<u>Machine Learning - Assessed Exercise</u>
<u>Final Report</u>

Kenneth Murray - 2140010M
Jamie Sweeney - 2137284s
Lukáš Krauz - 2376641K

Introduction

In this assessed exercise we are tasked with developing 2 regression algorithm and 2 classification algorithms for different problems. We are given training data for each task, as well as test data to submit predictions for.

For the regression task, we are provided with a data set of CPUs with each element consisting of a set of 6 features:

- Cache memory size.
- Minimum and maximum number of IO channels.
- Cycle time.
- Minimum and maximum memory size.

Using the data set of 168 CPUs, along with their relative performance score, we are asked to create 2 regression algorithm that predicts performance scores based on the 6 provided features. Algorithms such as these could be useful in large scale machine comparison without needing manual benchmarks, and could also bring us a greater understanding of the relationships between physical attributes and overall performance.

Similarly, for the classification task, we are given a data set of 112 computer vision extracted features taken from images of epithelial and stromal regions of the brain. We are given a data set with 600 elements, each containing the 112 features. We are asked to develop 2 classification algorithms that, given a single set of features, will be able to classify the relevant brain region as an epithelial region or stromal region. Algorithms such as these would have great potential in the field of medicine, one of the most promising applications of machine learning techniques.

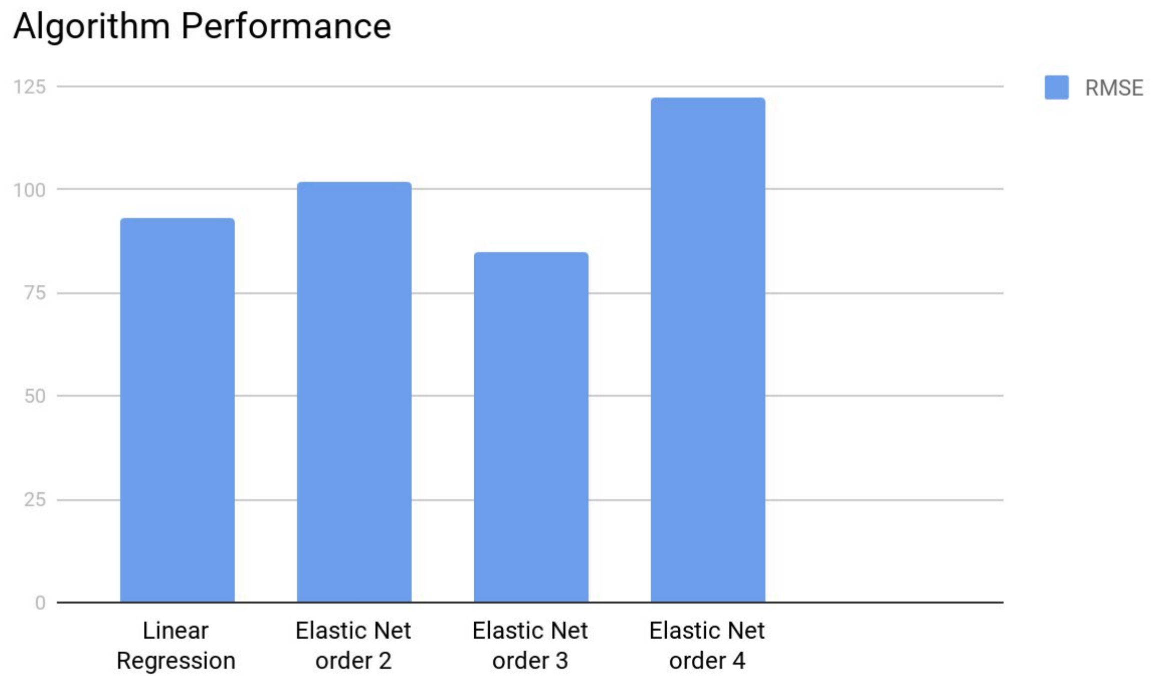
Regression Task

There are a wide variety of regression algorithms to choose from, ranging from simple linear using one independent variable, to more complex methods such as ensembling. For this task we choose to implement

1. A multivariate simple linear regression model.
2. Elastic Net regression, which linearly combines Lasso and Ridge penalties.

All algorithm code was written in Python 3.6, along with any auxiliary code that was produced. In addition to our own code, libraries “numpy”, “sklearn.preprocessing” and “sklearn.linear_model” were used to implement the algorithm.
To evaluate the performance of our algorithms, we use the root mean squared error (RMSE) defined as: $RMSE(\check{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \check{y}_i)^2}$ where y is the real answers, and $\check{y}$ is our predictions. If we train a model with the entire data set, any performance metric we get will be unrepresentative of unseen prediction, since the algorithm has already trained on the actual answers. To get a more useful performance metric, we train the model using only 87.5% of the training data (147 entries), the remaining entries are then used to predict answers.We can repeat this process 8 times using a different 12.5% for testing each time, at the end we can concatenate all the predictions made and calculate the total RMSE against the training answers.

