Final Project

Christian McDaniel, Jeremy Shi

CSCI 6360 Spring 2018

In this final project, we focus on the “Wine Quality Data Set” from UC Irvine’s Machine Learning Repository (<https://archive.ics.uci.edu/ml/datasets/wine+quality>). This project report covers the dataset and background, techniques, evaluation methods and our results related to this project.

**1. Background, Goal, and Data Description**

There are two datasets involved in this project -- one is for red wine and the other is for white wine, both of which are variants of Portuguese “Vinho Verde” wine. There are 6497 wines/instances in total, 4898 from the white wine dataset, and 1599 from the red wine dataset. There is no brand, price or grape type information. Features included in the datasets consist of common physicochemical measurements (all numerical, some integer; some real), which are as follows:

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

The target/output variable in the dataset is the quality score (between 0 and 10), based on sensory tests. The distribution is unbalanced as the target variable takes a Gaussian shape, and is missing scores from 0-2 or above 9. Furthermore, as taste is a notoriously difficult perception to quantify, the use of machine learning to glean meaningful nonlinear relationships between the predictors and the target variable is justified.

During exploration of the data, the histograms show that most features also follow a Gaussian distribution. There are a few features, however, that appear to have a significant left skew in their distribution, namely residual sugar, chlorides, free sulfur and sulfates. We made note of these features, hypothesising a correlation between the outliers and a resulting assignment at one end of the quality spectrum.

However, upon examination of the scatterplots of these features against the target variable, no trend in quality is apparent, and the examples containing these outliers tend to be classified in the middle of the quality bell curve. In fact, most features follow a shallow Gaussian or even uniform distribution when plotted against the quality, showing no direct relationship with a change in quality. The only feature that does seem to have a gradual positive correlation with increased quality is alcohol content of the wine. A study which uses this dataset by Cortez, et. al. purports that the importance of alcohol content in determining wine quality is a well-established trend in oenology.

A final observation from the data is noted when both datasets are compared side by side. Again plotting each feature individually against the quality, but this time as a violin plot where one side is for the red wine dataset and the other side for the white wine dataset, it is clear that a number of variables show distinct distributions for red vs. white wine. Thus, we hypothesize that while predicting quality may not yield outstanding results (Cortez et. al. reach an accuracy of 62% before allowing for the model to take the best of two prediction choices), predicting the type of wine from the physicochemical properties will prove to be an easy and well-performed task by our models.

**2. Methods**

The dataset adheres to both regression and classification techniques quite well. This is due to the nature of the response variable, which consists of ordered integer values from 0-10, ranking the wine by increasing quality. Models were run using ScalaTion, R and Python programming languages and a myriad of associated packages and libraries to be discussed further below.

**2.1 Data Preparation**

Before running each model, the data was prepared by first reading each dataset into the script and separating them into features vs. target variables. The target variable was then rescaled to 0 since neither dataset had quality scores below 3. Finally, for the neural network model performing classification, the target variables for the white and red datasets were converted into one-hot encodings per requirements in Keras.

Further preprocessing such as standardization was performed on specific training folds inside the cross validation pipeline, as discussed below.

**2.2 Parameter Selection, Standardization and Cross Validation Pipeline**

Models run in Keras and Sci-kit Learn were tested on a multitude of different parameters by constructing a dictionary with ranges of values for each parameter to test and randomly sampling from this dictionary. For models with a low to moderate number of parameters to test (under 200), an exhaustive grid search was conducted. For models with larger search spaces, at least 10% (but usually a much greater proportion) was searched. For the neural network performing classification, the parameter grids were too large to test even 10% of the possible permutations. Values used for the parameter grid can be seen in the code provided.

Once a parameter combination was selected, the parameters were provided to a 5-fold cross validation pipeline which first generated the training and testing folds, and next standardized the data in the training folds. The resulting standardization parameters (i.e., mean and standard deviation) were used to separately standardize the testing folds.

Following standardization, the training data were fed into the model for training and the testing data were subsequently used for predicting and scoring. When provided, built-in scoring metrics were used. Otherwise, prediction values were used to hand-score the performance against the ground truth values. This pipeline was largely hand-coded since using all of these classes within a built-in Pipeline, especially when using Callbacks in Keras as a part of the pipeline, proved problematic.

