University of the Witwatersrand

School of Computer Science and Computational and Applied Mathematics

Machine Learning

COMS 3007 Assignment

Author:
Bhekamandlenkosi Khanyile *(1421169)*,
Ngoako Mowasa *(1442930)*,
Sandile Khendle *(1490320)*

Lecturer:
Dr. Benjamin Rosman

May 20, 2019

1 Purpose

In this assignment we are required to find a data set and apply two supervised learning classification algorithms, implemented from scratch. We have elected to implement Naive Bayes and Decision Trees. We were also looking to do some regression and probably some visualization; both as a challenge and for prospects of extra marks. We chose Decision Trees precisely because it can be used for classification and regression, and Naive Bayes because... well we just like it.

2 Data Set

Having chosen two algorithms, we headed over to the UCL Machine Learning repository for a data set that could be used for the aforementioned tasks and found the red wine quality data set. This dataset is related to the red variant of the Vinho Verde from the north of Portugal. It has 1599 data points, there are eleven(11) physio-chemical input and one sensory output, making it twelve(12) attributes in total.

Pictured below is a sample of our data set.

```
In [11]: sampleDF = pd.read_csv('winequality-red.csv', header=0)
         print(sampleDF[1:4])
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58

   alcohol  quality
1      9.8        5
2      9.8        5
3      9.8        6
```

3 Pre-Processing Techniques

Our attributes had quite long names, with no clear naming convention as seen in the picture above. For purposes of convenience, we chose to rename them using camel case convention.

Rescaling v Binarizing v Standardizing

We read about and explore three(3) pre-processing techniques on our dataset. Binarizing data transforms our data to 0 or 1 with respect to some binary threshold. This definitely cannot work with since our data has three(3) classes. Decision Trees don't require heavy pre-processing and results are expected be consistent for any method used. For Decision Trees we used Rescaling and Standardizing to compare and visualize results.

Standardizing any data transforms all attribute values such that they have a mean of 0 and standard deviation of 1. An example of our standardized data below.

```
# Standardizing
standardizer = StandardScaler().fit(X)
standardX = standardizer.transform(X)
np.set_printoptions(precision=3)

print(standardX[0:2,:])
```

```
[[-0.528  0.962 -1.391 -0.453 -0.244 -0.466 -0.379  0.558  1.289 -0.579
  -0.96 ]
 [-0.299  1.967 -1.391  0.043  0.224  0.873  0.624  0.028 -0.72   0.129
  -0.585]]
```

We decided against using standardized data on both Naive Bayes and Decision Trees since some attributes take negative values, or those that are greater than 1. Rescaling gives values that range between 0 and 1, which works perfectly since Naive Bayes works with probabilities($0 \leq P(X) \leq 1$)

The data was split using both the built-in method and one written by ourselves. All these pre-processing techniques were performed on the same data with the scikit learn library.

Experiments

After implementing the algorithms, we ran experiments on our data.

Naive Bayes Experiment

On our first tries with Naive Bayes, we had very low accuracy, struggling to go above 68% for most of our experiments using a training size of 60%. We then proceeded into divide our labels to three classes: bad, average, and good, represented as -1, 0, and 1 respectively. From then we observed an improvement in prediction accuracy. Notably, the accuracy decreased as the training size increased.

Training Size	Avg. Accuracy
0.3	72.59%
0.4	76.2%
0.6	62.8%
0.7	53.83%

A summary of Naive Bayes experiments

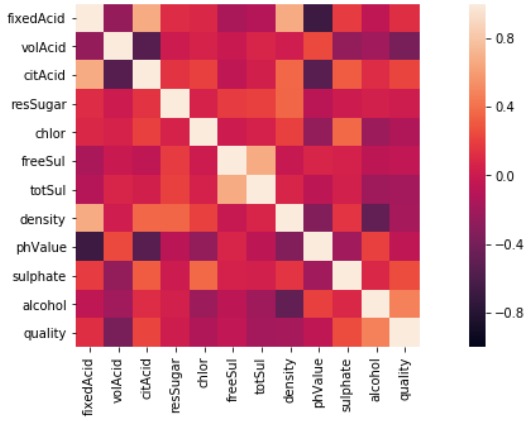
Decision Trees

We also employed the same strategy of dividing our target to three classes on Decision Trees. The average accuracy was not affected by the change in training size, the average run-time was 141.34 as shown below.

Training Size	Avg. Accuracy	Run Time
0.5	58.062	141.34

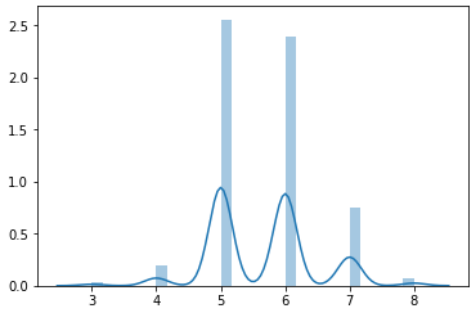
A summary of Decision Trees experiments

Some Interesting Figures



Feature Heatmap

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f11c6c4b860>
```

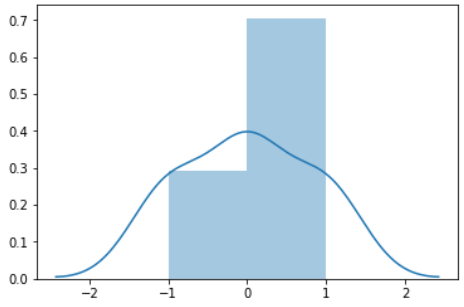


Target Distribution

Target Distribution

```
sns.distplot(Y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d72355fd0>
```



References

1. Towards Data Science: Principal Component Analysis
2. Edureka: Naive Bayes Classifier in Python
3. Google Developers: Decision Tree Classifier from Scratch
4. Geeks for Geeks: Data Pre-Processing
5. Towards Data Science: Guide to Decision Trees