Prediction of Red wine quality using Support Vector Machines, K-Nearest Neighbor, Random Forest

Gadupudi Manvitha

*Student of Data science*

*Coventry University*

Coventry, England

[gadupudim@uni.coventry.ac.uk](mailto:gadupudim@uni.coventry.ac.uk)

***Abstract*—This paper looks at how to predict the quality of Red wine by applying Support Vector Machines, K-Nearest Neighbor and Random Forest machine learning algorithms.** **In this study, I have used one wine quality datasets which is red wine . To evaluate the feature importance I have used the Variance Inflation Factor and performance measurement matrices such as accuracy, recall, precision, and f1 score for comparison of the machine learning algorithm. A hyper parameter tuning algorithm was applied to improve the model accuracy. Multiple data preparation techniques were implemented removing the duplicates, and data balancing. Techniques and algorithms were implemented using Python programing language utilizing machine learning repositories, prediction results and algorithm performance measures were obtained, and visualized for comparison and discussion.**

***Keywords—machine learning; python; over sampling; EDA , metrics, random forest, k- nearest, support vector machines, hyper parameter tuning.***

1. Introduction

The quality of wine holds significant importance for both consumers and the manufacturing industries. Product quality certification is being utilised by industries to enhance their sales. In contemporary times, wine has become a ubiquitous beverage across the globe. Traditionally, the evaluation of product quality was conducted post-production, a procedure that is both resource-intensive and time-consuming. This approach necessitates the involvement of multiple human experts to assess product quality, resulting in significant costs. Assessing the quality of wine based on human experts is a challenging task due to the subjective nature of individual opinions. The dataset pertaining to the quality of wine is accessible to the public through the UCI machine learning repository, as documented by Cortez et al. in 2009. The dataset comprises two distinct files, one pertaining to the red wine variant and the other to the white wine variant of the renowned Portuguese "Vinho Verde" wine.

1. The Data Set

The repository comprises an extensive assemblage of datasets that have been utilized within the machine learning community. The dataset pertaining to red wine comprises a total of 1599 instances, The red wine datasets consist of 11 input variables and 1 output variable. The input features are derived from physicochemical assessments, while the output variable is based on sensory data that has been categorized into 11 quality classes, ranging from 0 to 10, where 0 represents very poor quality and 10 represents excellent quality.

TABLE 1. DATASET Attributes information

Input variables (based on physicochemical tests):

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

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

1. Problem

I have considered two aspects of the problems in this dataset. The first one is the study of the importance of the features for the prediction of wine quality. The secondly, performance of the prediction model can be improved using a data balancing and hyper parameter tuning with other ordinary classifiers.

IV. Data Preparation

1. *Data balancing*
2. *Unbalanced Data*

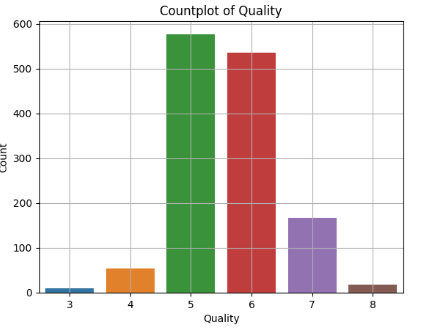
Visualize the quality class label in the red wine and white wine dataset as follows: 

Figure:1

The data presented in Figure 1 indicates the quality classification of the red wine dataset, revealing a distribution where the predominant value is 5. The quality classes of the red wine dataset range from 3 to 8. The datasets exhibit an imbalanced distribution of red and white wine, with unequal representation of the separate classes. The presence of imbalanced data has the potential to cause both overfitting and underfitting of algorithms. There are 681 instances of the highest quality class of red wine. The two datasets exhibit an imbalance in their distribution of instances, with the minority class containing as few as 5 observations and the red wine class containing as many as 681 observations. attained by achieving the utmost level of excellence.

To overcome this the below techniques are used. For source code see link- In [41]

1. *Near Miss Under-sampling*

The implementation of Near Miss under-sampling technique can potentially enhance run time efficiency by reducing the size of the training dataset, thereby mitigating memory constrains. This paper presents an analysis of the class distribution of the target variable the minimum value ranges from 10 to 577 prior to applying the techniques. However, after applying the techniques, the class distribution becomes uniform with a minimum value of 10 across all variables. The source code link for this analysis can be found in link- In[581].

1. *SMOTE Over-sampling*

One effective approach for addressing imbalanced datasets is to utilize the supervised synthetic minority oversampling technique (SMOTE) filter, as proposed by Chawla in 2005. The Synthetic Minority Over-sampling Technique (SMOTE) is a method of over-sampling that addresses class imbalance in the training set by generating new samples from the underrepresented class. Consequently, the SMOTE technique was employed to address the issue of imbalanced data.

A picture containing text, screenshot, rectangle, line

Description automatically generated

Figure:2

After applying the SMOTE technique to balance the dataset as shown in Figure 2, the default and non-default amount of instances are the same, that is 577 instances in the red wine. for source code link-In[576]

1. Data Standardization

Scikit-learn is a Python module that incorporates the latest machine learning algorithms for both supervised and unsupervised problems, as described by Pedregosa et al. in 2011.The technique of data standardisation involves scaling the features within the range of 0 and 1. This technique is beneficial for model learning. It involves applying the technique to all the numeric features and then segregating the data by standard deviation, as suggested by Pedregosa et al. in 2011. This technique is utilised for the purpose of standardising the data.The standardisation formula is expressed as follows:

zi = (xi - u)

σ

where σ represents the standard deviation, xi denotes each value, and u is the mean value of the x array.

C. Variance Inflation Factor

The Variance Inflation Factor (VIF) is a statistical tool utilised to identify the existence of multicollinearity. The Variance Inflation Factors (VIF) metric quantifies the degree to which the variance of the estimated regression coefficients is amplified in the presence of linearly related predictor variables. This paper involves the exclusion of variables with a VIF score exceeding 6, specifically fixed acidity. Consequently, the number of attributes has been reduced from 11 to 10.To view the code, please refer to the link labelled "In[571]".

paralleled to the middle classes. By using resampling this problem can be solved, the resampling is by adding copies of examples from the under-represented class of unnaturally creating such instances (over-sampling) or either by removing from the overrepresented class (under-sampling). Mostly, it will be better to over-sample unless you have sufficiently of data. However, there are some disadvantages to over-sampling it increases the instances of the dataset, so the processing time is increasing to build the model. Oversampling can lead to overfitting when putting the extremes (Drummond and Holte, 2003). Therefore, the resampling is preferred.

A good way to deal with the imbalanced datasets by applying the supervised synthetic minority oversampling technique (SMOTE) filter (Chawla, 2005). SMOTE is an over-sampling technique in which a lesser amount of classes in the training set is over-sampled and creating the new sample form to relieve the class imbalance.Therefore, to solve the data imbalanced problem we used the SMOTE technique.dataset. Code for importing data set and plotting class imbalance see appendix [IX.A.2)](#_bookmark26)

IV. MACHINE LEARNING CLASSIFICATION TECHNIQUES

A. Random Forest

Random forest is built on decision trees, decision three starts classification with the root variable and through binary decisions adds branches (variables) until it arrives at the leaf (class). Random Forest works by training multiple decision tree models, combining their classification results and using majority voting to arrive at final prediction.

B. K-Nearest Neighbor (K-NN)

K-NN algorithm compares given instance to the k number of nearest training instances and classifies the new instance based on majority voting of the nearest neighbor’s classes or by distance weighting. Distances are calculated using the Euclidean distance metric. A number of neighbors must be specified for better accuracy so by using Hyper parameter tuning we declared n\_neighbors as 2.

C. Support Vector Machines (SVM)

SVM is used to classify instances by linearly separating them with the highest margin possible between the class instances. When data is non-linear separable SVM uses a kernel and introduces another dimension in order to make the data separable with a hyperplane this type of action is referred to as kernel trick. SVM is highly preferred by many as it produces significant accuracy with less computation power. SVM is used for both regression and classification tasks. But, it is widely used in classification objectives.

V. APPLICATION OF TECHNIQUES AND RESULTS

Algorithms were implemented, and results obtained using HP Omen computer (laptop) equipped with Intel Core i7 6700HQ CPU with 2.60 GHz and 16GB RAM.

Models were trained using parameters that deliver the best results, parameters were obtained using grid search, Confusion Matrices, Classification Report and model metrics for all models were obtained and are presented for each model.