Table 1. Programming languages and associated packages used for the preprocessing pipeline.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Preparation | Parameter Selection | Cross-Validation | Standardization |
| ScalaTion | Relation.scala, MatriI.scala, MatriD.scala | default | PermutedVecI.scala, RNGStream.scala (rest hand-coded) | -- |
| R | lapply(dataset, as.factor) | default | folds <- createFolds(dataset$type, k = nFolds) cv<-lapply(folds, trainAndTestFold, dataset=dataset) | -- |
| Python | Pandas; LabelEncoder & LabelBinarizer (one-hot) (Sci-kit Learn) | ParameterSampler (Sci-kit Learn) | StratifiedKFold (Sci-kit Learn) (rest hand-coded) | StandardScaler (Sci-kit Learn) |

**3. Results**

The models which yielded the highest performance measures from each model are listed below.

**3.1. Classification**

For classification, Naive Bayes, TAN Bayes, LDA, Logistic Regression, Dense neural network, Linear SVM and C-SVM were trained/tested on the data and compared. ScalaTion generated the outputs for Naive Bayes, TAN Bayes, LDA and Logistic Regression using classes in the scalation.analytics package. The neural networks were constructed, trained and tested using Keras Sequential. Dropout, BatchNormalisation, EarlyStopping and CVSLogger were used. Sci-kit Learn’s SVM.SVC, SVM.linearSVC and SVM.SVR were used for the C-SVM, Linear SVM and SVM Regressor models, respectively.

Table 2.1. Classification results for the white wine dataset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Naive Bayes` | TAN Bayes` | LDA` | Logistic Regression` | Multilayer Perceptron (Dense NN)^ | Linear SVM” | C-SVM” |
| Accuracy, Recall, Precision, F1-Score | 95.0%, 100%, 95.0%, 96.667% | 10.823%, 0.0%, 0.0%, 0.0% | 89.177%, 100%, 89.177%, 94.040% | 89.177%, 100%, 89.177%, 94.040% | 53.104%,  ,  ,  53.104% | 52.106%, ,  ,  12.861% | 53.596%,  ,  ,  12.861% |
| Parameters | default | default | default | default | {n\_layers=2, units1=85, units2=45, units3=85, batch\_size=1280, use\_bias=False, krnl\_init=lecun\_normal, optimizer=’SGD’, activation=relu} | {'mult\_class': 'ovr',  'loss': 'squared\_hinge', 'C': 0.25} | {'kernel': 'rbf', 'gamma': 0.03125, 'C': 2} |

` : scalation; ^ : keras, sklearn; “ : sci-kit learn;

\* : if tp, fp, fn = 0 → recall, precis, fscore = 1; if tp = 0, fp or fn != 0 → recall, precis, fscore = 0

Table 2.2. Classification results for the red wine dataset.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TAN Bayes` | LDA` | Logistic Regression` | Multilayer Perceptron (Dense NN)^ | Linear SVM” | C-SVM” |
| Accuracy, Recall, Precision, F1-Score | 17.238%, 0.0%, 0.0%, 0.0% | 82.762%, 100%, 82.762%, 90.017% | 82.762%, 100%, 82.762%, 90.017% | 59.050%,  ,  ,  59.050% | 56.987%,  ,  ,  13.945% | 58.230%,  ,  ,  13.945% |
| Parameters | default | default | default | {n\_layers=2, units1=45, units2=45, batch\_size=50, use\_bias=Falsekernel\_init=’glorot\_uniform’, optimizer=’adam’, activationh=’relu’} | {'mult\_class': 'ovr',  'loss': 'squared\_hinge', 'C': 1} | {'kernel': 'rbf', 'gamma': 0.03125, ''C': 4} |

` : scalation; ^ : keras, sklearn; “ : sci-kit learn;

\* : if tp, fp, fn = 0 → recall, precis, fscore = 1; if tp = 0, fp or fn != 0 → recall, precis, fscore = 0

