Prediction of Online Shoppers Purchasing Intentions using Support Vector Machines, K-Nearest Neighbor, Random Forest, Adaptive Boosting and Extremely Randomized Trees Algorithms

Andrius Bertulis
Faculty of Engineering Environment and Computing
Coventry University
Coventry, England
bertulia@uni.coventry.ac.uk

***Abstract*—This paper looks at how online shopper's intentions could be predicted by analyzing their behavior and applying Support Vector Machines, K-Nearest Neighbor, Random Forest, Adaptive Boosting and Extremely Randomized Trees machine learning algorithms. Multiple data preparation techniques were implemented covering class imbalance issues, encoding categorical data and feature extraction. 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; class imbalance; feature extraction; categorical data; online shoppers' intention, adaptive boosting, extremely randomized trees, random forest, k-nearest, support vector machines.

I. INTRODUCTION

The online retail market is increasing. According to Statista (Statista 2018), it has more than doubled since 2014. In 2014 it was 1336 billion US dollars and in 2018 at the end of the year, it is expected to be 2842 billion US dollars. Businesses using e-commerce applying various online marketing techniques to attract customers to their webpages and to purchase their products or services. However, to properly analyze what works best they can only look at one technique at a time, but there are multiple factors that contribute to the customer decision to make a purchase and for a human to analyze all of them together would take a very long time or not be possible at all. Machine learning comes to assist. Computers can process information faster than humans. Each customer behavior can be analyzed and identified how it contributes to customer purchase. When a new customer visits the page, the computer can recognize their behavior and if the behavior suggests a non-purchase customer, various techniques could be used to attempt to influence and change the behavior of the customer so it could lead to the purchase. Similar techniques, but on the bigger scale were utilized by Cambridge Analytica to influence voters decision in America's elections (Kanter 2018). This indicates how powerful machine learning can be.

II. THE DATA SET

Dataset was obtained from the UCI Machine Learning repository. It contains 12330 instances each containing 18 attributes with no missing values. Each instance is one person's unique visit to the site. For returning customer visits to the site, where customer visited multiple times, only first visit from the time period analyzed is taken and subsequent visits not included. First 17 attributes are the features relating to the visitor and its behavior and consist of 10 numerical values and 7 categorical features. Last 18th feature represents the class if the visit lead to revenue or not meaning if visitor made a purchase it is classed as True or did not purchase False. *TABLE 1* display the summary of the data set describing features, feature types and value ranges for categorical features.

TABLE 1. DATASET FEATURES

No	Description	Type	Categorical Value Range
1	Administrative	Numeric	
2	Administrative Duration	Numeric	
3	Informational	Numeric	
4	Informational Duration	Numeric	
5	Product Related	Numeric	
6	Product Related Duration	Numeric	
7	Bounce Rates	Numeric	
8	Exit Rates	Numeric	
9	Page Value	Numeric	
10	Special Day	Numeric	
11	Month	Categorical	February to December
12	Operating Systems	Categorical	1 to 8
13	Browser	Categorical	1 to 13
14	Region	Categorical	1 to 9
15	Traffic Type	Categorical	1 to 20
16	Visitor Type	Categorical	New, Returning, Other
17	Weekend	Categorical	True, False
18	Revenue	Categorical	True, False

III. DATA PREPARATION

This section covers data analysis on class imbalance and overcoming imbalance issues, dealing with categorical data and categorical data encoding, analysis of numerical values and data scaling, feature analysis, and feature extraction.

A. Class Imbalance

Haibo He in the book Self-adaptive Systems for Machine Intelligence (2011) explains that class imbalance caused

machine learning algorithms to underperform and this issue must be addressed before proceeding to implement algorithms on the dataset. Haibo He suggests to use random under-sampling, random over-sampling or SMOTE over-sampling techniques. In this paper, class imbalance analysis was carried out on the dataset using python. Example code was obtained from Kaggle written by Manish Kumar (2018) and modified to suit dataset. The analysis identified 10422 visits that lead to false revenue (non-purchase) and just 1908 that led to true revenue (Purchase). Results visualized in *Fig 1* show class imbalance on the dataset. Code for importing data set and plotting class imbalance see appendix IX.A.2)