All models were trained using 11 best features and each model was trained multiple times to evaluate which class imbalance technique is best and to obtain the best results

Model evaluation metrics:

• 10-fold cross-validation was performed and model accuracy obtained.

• Model Accuracy

• Model Precision

• Sensitivity / Recall

• F1 Measure

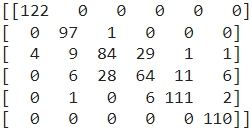
• Geometric Mean Measure

• Mathews Correlation Coefficient

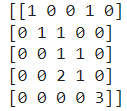
A. Random Forest

Results were obtained after training and testing the random forest method in Python from the scikit-learn repository. Confusion matrices and Classification reports for several imbalancing methods, including under sampling and oversampling, as well as the outcomes of applying hyperparameter adjustment to the model below, are available.

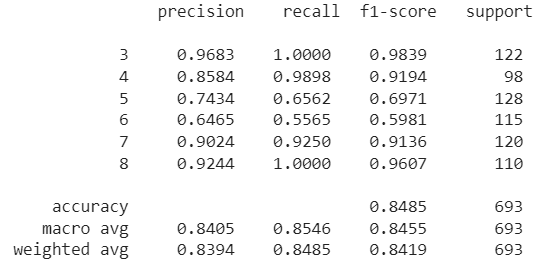
a. When used with Over Sampling SMOTE technique, the confusion matrix



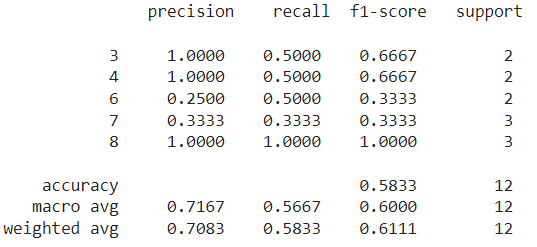
b. When used with Under Sampling Near Miss technique, the confusion matrix



c. When used with Over Sampling SMOTE technique, the Classification Report



c. When used with Under Sampling Near Miss technique, the Classification Report



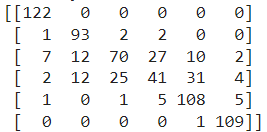
Comparing various Random Forest approaches for class imbalance, By comparing the SMOTE over sampling technique to the under sampling Near Miss technique, we can plainly see in classification report that the SMOTE over sampling technique performed better with 84.85 accuracy.

The best results were as follows: Train accuracy: 100 Test accuracy: 84.84

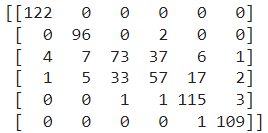
B. K-Nearest Neighbor (K-NN)

Results were obtained after training and testing the k-Nearest Neighbor method in Python from the scikit-learn repository. Confusion matrices and Classification reports for several imbalancing methods, including under sampling and oversampling, as well as the outcomes of applying hyperparameter adjustment to the model below, are available

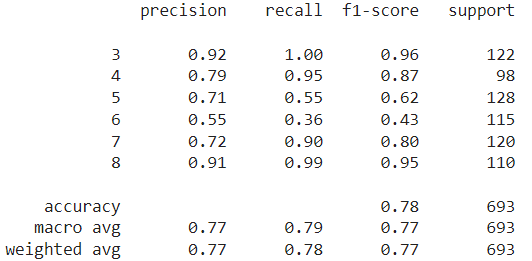
a. Before applying hyper parameter tuning When used with Over Sampling SMOTE technique, the confusion matrix



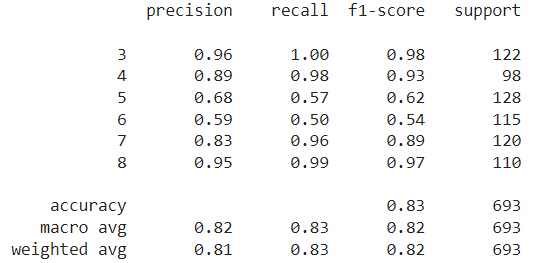
b. After applying hyper parameter tuning When used with Over Sampling SMOTE technique, the confusion matrix



c. Before applying hyper parameter tuning When used with Over Sampling SMOTE technique, the Classification Report



d. After applying hyper parameter tuning When used with Over Sampling SMOTE technique, the Classification report



1. *Dimensionality Reduction and Feature Extraction*

Dimensionality reduction is performed to improve the speed of data processing by reducing the amount of data to be processed while minimizing information loss, it also enables users to reduce data to two dimensions or three dimensions for visualization purposes. Dimensionality reduction can be done using Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA) or Kernel PCA (Gnanadesikan 1988, Hackeling 2014). For source code of techniques see appendix [IX.A.7)](#_bookmark29)

* 1. *LDA*

LDA is a supervised learning technique that analyses and identifies the features that have the highest class separation. In python, LDA was implemented using scikit-learn repository. LDA analysis was run however it has identified that features are collinear meaning features are closely correlated and it is unable to separate them, therefore, LDA is not an option for extracting features from the online shopper’s data set. A paper by Naes and Mevik (2001) states that this issue can be overcome by applying PCA.

* 1. *PCA*

PCA analyses features identifies their variance and sorts by highest variance. In python, PCA was implemented using scikit-learn repository and results displayed in [*Fig. 3*](#_bookmark5) and [*Fig. 4*](#_bookmark6)obtained.

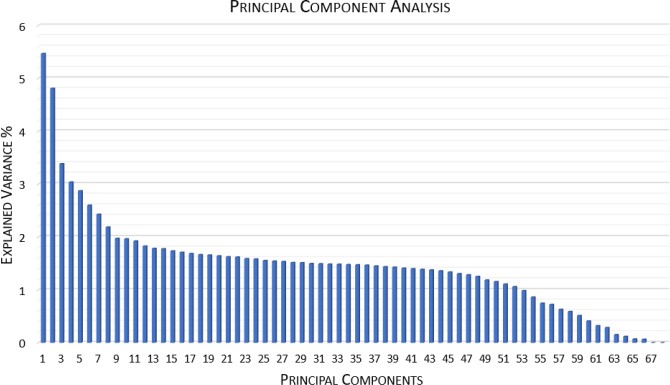


Fig. 3 PCA Individual Component Variance

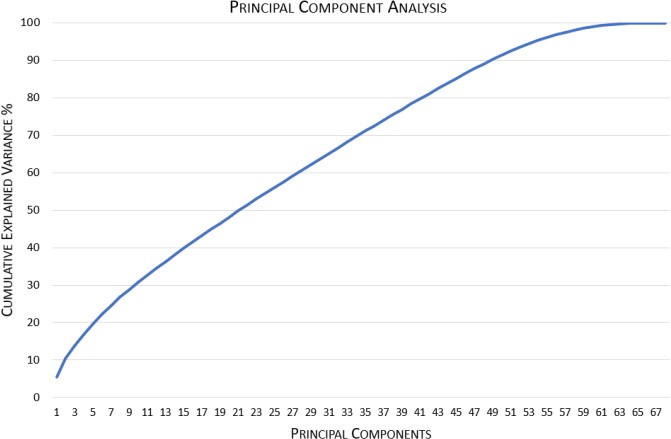


Fig. 4 PCA Cumulative Variance

Results show that most of the variables carry similar variance, meaning variables are almost equally important and only minimal dimensionality reduction can be performed.

* 1. *Kernel PCA*

Kernel PCA extends PCA by incorporating a kernel allowing to separate features that are non-linearly separable.

In python Kernel PCA was implemented using scikit-learn repository and results displayed in [*Fig. 5*](#_bookmark7)and [*Fig. 6*](#_bookmark8)(Hill and Lewicki 2006).