Table 2.3. Class-wise analysis from neural netwworks on the white and red datasets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASS | Top Accuracy (white) | Noteworthy Parameters (white) | Accuracy (Red) | Noteworthy Parameters (Red) |
| 0 (quality=3) | 82.857% | 5 layers (85,85,45… units), SGD opt, tanh act, no bias, zeros kernel init | 72.222%, | 2 layers (45 untis), adam opt, linear act |
| 1 (quality=4) | 32.208% | 2 layers(45,45), SGD, linear, batch\_size 4898 | 32.500% | 10 layers (10,10.45... units), adam, relu |
| 2 (quality=5) | 80.557% | 3 layers (10,45,85), adam, sigmoid, batch\_size 1280, zeros kernel init | 99.393% | 3 layers (85,85,45... ), SGD, sigmoid, zeros kernel init |
| 3 (quality=6) | 99.805% | 5 layers (45,45,10…), SGD, softmax, batch\_size 4898, zeros kernel init | 65.021% | 1 layer (45), SGD, sigmoid, 1280 batch\_size |
| 4 (quality=7) | 54.318% | 2 layers(45,45), SGD, tanh, no bias, batch\_size 4898 | 84.750% | 5 layers (85,45…), SGD, tanh, batch\_size=50 |
| 5 (quality=8) | 21.508% | 5 layers (10,45,10…), SGD, relu, no bias, batch\_size 1280 | 72.667% | 5 layers (10,45,10…), adam, tanh, no bias |
| 6 (quality=9) | 47.143% | 5 layers (45,85,10…), SGD, sigmoid | -- | -- |

**3.2. Regression**

For regression, we apply the following techniques on the white wine and red wine dataset: SVM Regression, Multi-layer Perceptron, Random Forest, Gradient Boosting, AdaBoost, Lasso Regression, Elastic Nets, Ridge Regression, and Multiple Linear Regression.

The results (5-fold cross validation) are as follows:

Table 3.1. Regression results for the red wine (all from scikit-learn).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SVM Regressor | Multi-layer Perceptron | Random Forest Regressor | Gradient Boosting | AdaBoost | Lasso Regression | Elastic Nets | Ridge Regression | Multiple Linear Regression |
| Best MAE | 0.5076 | 0.5057 | 0.5076 | 0.4991 | 0.5289 | 0.5642 | 0.5476 | 0.5101 | 0.5104 |
| Params in the best model | {'degree': 2, 'kernel': 'linear'} | {'activation': 'tanh', 'hidden\_layer\_sizes': (9000,), 'learning\_rate': 'invscaling', 'solver': 'lbfgs'} | {'max\_depth': 5, 'n\_estimators': 50} | {'max\_depth': 3, 'n\_estimators': 100} | {'n\_estimators': 50000} | {'alpha': 0.1} | {'alpha': 0.1, 'l1\_ratio': 0.3} | {'alpha': 0.1} | {'normalize': False} |

Table 3.2. Regression results for white wine (all from scikit-learn).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SVM Regressor | Multi-layer Perceptron | Random Forest Regressor | Gradient Boosting Regressor | AdaBoost Regressor | Lasso Regression | Elastic Nets | Ridge Regression | Multiple Linear Regression |
| Best MAE | 0.6528 | 0.5989 | 0.5864 | 0.5749 | 0.6006 | 0.6248 | 0.6237 | 0.5959 | 0.5938 |
| Params in the best model | {'degree': 2, 'kernel': 'linear',  0.03125, 'ep': 0.1, **'C': 1**} | {'activation': 'relu', 'hidden\_layer\_sizes': (9000,), 'learning\_rate': ‘adaptive’,  'solver': 'lbfgs'}) | {'max\_depth': 5, 'n\_estimators': 10} | {'max\_depth': 5, 'n\_estimators': 40} | {'n\_estimators': 10000} | {'alpha': 0.1} | {'alpha': 0.1, 'l1\_ratio': 0.5} | {'alpha': 1} | {'normalize': True} |

**3.3. Results from combining the red and white datasets**

To get more out of the datasets, we also tried to combine the red and white datasets together. That is, we add a new column indicating whether it is red or white wine (0 vs. 1). We can either treat this column as a new feature (in order to predict the score based on regression), or the target (classification task). Here is the regression results when we combine these datasets and use the wine category (red vs. white) as a new feature:

