ISE 364 Project

Jia Wo jiw416@lehigh.edu

Ming Zeng miz318@lehigh.edu

Ethan Li sil618@lehigh.edu

Professor: Dr. Louis Plebani

Lehigh University

Industrial and Systems Engineering

December 2019

1. Introduction

The objective of this project is to use the historical data-data.csv to build a classification model that can predict the feature-small or large in future.csv file and interpret the results to find the best model for prediction.

2. The summary

We built six models. The decision tree model with up-sampling in PySpark has the best result. The testing accuracy rate is 89.53%.

3. Data Preparation

The data.csv file contains 15 columns and 32,561 entries. There are 6 numerical columns and 9 categorical columns. The target column has two categories which are “large” and “small”. The historical data does not contain any description, so it is hard to understand what each column means. Column 1, 6, and 13 have missing values. Transforming categorical values to numerical values, drop missing values and normalize data are the three approaches applied in the data preparation process.

3.1 Transform categorical data

In order to convert categorical text data into model-understandable numerical data, we used LabelEncoder and OneHotEncoder to transform categorical attributes into a more representative numerical format. For the label, “1” stands for “SMALL” and “0” stands for “LARGE”.

3.2 Drop missing values

The missing values consists of 7% of the total data body, we moved any entries with missing values. We considered replacing the missing values with mode since all the missing values are categorical data. However, the imputation changed the distribution of the original dataset. In our case, the amount of missing data is trivial, we decided to drop them directly.

3.3 Normalize data

We use Min-Max Scaler to normalize numeric data since normalization improves the performance of the KNN and Neural Network models.

3.4 Feature selection

We used feature importance to evaluate features. We keep the top ten features to prevent overfitting, improve accuracy and reduce training time.

4. Data Visualization

4.1 Percentage of Target Variable

The percentage of "SMALL" in our dataset is 75.11% while the percentage of "LARGE" is only 24.89%. Therefore, imbalanced issue may be considered later in our models.

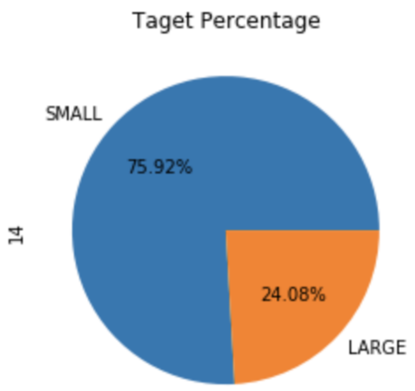


Figure 1 Percentage of Target Variable

4.2 Visualizing the categorical Columns

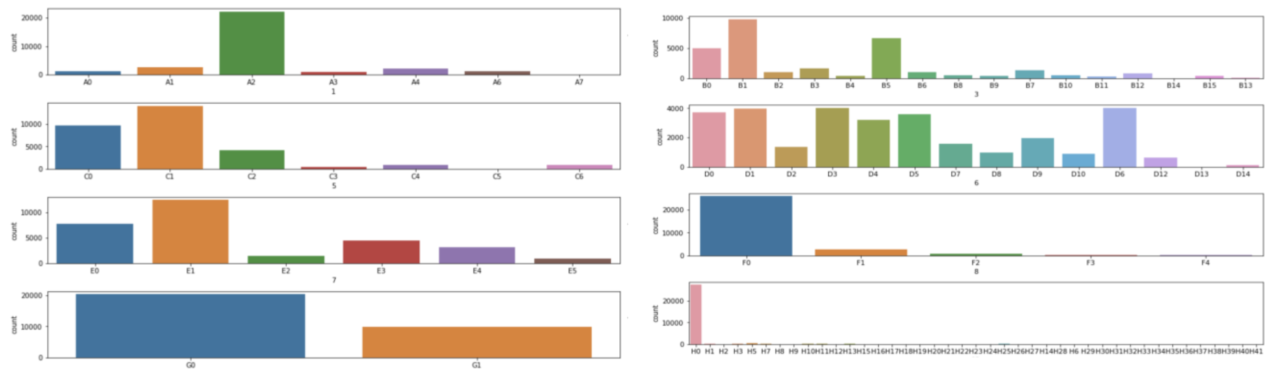
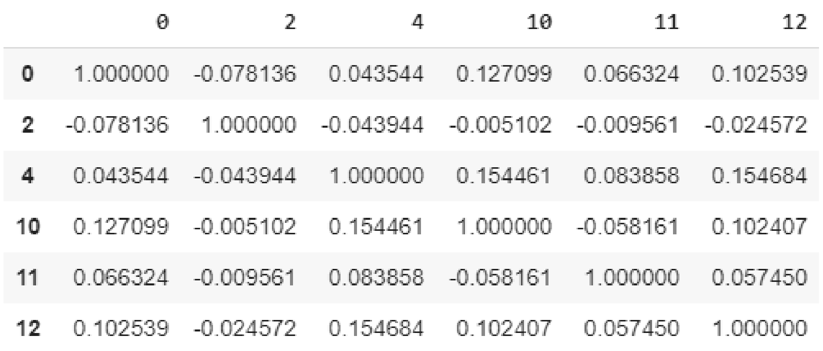