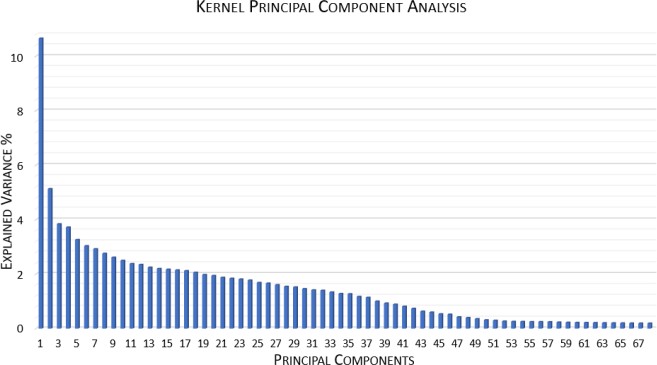


Fig. 5 Kernel PCA Individual Component Variance

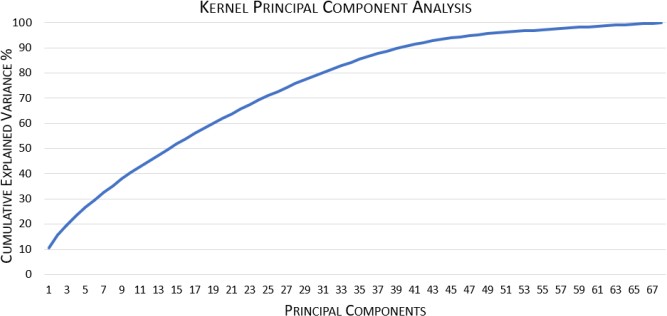


Fig. 6 Kernel PCA Cumulative Variance

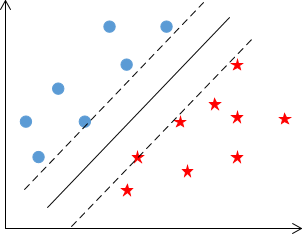
Analyzing results and comparing to PCA differences can be identified. Kernel PCA displays 38 components with a variance above 1% where PCA had 52 and Kernel PCA show that 40 components add up 90% of cumulative variance wherein PCA this number is 49. The comparison suggests that Kernel PCA performs better and it is the technique to be used for online shopper’s dataset, therefore LDA and PCA will not be used in this paper. Kernel PCA with 38 features will be used as it reduces dataset by 30 features while still having 88.77% cumulative variance.

1. Machine Learning Classification Techniques
2. *K-Nearest Neighbor (K-NN)*

K-NN algorithm compares given instance to the *k* number of nearest training instances and classifies the new instance based on majority voting of the nearest neighbor’s classes or by distance weighting. Distances are calculated using the Euclidean distance metric. A number of neighbors must be specified, and most commonly used number is 5 (Hill and Lewicki 2006).

1. *Support Vector Machines (SVM)*

SVM is used to classify instances by linearly separating them with the highest margin possible between the class instances. When data is non-linear separable SVM uses a kernel and introduces another dimension in order to make the data separable with a hyperplane this type of action is referred to as kernel trick. Examples of linear separation and non- linear separation utilizing kernel displayed in [Fig 7](#_bookmark9) [*Fig. 8*.](#_bookmark10) Figures are for example purposes and do not represent online shoppers dataset.



*Fig 7 SVM Linear Separation*

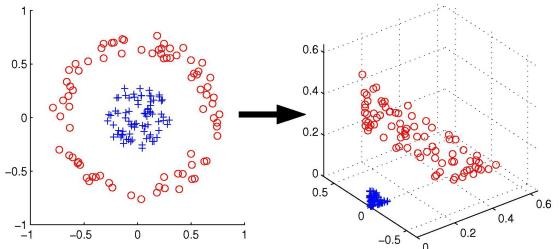


Fig. 8 SVM Non-linear Separation Using Kernel

1. *Ensembles*
   1. *Random Forest*

Random forest is built on decision trees, decision three starts classification with the root variable and through binary decisions adds branches (variables) until it arrives at the leaf (class). Random Forest works by training multiple decision tree models, combining their classification results and using majority voting to arrive at final prediction (Gra̧bczewski 2014).

* 1. *Adaptive Boosting (ADA Boost)*

ADA Boost most popular boosting algorithm. It uses other learning algorithms by repeatedly training them multiple times and each time focusing on misclassified data and adjusting to improve classification turning the weak learning algorithms into strong (Gra̧bczewski 2014). For online shopper’s classification ,ADA Boost was used with Random Forest classifier.

* 1. *Extremely Randomised Trees*

The algorithm is built on decision trees. It stands out from other ensemble tree-based models such as Random Forest and ADA Boost by splitting tree nodes at random and uses whole training data set rather than bootstrap version (Geurts et al. 2006).

1. APPLICATION OF TECHNIQUES AND RESULTS Algorithms were implemented, and results obtained using

HP Omen computer (laptop) equipped with Intel Core i7- 6700HQ CPU with 2.60 GHz and 16GB RAM.

Models were trained using parameters that deliver the best results, parameters were obtained using grid search. Confusion Matrices, ROC curve (except SVM) and model metrics for all models were obtained and are presented for each model.

All models were trained using 38 best features and each model was trained three times to evaluate which class imbalance technique is best and to obtain the best results.

Model evaluation metrics:

* 10-fold cross-validation was performed and model accuracy obtained.
* Model Accuracy
* Model Precision
* Specificity
* Sensitivity / Recall
* F1 Measure
* Geometric Mean Measure
* Mathews Correlation Coefficient
* Time taken to train and test. This includes 10 fold cross-validation and metrics calculations

1. *Support Vector machines*

Using python libraries SVC classifier, support vector machine model was trained and results presented in [*Fig. 9*](#_bookmark11) and [*TABLE 4*.](#_bookmark12)

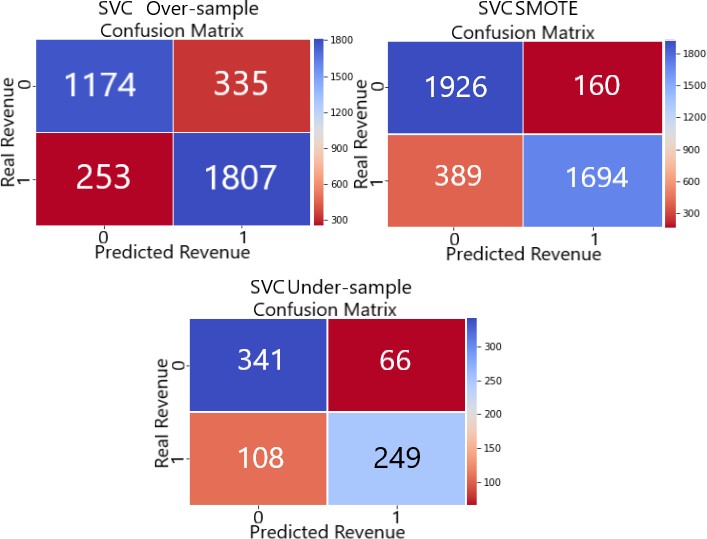


Fig. 9 Confusion Matrices SVM Classifier

TABLE 4 SUPPORT VECTOR MACHINES RESULTS SUMMARY

|  |  |  |  |
| --- | --- | --- | --- |
| **Support Vector Machines** | | | |
| ***Algorithm Parameters*** | | | |
| ***C*** | 4 | 10 | 4 |
| ***Kernel*** | Gaussian | Gaussian | Gaussian |
| ***Gamma*** | 0.3 | 1 | 1 |
| ***Results*** | | | |
| ***Class Imbalance*** | SMOTE | Over-sample | Under- sample |
| ***No of Test Examples*** | 4169 | 4169 | 764 |
| ***True Positive*** | 1694 | 1807 | 249 |
| ***True Negative*** | 1926 | 1774 | 341 |
| ***False Positive*** | 160 | 335 | 66 |
| ***False Negative*** | 389 | 253 | 108 |
| ***Correctly Classified*** | 3620 | 3581 | 590 |
| ***Miss Classified*** | 549 | 588 | 174 |
| ***Cross-Validation 10-Fold*** | | | |
| ***Accuracy*** | 0.86 | 0.85 | 0.76 |
| ***Variation +/-*** | 0.08 | 0.1 | 0.12 |
| ***Metrics*** | | | |
| ***Accuracy*** | 0.8683 | 0.8590 | 0.7723 |
| ***Precision*** | 0.9137 | 0.8436 | 0.7905 |
| ***Specificity*** | 0.9233 | 0.8412 | 0.8378 |
| ***Sensitivity / Recall*** | 0.8133 | 0.8772 | 0.6975 |
| ***F1 Measure*** | 0.8606 | 0.8601 | 0.7411 |
| ***G Measure*** | 0.8665 | 0.8590 | 0.7644 |
| ***Matthews Corr Coef*** | 0.7411 | 0.7186 | 0.5426 |
| ***Time taken*** | 136.25 | 169.78 | 5.40 |

Results show that SVM performed best using SMOTE over-sampling technique. The random over-sampling technique had very similar results and random under- sampling produced the worst results. Undersampling was expected to produce the worst results as the training set is significantly smaller when compared to Random over- sampling and SMOTE over-sampling.