Table 4.1. Classification Results on combined datasets to predict wine type.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Naive Bayes` | TAN Bayes` | LDA` | LR` | KNN` | Multilayer Perceptron (Dense NN)^ | Linear SVM” | SVM Regressor” |
| Accuracy/R2 | 98.153% | 75.388% | 99.708% | 99.969% | 99.908% | 99.554% | 98.813% | 0.96639 (R2) |
| F1 Score | 0.32852 | 0.0 | 0.32840 | 0.32963 | 0.32935 | 0.99084 | 0.13020 | 8.0997 (sse) |
| Parameters | default | default | default | default | default | {n\_layers: 1, units1: 45, batch\_size: 128, use\_bias: True, bias\_initializer: ‘lecun\_normal’, kernel\_initializer: ‘lecun\_normal’, optimizer: SGD, activationh: relu} | {mult\_class: 'ovr',  'loss': 'sqrd\_hinge  'C': 0.25} | {'kernel': 'rbf', 'gamma': 0.03125, 'ep': 0, 'degree': 4, 'C': 4} |

` = R; ^ = Keras + Sci-Kit Learn; “ = Sci-Kit Learn

Table 4.2. Regression Results on combined datasets to predict wine type (all from scikit-learn)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SVM Regressor | Multi-layer Perceptron | Random Forest Regressor | Gradient Boosting | AdaBoost | Lasso Regression | Elastic Nets | Ridge Regression | Multiple Linear Regression |
| Best r2 score | < 0 | 0.33303 | 0.19273 | 0.21651 | 0.17258 | 0.17315 | 0.19218 | 0.18569 | 0.05156 |
| Params in the best model | {'kernel': 'rbf’, ‘gamma': 0.03125, 'ep': 0.1, 'C': 1} | {'activation': 'tanh', 'hidden\_layer\_sizes': (4000,), 'learning\_rate': 'adaptive', 'solver': 'lbfgs'} | {'max\_depth': 2, 'n\_estimators': 50} | {'max\_depth': 2, 'n\_estimators': 22} | {'n\_estimators': 10} | {'alpha': 0.1} | {'alpha': 0.1, 'l1\_ratio': 0.3} | {'alpha': 1} | {'normalize':False} |

**4. Discussion**

In general, here are some findings of our experiments. First, Physicochemical properties show mild-moderate correlation with quality assessment. Second, Sklearn, Keras are very user-friendly. That is where we mainly conducted our experiments, although we experienced significant time delays with both of these libraries. Third, in this task, normalizing and scaling data does make a difference, although sometimes the results would get worse, namely for specific regression models. Fourth, contrary to the original authors (according to which SVM is the best), there is no single best method for every task. In our findings, ensemble models generally performwell , although sometimes SVM or Neural networks have better performance than ensemble models. Discussion of specific results from classification, regression and the combined dataset are below.

**4.1. Classification**

As can be seen from the Results section 2, classification of the white wine dataset performed much better than that of the red wine dataset. This is likely due to the fact that the white wine dataset contained over 3 times as many examples as the red wine dataset. For the white wine dataset, Naive Bayes performed the best with 95% accuracy. However, this finding is suspect as we could only get a non-“Nan” accuracy on Naive Bayes once (and never for Naive Bayes on the red wine dataset). Thus, it may be better to look at the results excluding Naive Bayes in general, in which case both LDA and Logistic Regression performed the best. Also suspect regarding the LDA and Logistic Regression tests are that both of them produce the exact results for both the white wine dataset and the red wine dataset, and for both the Accuracy is the exact same as the Precision. We also find it odd that TAN Bayes is lower than Naive Bayes (and all the other models), and KNN did not work for these datasets. All of these models were run using ScalaTion, and there is likely an issue with processing the multiple-class target value, where multiple “one-vs-rest” iterations are needed. Unfortunately due to timing, digging into these issues is beyond the scope of this project.

We see the neural networks performing only slightly higher than a random guess (50%). This is possibly due to the immense size of the parameter space and a lack of time to explore this space. Going forward, the options should be reduced and a genetic algorithm should be used to search the parameter space; however, obtaining justification for which parameters to remove and implementing a genetic algorithm are outside the scope of this project. Also suspect is that the Accuracy and F1 Score are the same for the neural networks. These statistics were hand-coded and summed across multiple classes. Each class would have different Accuracy and F-Score, but once summed, they were always the same.