Figure2 Countplots of categorical data

From Figure 2, we can have a brief understanding of the distribution of the categorical attributes.

4.3 Visualize the Correlation between numerical data



	0	2	4	10	11	12
0	1.000000	-0.078136	0.043544	0.127099	0.066324	0.102539
2	-0.078136	1.000000	-0.043944	-0.005102	-0.009561	-0.024572
4	0.043544	-0.043944	1.000000	0.154461	0.083858	0.154684
10	0.127099	-0.005102	0.154461	1.000000	-0.058161	0.102407
11	0.066324	-0.009561	0.083858	-0.058161	1.000000	0.057450
12	0.102539	-0.024572	0.154684	0.102407	0.057450	1.000000

Figure 3 Correlation between numerical data

From Figure 3, we can have a brief understanding of each two of the numerical attributes.

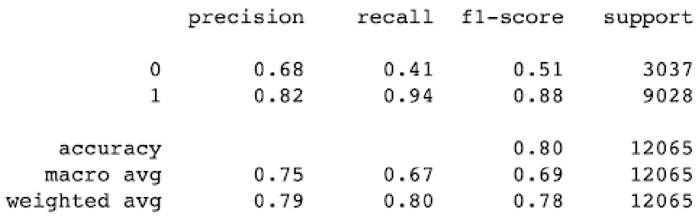
5. Modelling

5.1 Summary

We built six classification models. They are Logistic Regression, KNN, Decision Trees(with upsampling), K Means Clustering, SVM and Neural Networks.

5.2 Logistic Regression

We tried logistic regression first as our target value is a categorical variable with 2 classes. We used all the independent variables. Below is the confusion matrix. We think that the logistic regression might not be the best model because the accuracy rate is not very high and the precision is low.

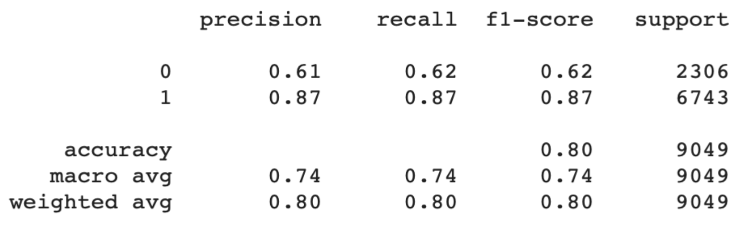


```
              precision    recall  f1-score   support

           0       0.68      0.41      0.51      3037
           1       0.82      0.94      0.88      9028

    accuracy                           0.80     12065
   macro avg       0.75      0.67      0.69     12065
weighted avg       0.79      0.80      0.78     12065
```

Figure 4 Classification report for Logistic Regression model

5.3 Decision Trees

5.3.1 Original Decision Tree



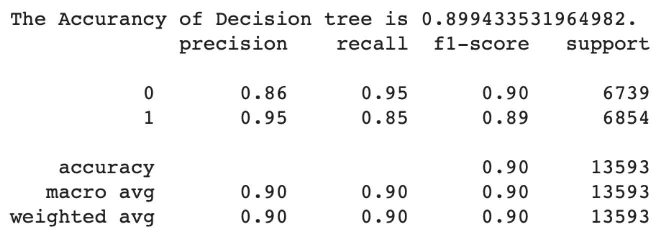
```
              precision    recall  f1-score   support

           0       0.61      0.62      0.62      2306
           1       0.87      0.87      0.87      6743

    accuracy                           0.80      9049
   macro avg       0.74      0.74      0.74      9049
weighted avg       0.80      0.80      0.80      9049
```

Figure 5 Classification report for original Decision Tree model

By using the original data, the test accuracy we could get from the decision tree model is 0.80. But as we mentioned before, the percentage of “SMALL” is much higher than “LARGE”, in this way, our model is more likely to predict the new observation as “SMALL”. Therefore, we decided to run a decision tree model with balanced labels.