Results also indicate that the dataset is challenging and basic models such as support vector machines have difficulties in learning and delivering perfect predictions. For source code on running the techniques refer to appendices

[IX.A.4)](#_bookmark27) for modeling function and producing results, [IX.A.8)](#_bookmark30) for source code using SMOTE, [IX.A.9)](#_bookmark31) for source code using over-sampling, [IX.A.10)](#_bookmark32) for source code using under- sampling, [IX.A.11)](#_bookmark33) for grid search.

1. *K-Nearest Neighbor*

Using python libraries K-Nearest Neighbor classifier was trained and results presented in [*Fig. 10*,](#_bookmark13) [*Fig. 11*](#_bookmark14) and [*TABLE 5*](#_bookmark15)

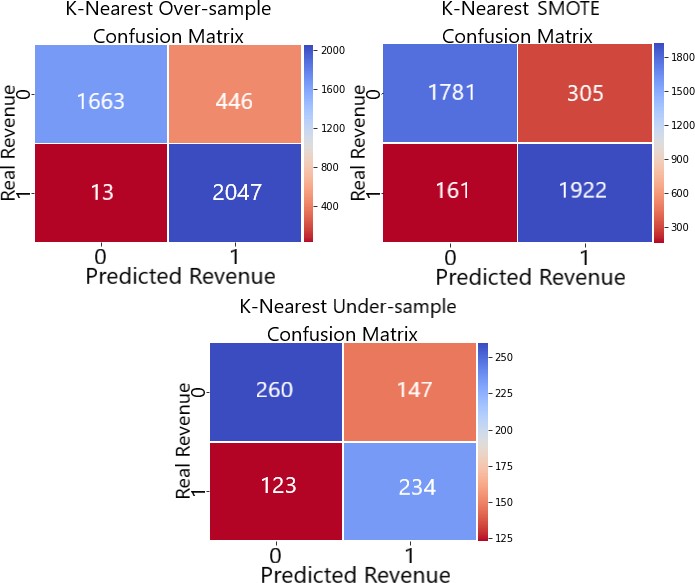


Fig. 10 Confusion Matrices K-Nearest Classifier

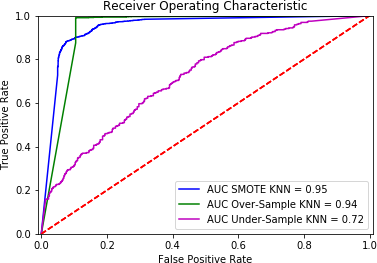


Fig. 11 ROC Curve K-Nearest

TABLE 5 K-NEAREST RESULTS SUMMARY

|  |  |  |  |
| --- | --- | --- | --- |
| **K-Nearest Neighbor** | | | |
| **Algorithm Parameters** | | | |
| ***Neighbors*** | 3 | 3 | 5 |
| ***Weights*** | Distance | Distance | Distance |
| ***Distance*** | Euclidean | Euclidean | Euclidean |
| ***Results*** | | | |
| ***Class Imbalance*** | SMOTE | Over- sampling | Under- Sampling |
| ***No of Test Examples*** | 4169 | 4169 | 764 |
| ***True Positive*** | 1922 | 2047 | 234 |
| ***True Negative*** | 1781 | 1663 | 260 |
| ***False Positive*** | 305 | 446 | 147 |
| ***False Negative*** | 161 | 13 | 123 |
| ***Correctly Classified*** | 3703 | 3710 | 494 |
| ***Miss Classified*** | 466 | 459 | 270 |
| ***Area Under ROC*** | 0.9460 | 0.9384 | 0.7181 |
| ***Cross-Validation 10-Fold*** | | | |
| ***Accuracy*** | 0.87 | 0.88 | 0.60 |
| ***Variation +/-*** | 0.07 | 0.06 | 0.18 |
| ***Metrics*** | | | |
| ***Accuracy*** | 0.8882 | 0.8899 | 0.6466 |
| ***Precision*** | 0.8630 | 0.8211 | 0.6142 |
| ***Specificity*** | 0.8538 | 0.7885 | 0.6388 |
| ***Sensitivity / Recall*** | 0.9227 | 0.9937 | 0.6555 |
| ***F1 Measure*** | 0.8919 | 0.8992 | 0.6341 |
| ***G Measure*** | 0.8876 | 0.8852 | 0.6471 |
| ***Matthews Corr Coef*** | 0.7783 | 0.7976 | 0.2937 |
| ***Time taken*** | 15.0816 | 7.7202 | 1.0166 |

K-nearest model results were very similar in comparison to SVM. In this scenario, random over-sampling technique delivered the best results however it still has misclassified 12% of the test cases.

1. *Random Forest*

Random forest algorithm from the scikit-learn repository in python was trained, tested and results obtained. Confusion matrices presented in [*Fig. 12*,](#_bookmark16) ROC curve in [*Fig. 13*](#_bookmark17) and results summary presented in [*TABLE 6*.](#_bookmark18)

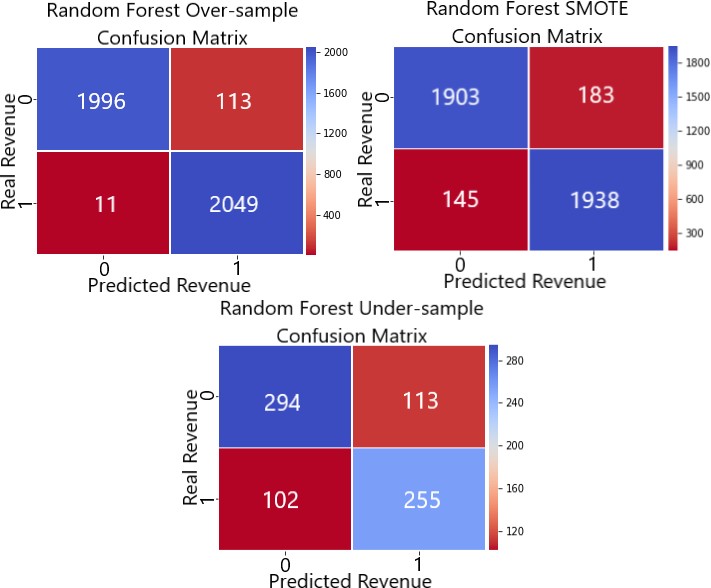


Fig. 12Confusion Matrices Random Forest

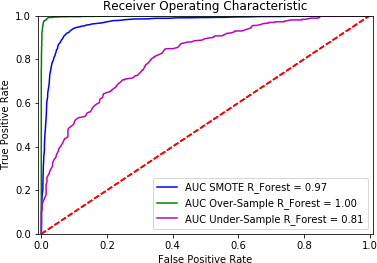


Fig. 13 ROC Curve Random Forest

TABLE 6 RANDOM FOREST RESULTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Random Forest** | | | |
| **Algorithm Parameters** | | | |
| ***No. Estimators*** | 145 | 123 | 110 |
| ***Results*** | | | |
| ***Class Imbalance*** | SMOTE | Over- sampling | Under-sampling |
| ***No of Test Examples*** | 4169 | 4169 | 764 |
| ***True Positive*** | 1938 | 2049 | 255 |
| ***True Negative*** | 1903 | 1996 | 294 |
| ***False Positive*** | 183 | 113 | 113 |
| ***False Negative*** | 145 | 11 | 102 |
| ***Correctly Classified*** | 3841 | 4045 | 549 |
| ***Miss Classified*** | 328 | 124 | 215 |
| ***Area Under ROC*** | 0.9666 | 0.9983 | 0.8138 |
| ***Cross-Validation 10-Fold*** | | | |
| ***Accuracy*** | 0.90 | 0.97 | 0.69 |
| ***Variation +/-*** | 0.07 | 0.04 | 0.19 |
| ***Metrics*** | | | |
| ***Accuracy*** | 0.9213 | 0.9703 | 0.7186 |
| ***Precision*** | 0.9137 | 0.9477 | 0.6929 |
| ***Specificity*** | 0.9123 | 0.9464 | 0.7224 |
| ***Sensitivity / Recall*** | 0.9304 | 0.9947 | 0.7143 |
| ***F1 Measure*** | 0.9220 | 0.9706 | 0.7034 |
| ***G Measure*** | 0.9213 | 0.9702 | 0.7183 |
| ***Matthews Corr Coef*** | 0.8428 | 0.9417 | 0.4360 |
| ***Time taken*** | 117.74 | 103.56 | 20.76 |

Comparing Random Forest results with KNN and SVM there is a significant improvement in the results. Comparing different class imbalance techniques of Random Forest, over- sampling outperformed other two in almost all the areas.