SVM models outperformed the neural networks, and had much smaller parameter spaces to explore. Interestingly, the F-scores for SVC and SVR turned out to be the same for both datasets. The code was double checked to ensure correctness, and this may reflect some characteristic of the data that is preserved across the models.

Finally, class-wise analysis was conducted from the predictions made by the neural network. We see distinct differences in the optimal networks and performances for each class. Ideas for dealing with this are discussed in the Future Works section.

**4.2. Regression**

As we can see, the best models are Gradient Boosting, Multi-layer Perceptron, SVM and Random Forest. According to the original paper (Cortez et. al), the best MAD score for the red wine is 0.46 by SVM. However, in our experiment, SVM doesn’t render such a low score. Only the Gradient Boosting comes close. Multiple Regression is similar with the one the author provides (0.51 vs. 0.50), and our Multilayer perceptron performs a little better than the Neural Network model they provide (0.505 vs. 0.51).  So it is not clear how the authors got the results in SVM. But we did discover that Gradient Boosting is an exceptional way to do this task when comparing to other models.

In the white wine dataset, similarly, we still do not know how the author gets the best MAD score of 0.45 by using SVM. Our best SVM model still has a MAD score of 0.65. But our Multiple Regression and the Multilayer Perceptron share similar performances with the author’s (0.60 vs. 0.59, 0.60 vs. 0.58, respectively.) No surprise, Gradient Boosting performs the best among our models and carries a MAD score of 0.588.

We have also tried using Keras to build a deep neural network on this problem. The best architecture is as follows:

def new2\_advanced\_hidden\_model():

model = Sequential()

model.add(Dense(12, activation='relu', input\_dim=11))

model.add(Dense(19, activation='relu', input\_dim=12))

model.add(Dense(13, activation='relu', input\_dim=19))

model.add(Dense(10, activation='relu', input\_dim=13))

model.add(Dense(7, activation='relu', input\_dim=10))

model.add(Dense(1, activation='linear', input\_dim=7))

model.compile(loss='mean\_squared\_error', optimizer='adam')

return model

As we can see, the network has 6 layers with the adam optimizer, and each hidden layer uses the relu activation function. This network renders remarkable results. For the white wine dataset, in a 5-fold cross validation with a batch size of 140 and 700 epochs, the network gives a MAE score of 0.5810, which is almost as good as the Gradient Boosting result above. For the white wine dataset,  in a 5-fold cross validation with a batch size of 140 and 700 epochs, the network gives a MAE score of 0.5187, which is not as exceptional as the one from the white dataset. Still, neither is not as good as the best SVM score the original authors have provided. But its performances are comparable, if not better, other methods.

**4.3. Combined dataset**

Our initial hypothesis that the models would perform much better for classification tasks on this dataset proved correct. We see an overall improvement of 32.644% across all models from the performance on the white wine dataset alone, and a 36.423% improvement for the red wine dataset. This finding is to be expected, since the problem is made much easier - reduced from predicting one out of six (for white) or five (for red) classes for which no one predictor linearly predicts, to a problem of predicting one of only two classes for which the data are well-separated. Furthermore, all models that previously did not work the multi-class data produced output for the binary combined dataset.

As we can see, when we combine the white and red datasets together and use regression models to predict the wine quality, the performance drops a lot. The only noticeable good model is multi-layer perceptron. On the contrary, SVM doesn’t even given a positive result. Among other models, gradient boosting still performance fine. This makes sense since we have reduced the target output to two binary numbers.

**5. Future Work**

There are still a lot of implementations we could try beyond the current status of the project. We can explore parallel neural network and capsule networks to specify parameters to each class. We can also contribute to open source by updating ScalaTion examples with this dataset. Last but not least, we can also assess time complexities in our training in comparison to the original author’s results. However, the paper is from 9 years ago and it is hard to conduct a comparison given the current software and hardware specs.