5.3.2 Decision Tree- Upsampling

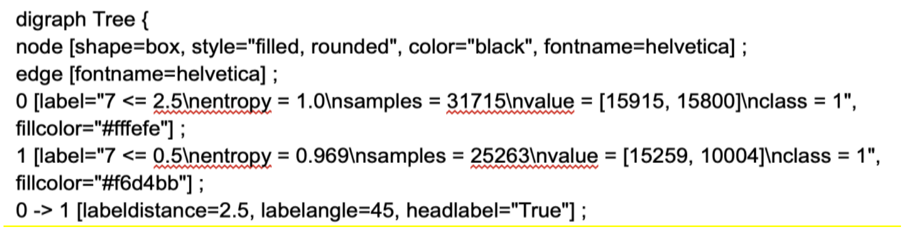
There are many solutions to handle imbalanced data. In this project, we used upsampling. We increased the number of minority class members in the dataset. The accuracy of the decision tree has been improved from 0.8 to 0.9.



```
The Accurancy of Decision tree is 0.899433531964982.
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      6739
           1       0.95      0.85      0.89      6854

    accuracy                           0.90     13593
   macro avg       0.90      0.90      0.90     13593
weighted avg       0.90      0.90      0.90     13593
```

Figure 6 Classification report for original Decision Tree model with upsampling

The tree has been stored in the file: <<decision tree in sklearn.doc>>. The graph below is only a part of our tree. We can see that the sklearn decision mistakenly regard the categorical attributes as continuous variables. For example, the variable 7 is a categorical variable but the tree split the node when variable 7<=2.5. Therefore, we used PySpark decision tree classifier to deal with it.

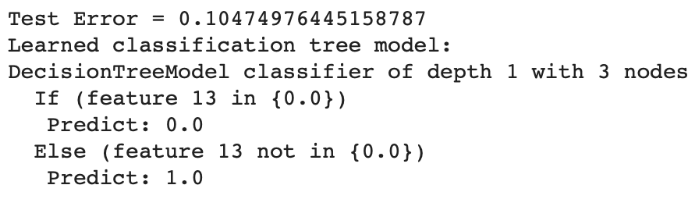


```
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
edge [fontname=helvetica] ;
0 [label="7 <= 2.5\nentropy = 1.0\nsamples = 31715\nvalue = [15915, 15800]\nclass = 1",
fillcolor="#fffefe"] ;
1 [label="7 <= 0.5\nentropy = 0.969\nsamples = 25263\nvalue = [15259, 10004]\nclass = 1",
fillcolor="#f6d4bb"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
```

Figure 7 Part of original Decision Tree model with upsampling

5.3.3 Decision tree with categorical variables

We transferred our data into LabeledPoint (label, features). And then we specified the categorical variables by using: categoricalFeaturesInfo = {1:7,3:16,5:7,6:14,7:6,8:5,9:2,13:41}.



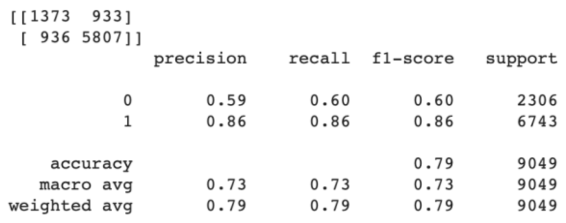
```
Test Error = 0.10474976445158787
Learned classification tree model:
DecisionTreeModel classifier of depth 1 with 3 nodes
  If (feature 13 in {0.0})
   Predict: 0.0
  Else (feature 13 not in {0.0})
   Predict: 1.0
```

Figure 8 PySpark Decision Tree model with upsampling

All categorical attributes are not treated as continuous now. The test accuracy is 0.89, and the decision tree is very easy to interpret. If feature 13 is 0.0, then we predict the label as “LARGE”, otherwise, we predict the label as “SMALL”.

5.4 K Nearest Neighbors

As KNN naturally deal with the numeric data, we first use the oneHotEncoder to transfer the categorical data into model-understandable numerical data. Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. We normalized our numeric columns (0,2,4,10,11,12,14). We then combined the numeric columns and oneHotEncoder vector into an array. We can see that, when we choose k=1, then the confusion matrix and classification_report is as below. The test accuracy is 0.79.



```
[[1373  933]
 [ 936 5807]]
              precision    recall  f1-score   support

           0       0.59      0.60      0.60      2306
           1       0.86      0.86      0.86      6743

    accuracy                           0.79      9049
   macro avg       0.73      0.73      0.73      9049
weighted avg       0.79      0.79      0.79      9049
```

Figure 9 Classification report for K Nearest Neighbors model

By using the elbow method, we find the optimal k is 5.