1. *Adaptive Boosting*

The adaptive boosting model was based on already well performing random forest algorithm. Adaptive boosting was used with SAMME parameter algorithm which focuses on misclassified instances and SAMME.R on their probabilities. SAMME.R is faster to run, but grid search identified that SAMME delivers better results (Scikit-Learn 2018). The model was trained, tested and results obtained. Confusion matrices displayed in [*Fig. 14*,](#_bookmark19) ROC curve [*Fig. 15*](#_bookmark20)and results summary in [*TABLE 7*.](#_bookmark21)

Fig. 14 Confusion Matrices Adaptive Boosting

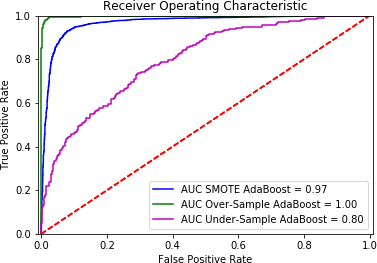
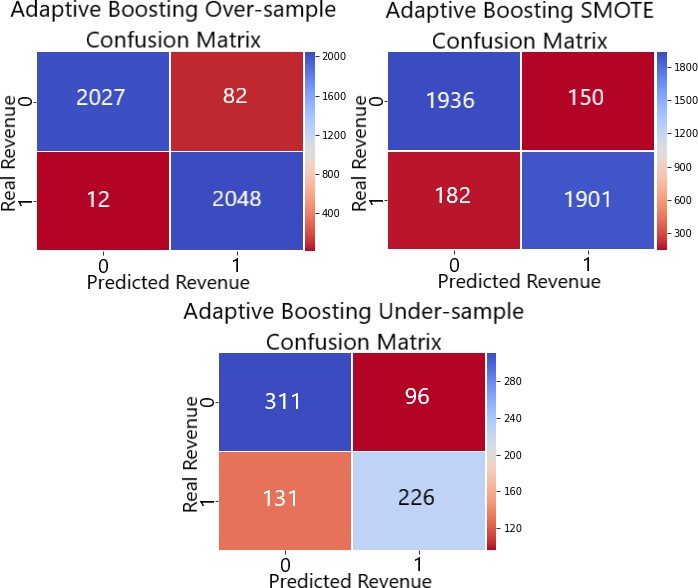


Fig. 15 ROC Curve Adaptive Boosting

TABLE 7 ADAPTIVE BOOSTING RESULTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Adaptive Boosting** | | | |
| **Algorithm Parameters** | | | |
| ***Base Estimator*** | Random Forest | | |
| ***No. Estimators*** | 123 | 123 | 84 |
| ***Algorithm*** | SAMME | | |
| ***Results*** | | | |
| ***Class Imbalance*** | SMOTE | Over- sampling | Under-sampling |
| ***No of Test Examples*** | 4169 | 4169 | 764 |
| ***True Positive*** | 1901 | 2048 | 226 |
| ***True Negative*** | 1936 | 2027 | 311 |
| ***False Positive*** | 150 | 82 | 96 |
| ***False Negative*** | 182 | 12 | 131 |
| ***Correctly Classified*** | 3837 | 4075 | 537 |
| ***Miss Classified*** | 332 | 94 | 227 |
| ***Area Under ROC*** | 0.9675 | 0.9980 | 0.7989 |
| ***Cross-Validation 10-Fold*** | | | |
| ***Accuracy*** | 0.90 | 0.98 | 0.79 |
| ***Variation +/-*** | 0.08 | 0.03 | 0.19 |
| ***Metrics*** | | | |
| ***Accuracy*** | 0.9204 | 0.9775 | 0.7029 |
| ***Precision*** | 0.9269 | 0.9615 | 0.7019 |
| ***Specificity*** | 0.9281 | 0.9611 | 0.7641 |
| ***Sensitivity / Recall*** | 0.9126 | 0.9942 | 0.6331 |
| ***F1 Measure*** | 0.9197 | 0.9776 | 0.6657 |
| ***G Measure*** | 0.9203 | 0.9775 | 0.6955 |
| ***Matthews Corr Coef*** | 0.8408 | 0.9555 | 0.4013 |
| ***Time taken*** | 203.37 | 176.19 | 16.96 |

As expected, results adaptive boosting have delivered improved from results Random Forest results. When comparing different class imbalance techniques random over-sample outperformed other techniques in all measures.

1. *Extremely Randomized Trees*

Extremely randomized trees also part of the ensembles group of algorithms. The model was trained, tested and results obtained. Confusion matrices displayed in [*Fig. 16*,](#_bookmark22) ROC curve in [*Fig 17*](#_bookmark24) and results summary in [*TABLE 8*](#_bookmark23)

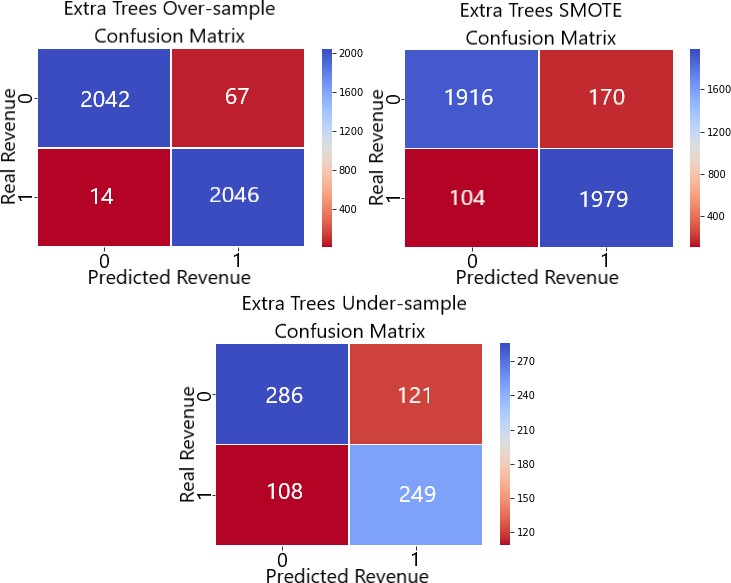


Fig. 16 Extremely Randomized Trees

TABLE 8 EXTREMELY RANDOMIZED TREES RESULTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Extremely Randomized Trees** | | | |
| **Algorithm Parameters** | | | |
| ***No. Estimators*** | 2000 | 144 | 162 |
| ***Results*** | | | |
| ***Class Imbalance*** | SMOTE | Over- sampling | Under-sampling |
| ***No of Test Examples*** | 4169 | 4169 | 764 |
| ***True Positive*** | 1979 | 2046 | 249 |
| ***True Negative*** | 1916 | 2042 | 286 |
| ***False Positive*** | 170 | 67 | 121 |
| ***False Negative*** | 104 | 14 | 108 |
| ***Correctly Classified*** | 3895 | 4088 | 535 |
| ***Miss Classified*** | 274 | 81 | 229 |
| ***Area Under ROC*** | 0.9721 | 0.9989 | 0.7910 |
| ***Cross-Validation 10-Fold*** | | | |
| ***Accuracy*** | 0.92 | 0.98 | 0.66 |
| ***Variation +/-*** | 0.07 | 0.02 | 0.2 |
| ***Metrics*** | | | |
| ***Accuracy*** | 0.9343 | 0.9806 | 0.7003 |
| ***Precision*** | 0.9209 | 0.9683 | 0.6730 |
| ***Specificity*** | 0.9185 | 0.9682 | 0.7027 |
| ***Sensitivity / Recall*** | 0.9501 | 0.9932 | 0.6975 |
| ***F1 Measure*** | 0.9353 | 0.9806 | 0.6850 |
| ***G Measure*** | 0.9342 | 0.9806 | 0.7001 |
| ***Matthews Corr Coef*** | 0.8690 | 0.9615 | 0.3995 |
| ***Time taken*** | 532.67 | 32.14 | 8.34 |

Extremely randomized trees delivered excellent results. Its ROC curve is almost vertical, visibly different from other models. The random over-sampling technique produced excellent results outperforming SMOTE and under-sampling in all areas.