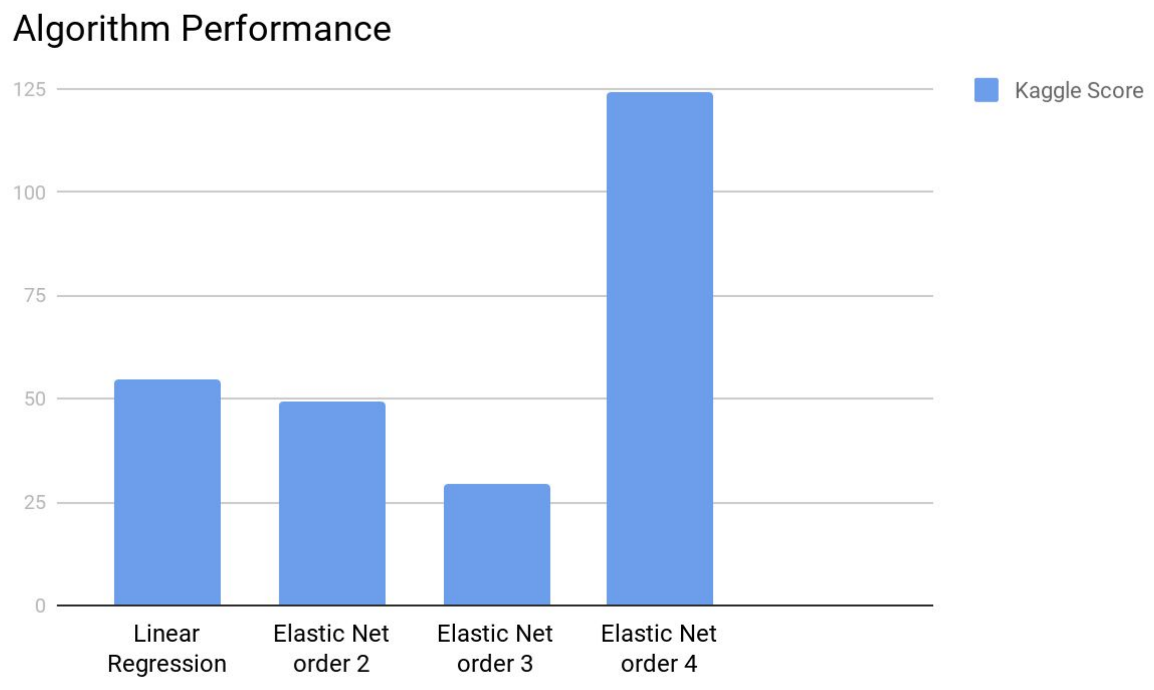
We developed and tested simple linear regression, as well as Elastic Net regression in addition to logarithmic scaling of the features using log1p ($f_{new} = logn(f_{old} + 1)$). By adding one to the feature before taking the natural log, we ensure that there are no zero-values that have an undefined natural log.



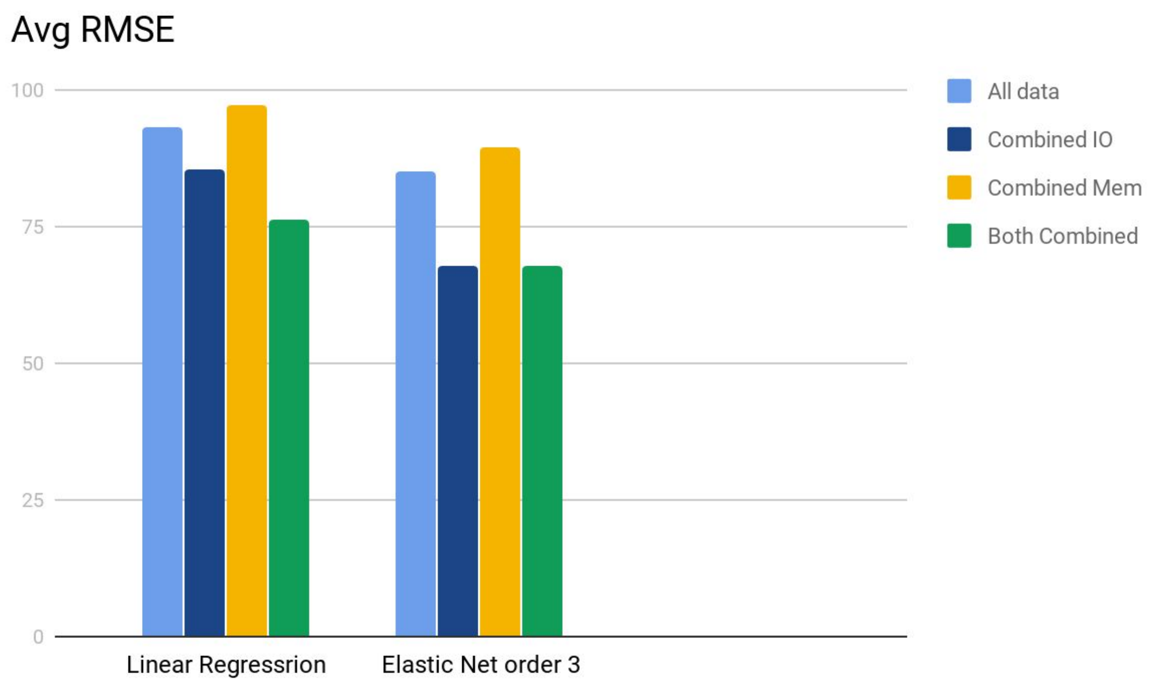
We can see that linear regression performs quite well in comparison the the more complex elastic net equivalent, however elastic net with a polynomial order of 3 seems to be best by our standards. Using this method of comparison, we fine tuned the constants “alpha” and “l1_ratio” of the elastic net implementation find the optimal configuration.

