a generalisable representation of the data.

Figure 3 is a cleaner representation of the findings in FIgure 2. The mean RMSE remains stable even in the highly dispersed left tail. The median being very close to the mean indicates little to no skewness. A clear **L** shape is identifiable. The model is able to learn the necessary coefficients after around 15% of the data and little improvement is made with more training data.

Figure 4 shows a scatter plot of the mean and the standard error of each test set proportion (color) over the 100 bootstraps. A clear **V** shape is visible. For the Boston Housing data set it appears that the lowest standard error is achieved at a 38-62 train test split. Lower test set proportions show

Figure 1

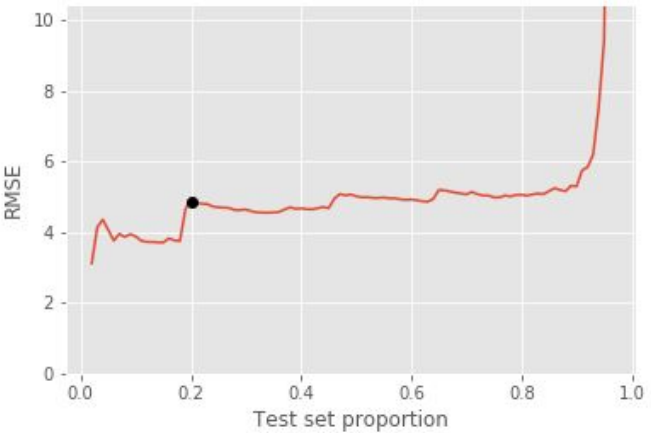


Figure 2

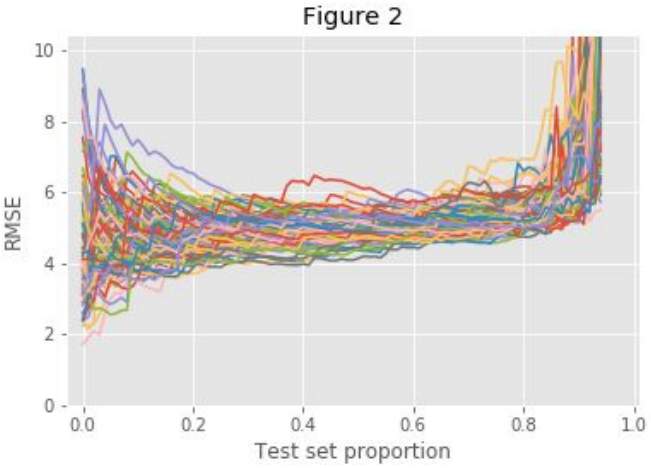


Figure 3

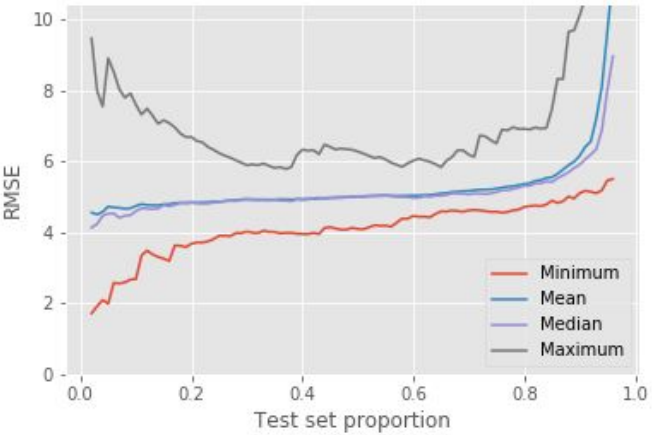
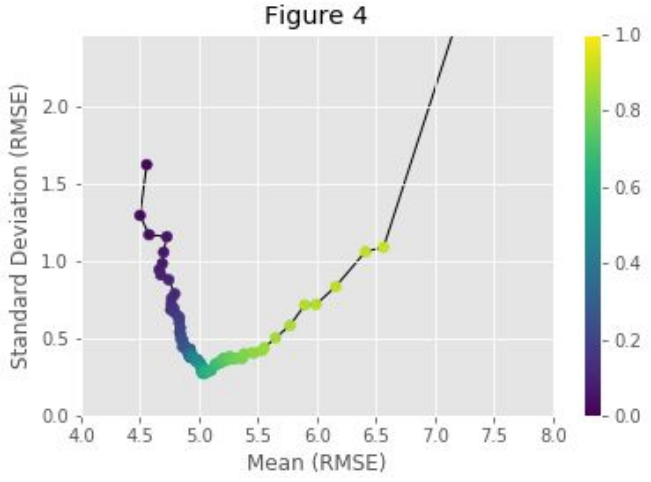


Figure 4



marginal improvements in the mean RMSE while the standard error increases.

Conclusion - The challenge underlying the train-test question is to find the right balance between having a sufficient proportion of data in the training set to learn a good representation of the underlying trend while minimising measurement error induced by an unrepresentative test set. The rule of thumb given in our data mining course was given with the expectation that it would be used on real data sets with more noise and more (informative) features. Most importantly it is expected to be used on far larger data sets. These three factors: 1) Noise, 2) (Informative) Features and 3) Sample size are explored in the next piece.