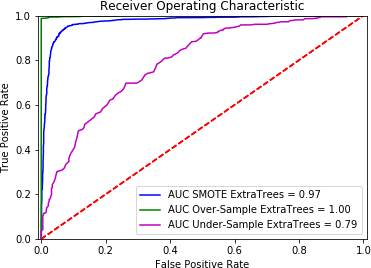


Fig 17 ROC Curve Extremely Randomized Trees

1. Conclusion

Random over-sampling class imbalance resolution technique performed best across all the models (except SVM). Results for random over-sampling has been combined and presented in [*Table 9*](#_bookmark25) for comparison.

Table 9 Random-oversampling Results Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **SVM** | **KNN** | **Random Forest** | **Extra Trees** | **Adaptive Boosting** |
| ***No of Test Examples*** | 4169 | 4169 | 4169 | 4169 | 4169 |
| ***True Positive*** | 1807 | 2047 | 2049 | 2046 | 2048 |
| ***True Negative*** | 1774 | 1663 | 1996 | 2042 | 2027 |
| ***False Positive*** | 335 | 446 | 113 | 67 | 82 |
| ***False Negative*** | 253 | 13 | 11 | 14 | 12 |
| ***Correctly Classified*** | 3581 | 3710 | 4045 | 4088 | 4075 |
| ***Miss Classified*** | 588 | 459 | 124 | 81 | 94 |
| ***Area Under ROC*** | N/A | 0.9384 | 0.9983 | 0.9989 | 0.9980 |
| ***Cross-Validation 10-Fold*** | | | | | |
| ***Accuracy*** | 0.85 | 0.88 | 0.97 | 0.98 | 0.98 |
| ***Variation +/-*** | 0.10 | 0.06 | 0.04 | 0.02 | 0.03 |
| ***Metrics*** | | | | | |
| ***Accuracy*** | 0.8590 | 0.8899 | 0.9703 | 0.9806 | 0.9775 |
| ***Precision*** | 0.8436 | 0.8211 | 0.9477 | 0.9683 | 0.9615 |
| ***Specificity*** | 0.8412 | 0.7885 | 0.9464 | 0.9682 | 0.9611 |
| ***Sensitivity / Recall*** | 0.8772 | 0.9937 | 0.9947 | 0.9932 | 0.9942 |
| ***F1 Measure*** | 0.8601 | 0.8992 | 0.9706 | 0.9806 | 0.9776 |
| ***G Measure*** | 0.8590 | 0.8852 | 0.9702 | 0.9806 | 0.9775 |
| ***Matthews Corr Coef*** | 0.7186 | 0.7976 | 0.9417 | 0.9615 | 0.9555 |
| ***Time taken (seconds)*** | 169.78 | 7.72 | 103.56 | 32.14 | 176.19 |

Review of random over-sampling results show that ensemble models produced very similar results and extremely randomized model had the best results in most areas, especially in the time taken to process. Extremely randomized trees were more than three times faster when compared to random forest and five times faster when compared to adaptive boosting.

Based on results obtained the best model to use for online shoppers’ intentions prediction is Extremely Randomized Trees with a dataset that had its classes balanced using random over-sampling technique and with just 38 features that were extracted using Kernel PCA.

1. Future Research

The trained model already performed well, however deep learning methods or neural networks could be applied to it in attempt to get better performance as well as more in-depth grid search technique combined with piping which allows to search for an optimum number of components for an algorithm for each of the specific parameters.

1. References

Alpaydın, E. (2014) *Introduction to Machine Learning*. 3. ed. Adaptive computation and machine learning. Cambridge, Mass.: MIT Press

Brownlee, J. (2017) ‘Why One-Hot Encode Data in Machine Learning?’ [27 July 2017] available from

<https://machinelearningmastery.com/why-one- hot-encode-data-in-machine-learning/> [13

December 2018]

Geurts, P., Ernst, D., and Wehenkel, L. (2006) ‘Extremely Randomized Trees’. *Machine Learning* 63 (1), 3–42

Gnanadesikan, R. (1988) *Discriminant Analysis and Clustering.* [online] National Academies Press. available from

[<http://p](http://public.eblib.com/choice/publicfullrecord.as)u[blic.eblib.com/choice/publicfullrecord.as](http://public.eblib.com/choice/publicfullrecord.as) px?p=4388301> [7 December 2018]

Gra̧bczewski, K. (2014) *Meta-Learning in Decision Tree Induction*. Studies in computational intelligence Volume 498. Cham ; New York: Springer

Hackeling, G. (2014) *Mastering Machine Learning with Scikit-Learn: Apply Effective Learning Algorithms to Real-World Problems Using Scikit-Learn*. Packt open source. Birmingham: Packt Publ

He, H. (2011) *Self-Adaptive Systems for Machine Intelligence*

[online] Hoboken, N.J.: Wiley. available from

[<http://](http://site.ebrary.com/id/10484685)s[ite.ebrary.com/id/10484685](http://site.ebrary.com/id/10484685)> [7 December

2018]

Hill, T. and Lewicki, P. (2006) *Statistics: Methods and Applications: A Comprehensive Reference for Science, Industry, and Data Mining*. Tulsa, OK: StatSoft

Kanter, J. (2018) *Cambridge Analytica Bosses Were Secretly Filmed Boasting about How They Helped Trump Win the US Election* [online] available from

<https://[www.businessinsider.com/cambridge-](http://www.businessinsider.com/cambridge-) analytica-boasts-won-trump-election-facebook- data-2018-3> [13 December 2018]

Kumar, M. (2018) *How To Handle Imbalance Data : Study in Detail | Kaggle* [online] available from

<https://[www.kaggle.com/gargmanish/how-to-](http://www.kaggle.com/gargmanish/how-to-) handle-imbalance-data-study-in-detail> [13

December 2018]

Naes, T. and Mevik, B.-H. (2001) ‘Understanding the Collinearity Problem in Regression and Discriminant Analysis: COLLINEARITY PROBLEM IN REGRESSION AND DISCRIMINANT ANALYSIS’. *Journal of*

*Chemometrics* 15 (4), 413–426

Scikit-Learn (2018) *sklearn.ensemble.AdaBoostClassifier — Scikit-Learn 0.20.1 Documentation* [online] available from <https://scikit- learn.org/stable/modules/generated/sklearn.ensemb le.AdaBoostClassifier.html> [13 December 2018]

Statista (2018) *Global Retail E-Commerce Market Size 2014- 2021* [online] available from

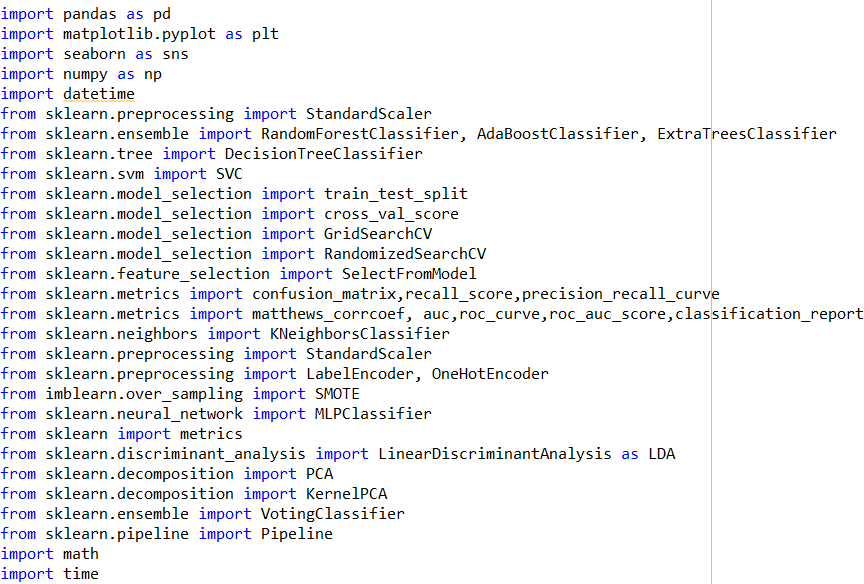
<https://[www.statista.com/statistics/379046/world](http://www.statista.com/statistics/379046/world) wide-retail-e-commerce-sales/> [13 December

2018]