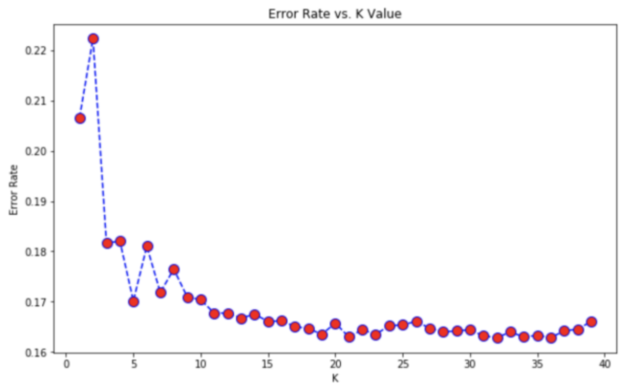
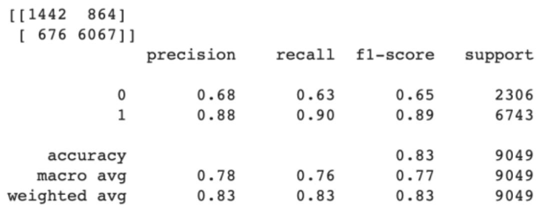


Figure 10 Elbow method for K Nearest Neighbors model

As we seen in the confusion matrix and classification_report. The test accuracy has increased to 0.83.



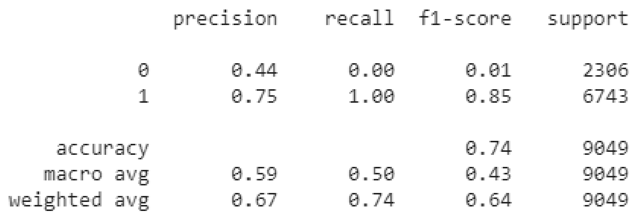
```
[[1442  864]
 [ 676 6067]]
              precision    recall  f1-score   support

           0       0.68      0.63      0.65      2306
           1       0.88      0.90      0.89      6743

    accuracy                           0.83      9049
   macro avg       0.78      0.76      0.77      9049
weighted avg       0.83      0.83      0.83      9049
```

Figure 11 Classification report for K Nearest Neighbors model using Elbow method

5.5 Support Vector Machine (SVM)

5.5.1 Original SVM

By using the original data, the test accuracy we could get from SVM is 0.74 which is not very desirable. The F score is almost 0. We can also see from the confusion matrix that everything is predicted everything belong to 1, which is "SMALL". The bias variance trade-off is clearly shown in this case.

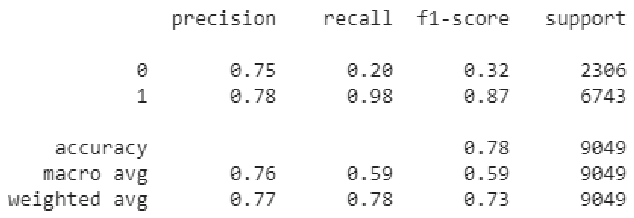


```
              precision    recall  f1-score   support

           0       0.44      0.00      0.01      2306
           1       0.75      1.00      0.85      6743

    accuracy                           0.74      9049
   macro avg       0.59      0.50      0.43      9049
weighted avg       0.67      0.74      0.64      9049
```

Figure 12 Classification report for SVM

5.5.2 SVM using Grid Search

We need to adjust the parameters to reduce bias, meanwhile, normalize the data. To do that, we implement grid search in our model to find the best parameter. From the output of the grid search, we find that the best parameters are{C:1; gamma:0.001}. We rerun the SVM model, and the accuracy rate actually improved to 0.78.



```
              precision    recall  f1-score   support

           0       0.75      0.20      0.32      2306
           1       0.78      0.98      0.87      6743

    accuracy                           0.78      9049
   macro avg       0.76      0.59      0.59      9049
weighted avg       0.77      0.78      0.73      9049
```