The performance metrics produced by our cross-fold validation technique were used in combination with kaggle submission results to ensure we were on the right track. We found that the relation between algorithm holds somewhat however Elastic Net regression seems to do a lot better on kaggle overall. This could be due to the fact that the algorithms submitted to kaggle were trained on the whole data set as opposed to 87.5%.

It is also worth noting that kaggle public scores are calculated on 33% of the data, so this may be unrepresentative of the actual performance.



We calculated the pairwise linear correlation coefficient between each one of the features and their p-values to test the hypothesis of no correlation against the possibility that there is a nonzero correlation as well. We discovered that features minimum and maximum memory size are significantly correlated with values of correlation coefficient 0.77 and a p-value of 1.92e-33. Another average correlation was found between Cache memory size features with numbers correlation of 0.57 and p-value 6.32e-17. We assumed linear data dependencies, so we used Pearson correlation coefficient instead of Spearman. For all these operations we used MATLAB environment.
We also defined the variance inflation factor (VIF), which measures the effect on variance of the estimated regression coefficients due to collinearity. Basically it determines how much multicollinearity (correlation between features) affects the regression analysis. Multicollinearity might be a problem, because due to variance it can make the regression coefficients unstable and not so easily determined. Especially for maximum and minimum memory features was the 3. regression coefficient considerably affected with value 3.373. After these correlation results we decided to linearly combine some of the features together in order to get better performance in our models.



Classification Task

There are many classification algorithms that are a viable choice when tackling a binary classification problem. The algorithms that we considered were K-nearest neighbours, Naive-bayes, Logistic Regression, Support Vector Machines, Random forests and gradient boosting.

Eventually we settled on logistic regression and naive-bayes as our two chosen classifiers.

All algorithms were implemented using python 3, with the use of the 3rd party libraries numpy and sklearn. In particular sklearn.linear_model and sklearn.naive_bayes.GaussianNB(although we did also implement our own version of naive-bayes). We used the sklearn library sklearn.model_selection.GridSearchCV to tune the hyperparameters for our models such as ‘C’ in the case of logistic regression. To improve results we only trained on a subset of the features, decided by the SelectKBest algorithm from sklearn.feature_selection. To improve our models performance, or in the case of naive-bayes make it work at all, we decided to scale the data to the range of [0,1] with the library sklearn.preprocess.minmaxscaler.
We used two metrics to decide what algorithms to use and tune our models, the average accuracy and area under the roc curve scores from a 5-fold cross validation process across the training set. We chose accuracy to start with as a performance metric as it was the scoring metric used for the kaggle competition. Then we decided that we would incorporate both sensitivity and specificity by using the area under the receiver operating characteristic curve, this means our algorithm should predict more false positive and less false negatives which is a good thing in regards to medicine where not diagnosing a patient could be fatal.

We noticed early on that logistic regression was achieving higher for both accuracy and area under the roc curve so we chose to our time on the logistic regression algorithm more. To choose how many of the 112 features we would use in the SelectKBest algorithm we varied the number of “best” features,used a gridsearch to tune our value for “C” each time and used the number of features that performed the best across our performance metrics. We found that a C value of 6-8 and using around 25-30 features produced the best results.

Conclusion

We are happy with our performance in the regression task, and confident that we made the correct decisions in choosing the algorithms. In retrospect, it could have been useful to develop test different methods of regression such as ensembling methods, however the final solution seems reasonable for the problem scope. It may also have been useful to measure performance using the average RMSE of all iterations instead of just taking the RMSE of all the predictions as a whole.

We are happy with our performance in the classification task, although we could have done a better job of learning about a wider array of classification algorithms earlier in the task. If we had developed more strong models with similar performance(preferably classifying different “spaces” of the problem) it would have been more viable to use the ensembling method called stacking. More time could have been spent on understanding the features given, to manually perform feature selection.