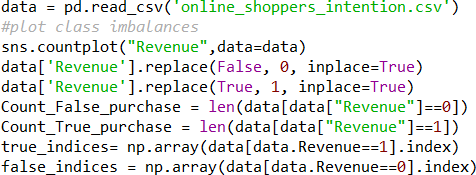
1. Appendix

*A. Source code*

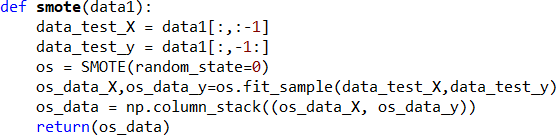
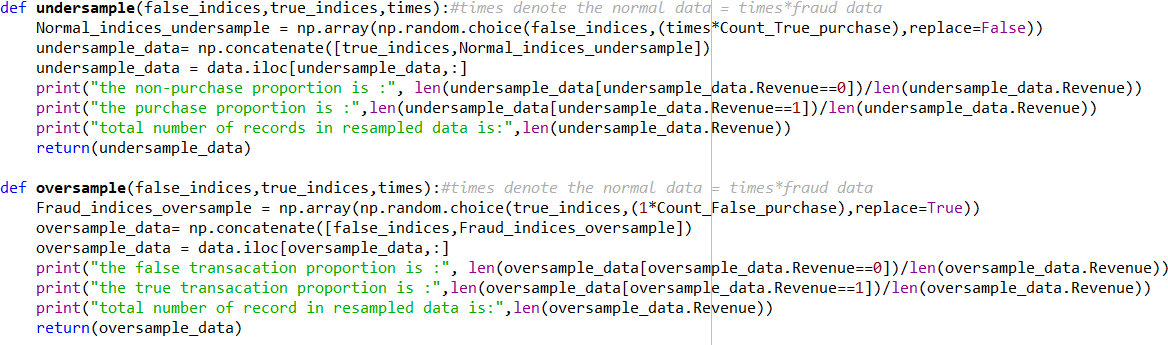
1. *Libraries that were imported as part of the experiment*



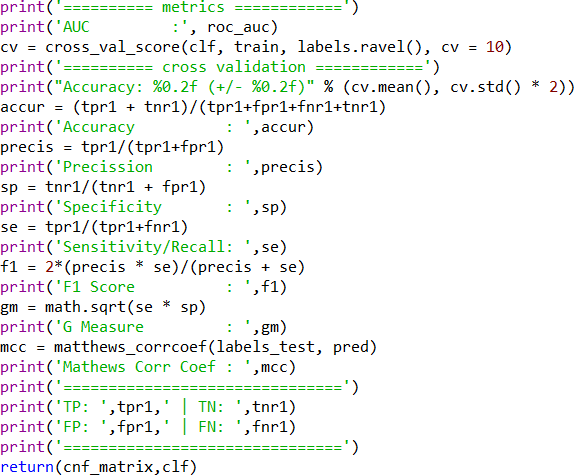
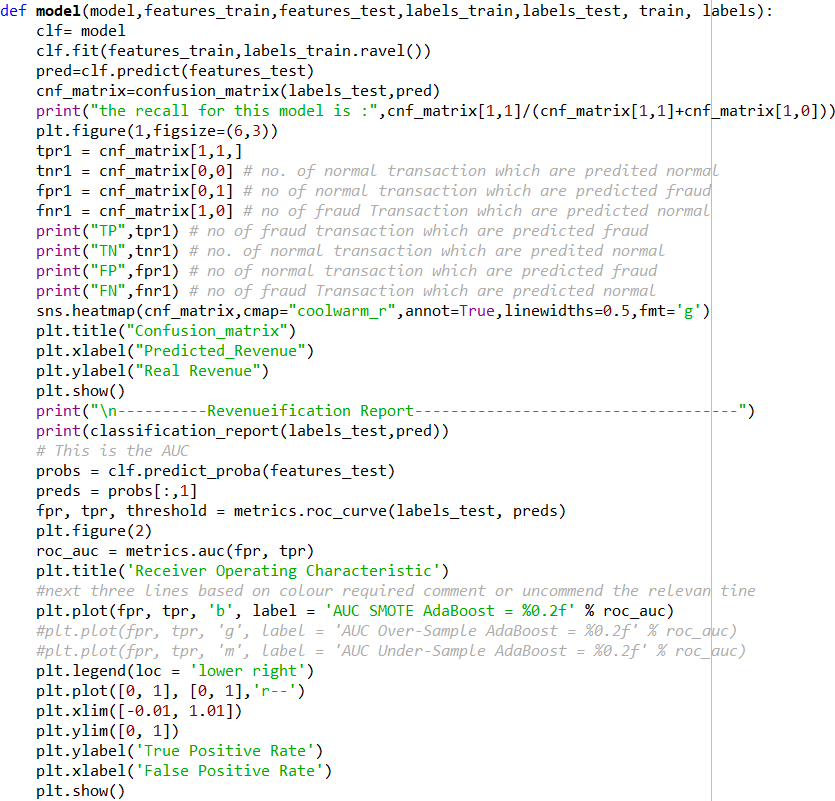
1. *Importing Dataset, Pot Class Imbalance and Prepare for Over/Under-sampling*

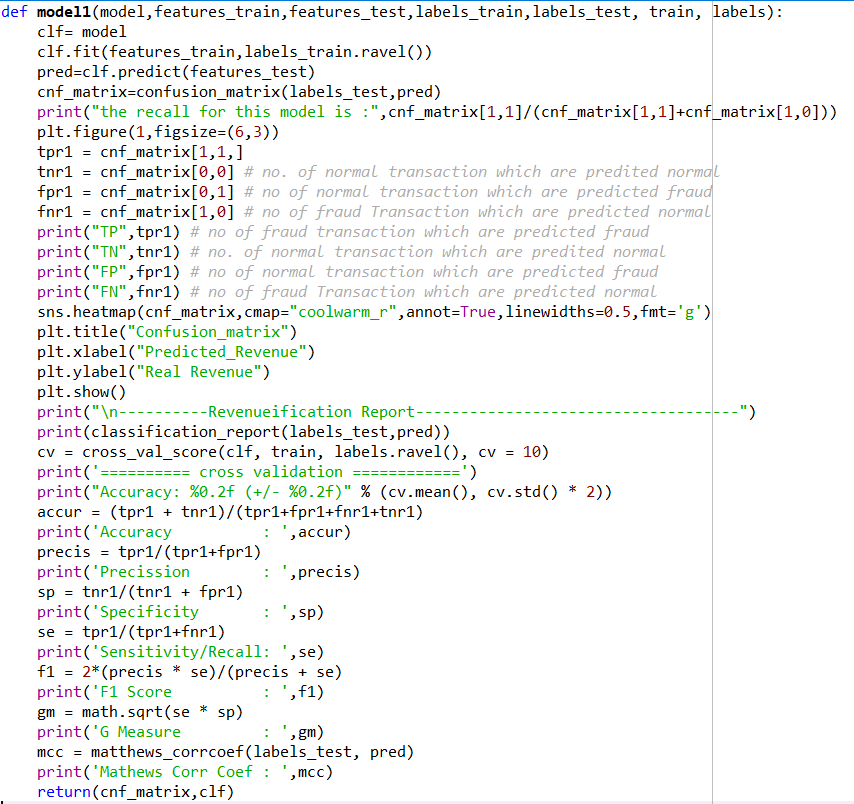


1. *Functions Defined and Used for Over/Under-Sampling*

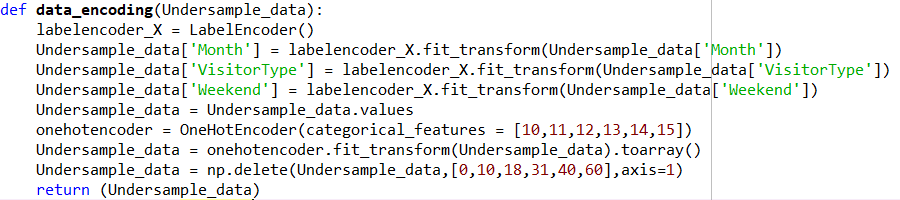


1. *Model and Model1 Functions Used for Training, Testing, Validation, and Metrics Calculation*