Figure 13 Classification report for SVM with Grid Search

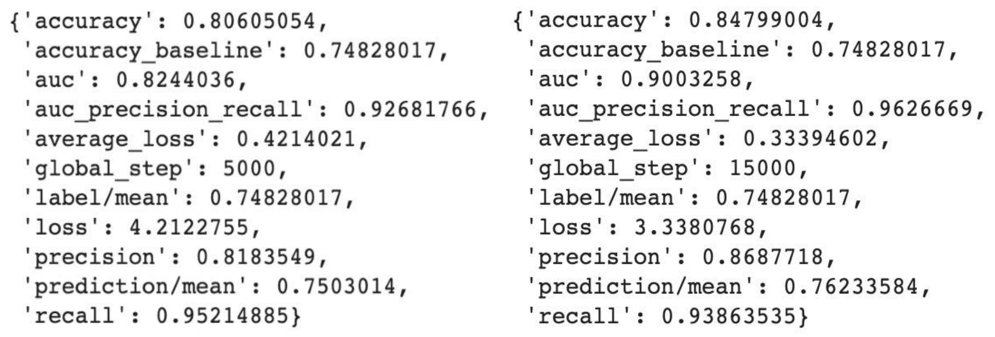
5.6 K Means Clustering

We used clustering to identify the structure of the data. We used 2 as the number of the cluster as the label suggested. The clustering model generates a new column "Cluster" and classified the data into two clusters. But we cannot relate the clusters to the output "SMALL" and "LARGE". Thus, we did not take this model into consideration. We simply used K Means Clustering to further explore the data structures.

5.7 TF-Classification & DNN Classifier

First, we used the data that all categorical variables have been transformed to numerical variables. Then we used formula: (x-x.min())/(x.max()-x.min()) to normalize all the numerical variables. After this we set all the independent variables to feature columns and used TF-Classification model. We trained our model 5000 times and the accuracy is shown below left.

For the next step, we used DNN Classifier putting all the variables as input nodes and setting hidden units as 10, 10, 10. The result is shown below right and the accuracy rate increased from 80.6% to 84.8%.



```
{'accuracy': 0.80605054,
 'accuracy_baseline': 0.74828017,
 'auc': 0.8244036,
 'auc_precision_recall': 0.92681766,
 'average_loss': 0.4214021,
 'global_step': 5000,
 'label/mean': 0.74828017,
 'loss': 4.2122755,
 'precision': 0.8183549,
 'prediction/mean': 0.7503014,
 'recall': 0.95214885}
```

```
{'accuracy': 0.84799004,
 'accuracy_baseline': 0.74828017,
 'auc': 0.9003258,
 'auc_precision_recall': 0.9626669,
 'average_loss': 0.33394602,
 'global_step': 15000,
 'label/mean': 0.74828017,
 'loss': 3.3380768,
 'precision': 0.8687718,
 'prediction/mean': 0.76233584,
 'recall': 0.93863535}
```

Figure 14 Classification report for DNN

6 Conclusions

The Decision Tree with upsampling in PySpark gives the highest test accuracy (0.89). And the prediction result for our future.csv has been stored in the file "*future_predictions.csv*". From figure 15, the percentage of the predicted "SMALL" is 7.65%, while the majority of the observations are predicted as "LARGE".

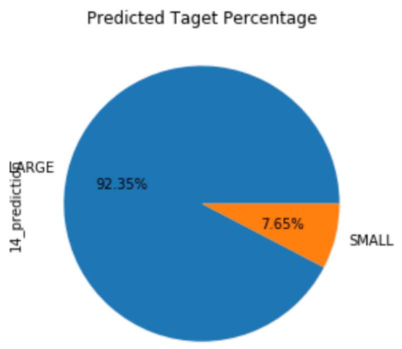


Figure 15 Predicted Target Percentage

7. Recommendations

With detailed data description, we can get a better understanding of each column. We can also impute the missing values if we obtained the details of the data. Therefore, our model will have a practical usage in real world scenario.

The accuracy of the model can be improved by cross-validation and ensemble models.

8.Reference

[1] Cios, K. (1995). Machine learning — Neural networks, genetic algorithms, and fuzzy systems. *Neurocomputing*, 8(2), pp.223-224.

[2] James, G., Witten, D., Hastie, T. and Tibshirani, R. (n.d.). *An introduction to statistical learning*.

[3] Medium. (2019). *Methods for Dealing with Imbalanced Data*. [online] Available at: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18 [Accessed 12 Dec. 2019].

[4]Pandey, A. and Jain, A. (2017). Comparative Analysis of KNN Algorithm using Various Normalization Techniques. *International Journal of Computer Network and Information Security*, 9(11), pp.36-42.

[5] Medium. (2019). *TensorFlow : DNNClassifier*. [online] Available at: https://medium.com/datadriveninvestor/tensorflow-dnnclassifier-4e68df3df00 [Accessed 12 Dec. 2019].