**5.1 Post hoc Analysis**

In a post-hoc experiment following our presentation, we aimed to match the second performance measure used by Cortez et. al. Initially, as can be seen in section 2.4, we used the MAD (i.e., mean absolute error) scores we produced to compare the performance in the paper. However, the paper also uses an Accuracy measure, for which they use a rounding scheme to produce integer classes from their regression outputs and determine accuracy on these derived class predictions. The rounding scheme involves a Tolerance (T) parameter. Of note, the paper uses T=0.5 to represent rounding the predicted regression output to its nearest integer value (e.g., 1.2 rounds to 1, 5.5 rounds to 6, and 7.8 rounds to 8). Additionally, a tolerance of T=1.0 is used, for which the regression output may assume the integer produced by either its floor function or its ceiling function. This allowed for much more wiggle room in the rounding scheme and produced the highest results of the paper - 89.0% for white and 86.5% for red. No information was given regarding the formula used for multiclass accuracy, so we summed the tp, fp, tn and fn across each class for each fold to compute a fold-wise summed accuracy ( (tp+tn) / (tp+tn+fp+fn) ). Class-wise accuracy and precision were also computed and stored. The results from our highest-performing post hoc SVM-C models for tolerance T=0.5 are as follows, using accuracy as the ranking performance measure:

Table 5.1. White wine:

Table 5.1.1. Overall Performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | MAE | R2 | Parameters |
| T=0.5 | 85.904% | 50.664% | 0.59958 | 24.299% | Kernel: ‘linear’, gamma: 0.3125, ep: 1, C: 0.25 |

Table 5.1.2. Class-wise Performance (same models as above)

|  |  |
| --- | --- |
|  | Precision (T=0.5) |
| Class 0 | 0.0% |
| Class 1 | 40.00% |
| Class 2 | 62.650% |
| Class 3 | 49.570% |
| Class 4 | 47.893% |
| Class 5 | 0.0% |
| Class 6 | 0.0% |

Table 5.2. Red wine:

Table 5.2.1. Overall Performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | MAE | R2 | Parameters |
| T=0.5 | 86.392% | 59.178% | 0.50936 | 31.500% | Kernel: linear, gamma: 3.05176, ep: 0, C: 2 |

Table 5.2.2. Class-wise Performance (same models as above)

|  |  |
| --- | --- |
|  | Precision (T=0.5) |
| Class 0 | 0.0% |
| Class 1 | 0.0% |
| Class 2 | 66.559% |
| Class 3 | 53.119% |
| Class 4 | 54.742% |
| Class 5 | 0.0% |

For both datasets, the accuracy at tolerance T=0.5 rivaled that obtained by the much more lenient tolerance of T=1.0 used by Cortez et. al. (and far exceeded the accuracies obtained by Cortez et. al. with tolerance T=0.5 on SVM - 62.4% for white and 64.6% for red). The class-wise precisions follow the Gaussian trend observed by Cortez et. al. as well.

Given the promising results, we combined the datasets again but this time used the quality as the target variable. This retains the difficult and non-linear multiclass problem from the original separate datasets but provides more data. Although other parameter combinations were tested, the default parameters from Sci-kit Learn performed the best. The scores are similar to the separated datasets. The full results are as follows:

Table 5.3. Combined datasets (Target = Quality)

Table 5.3.1. Overall Performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | MAE | R2 | Parameters |
| T=0.5 | 85.768% | 50.189% | 0.60851 | 18.465% | Sci-kit Learn default paramaters |

Table 5.3.2. Class-wise Performance (same models as above)

|  |  |
| --- | --- |
|  | Precision (T=0.5) |
| Class 0 | 0.0% |
| Class 1 | 40.500% |
| Class 2 | 56.408% |
| Class 3 | 50.799% |
| Class 4 | 41.878% |
| Class 5 | 4.7059% |
| Class 6 | 0.0% |

Furthermore and most importantly, with MAE and R2 scores that are comparable to those obtained by our previous best-performing models, we feel that this bridges the gap so as to allow comparison of our other models with the results from Cortez et. al.

**6. References**

Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

Cortez, Paulo, et al. "Modeling wine preferences by data mining from physicochemical properties." *Decision Support Systems* 47.4 (2009): 547-553.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.