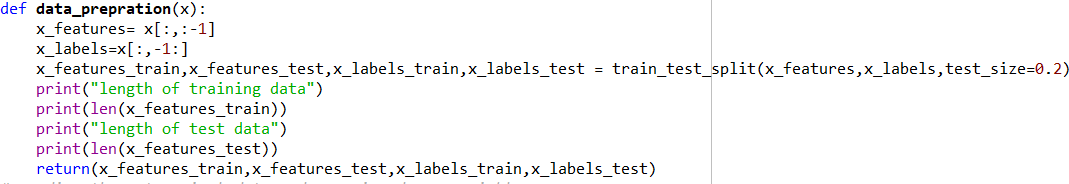




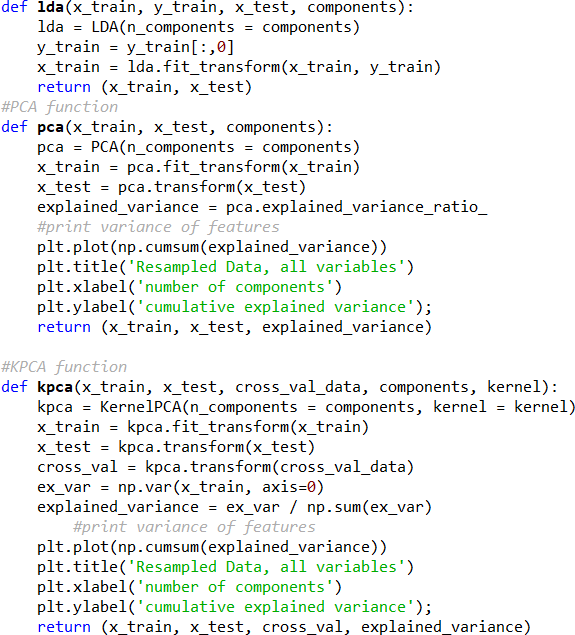
1. *Categorical Data encoding Function*



1. *Splitting Data into Train and Test Sets Function*



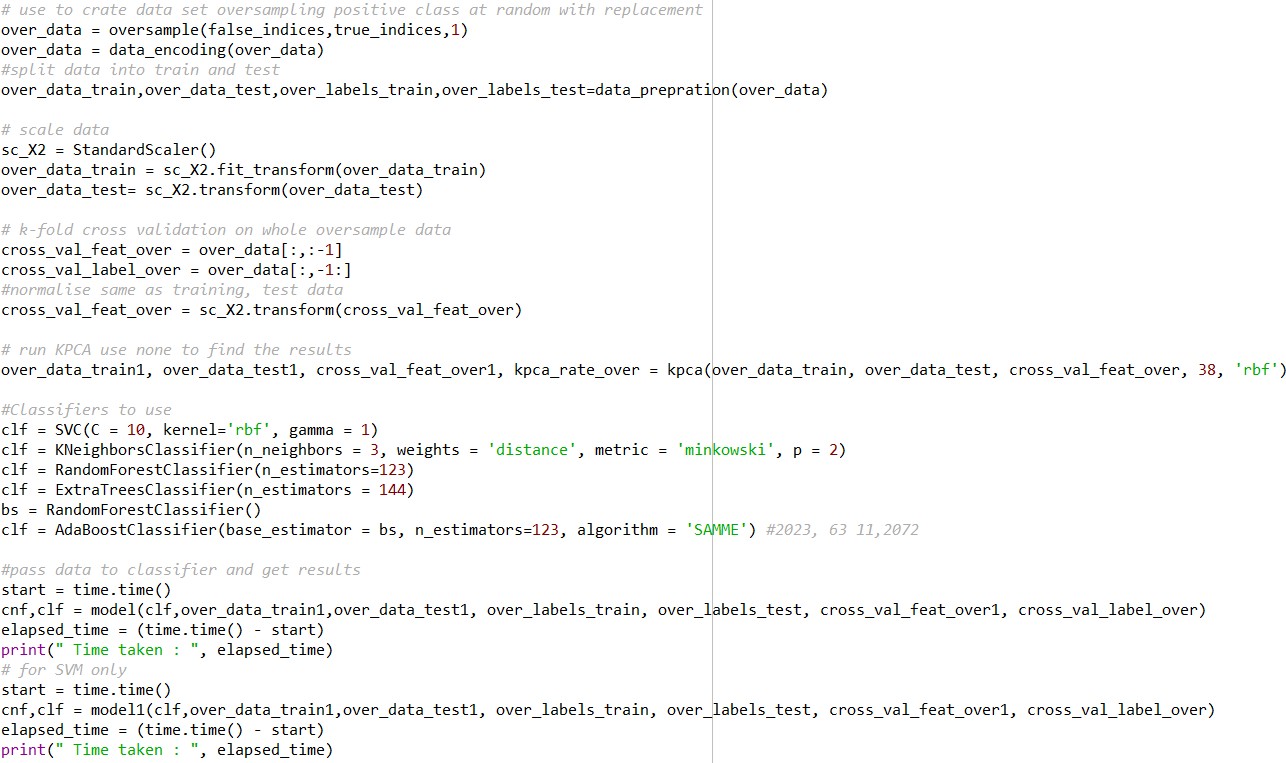
1. *LDA, PCA, KPCA Functions*



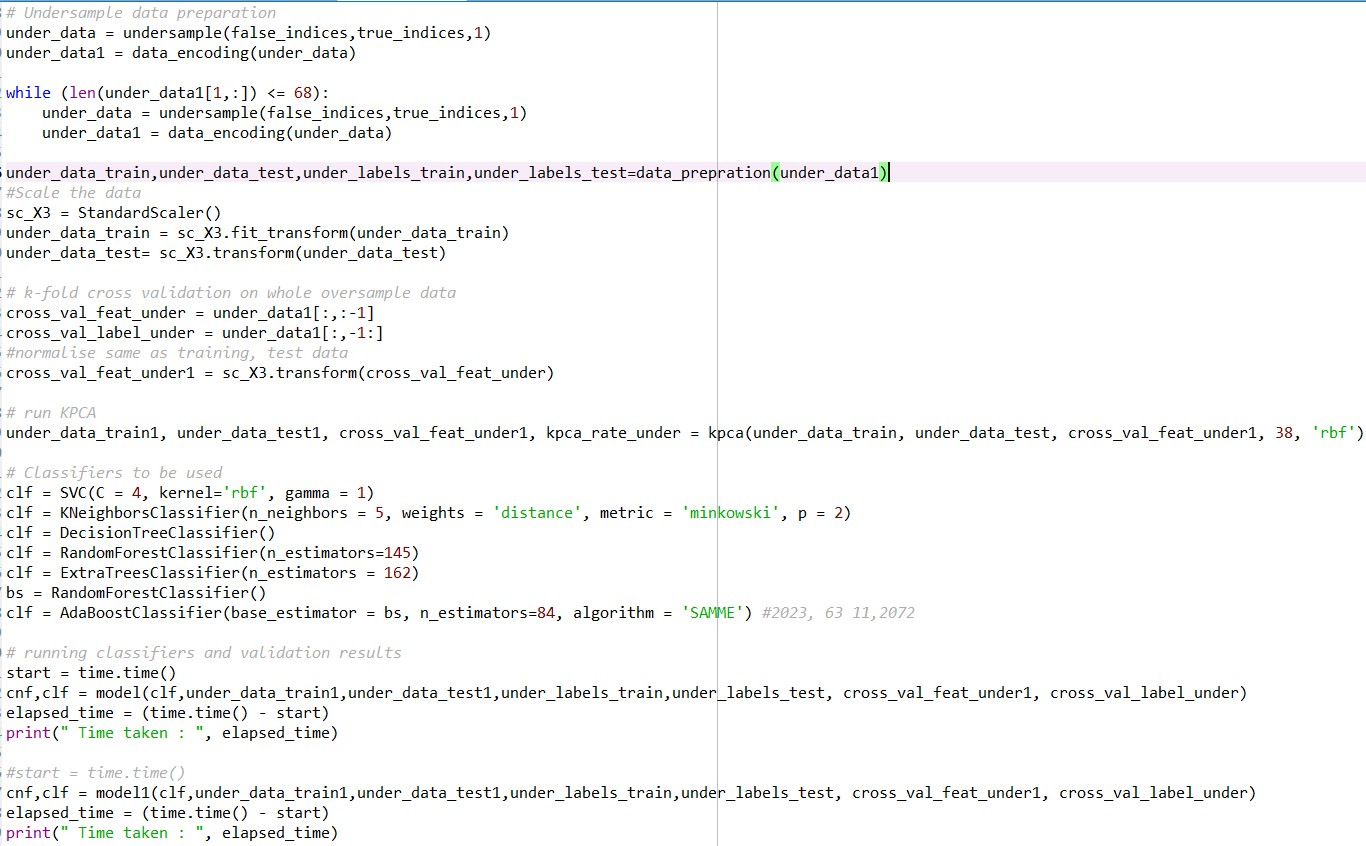
1. *Running SMOTE Scenario*



1. *Running Random Over-sampling Scenario*



1. *Running Under-sample Scenario*



1. *Grid Search*