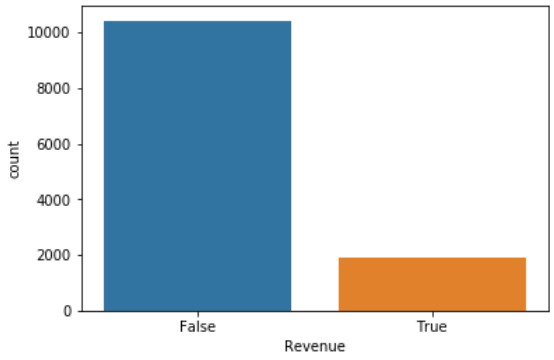

Fig 1. Class Imbalance

To overcome these three techniques were used as suggested by He. For source code see appendix IX.A.7)

1) Random Under-sampling

Random under-sampling is performed by randomly selecting a number of samples from the majority class to match the number of samples of a minority class. In this paper 1908 instances were selected where revenue was false and merged with 1908 instances where revenue is True, to create a new dataset of 3816 instances.

2) Random Over-sampling

Random over-sampling is performed by randomly duplicating instances of minority class to match the number of instances from majority class. In this paper, 10422 instances were selected at random with replacement from minority class and merged with 10422 instances of majority class to create a new dataset containing 20844 instances.

3) SMOTE Over-sampling

SMOTE over-sampling algorithm creates synthetic instances by a true revenue example and selecting the nearest neighbor that is also a True revenue example and creating an instance between those two records. A visual example of SMOTE displayed in *Fig 2*.

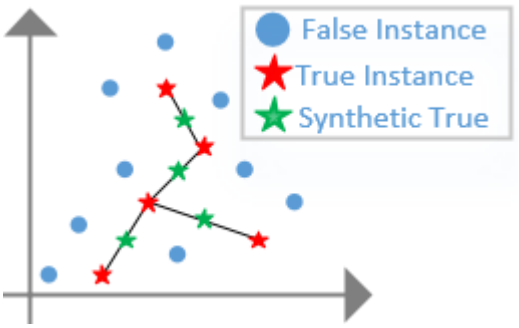

Fig 2. SMOTE Example

In this paper, SMOTE was implemented using the algorithm from python imbalanced learning library. Synthetic instances of True revenue class were created to match the number of true revenue class with false revenue class and new dataset produced with 20844 instances.

B. Categorical Feature Encoding

Many machine learning algorithms only work with numeric values, but categorical data can be represented as text. Even if categorical data is represented as a number (i.e. months 1-12) machine learning algorithm could give higher weight to December as it carries number 12 than January 1 and not classify them in the same way. To avoid this categorical data must be encoded into binary values and to simplify further dummy variable can be removed in order to reduce dimensionality of the dataset (Brownlee 2017, Alpaydın 2014). In this paper, eight categorical features have been encoded using python library scikit-learn utilizing LabelEncoder to encode labels into numbers (e.g. month from text February to number 2 and Tree to 1) and OneHotEncoder to convert values to binary as per example displayed in *TABLE 2*.

TABLE 2 ENCODING CATEGORICAL FEATURES TO BINARY

Text		**Numerical**		**Binary**		
Month		***Month***		***February***	***March***	***April***
February	→	2	→	1	0	0
March		3		0	0	1
April		4		0	1	0

Once encoding is completed the dummy variables are removed. (e. g. by removing one month it is not lost, it is represented by zeros in remaining month columns) the example displayed in *TABLE 3* show April being removed, but it can be identified by zero values in February and March columns same technique was applied in online shoppers dataset. For source code on encoding and dummy variable removal see appendix IX.A.5)

TABLE 3 DUMMY VARIABLE REMOVAL

February	**March**	**April**		**February**	**March**
1	0	0	→	1	0
0	0	1		0	0
0	1	0		0	1

C. Data Standardisation

In order to evaluate each feature equally, data standardization must be performed to ensure all the features are on the same scale and none of the features are given higher weighting by the algorithms. This could happen as 100 is not the same as 1 and in the formula 100 would have a higher weighting for prediction. Standardization is done by subtracting the mean from each feature and divide by the standard deviation (1) (Hackeling 2014).

$$x' = \frac{x - \bar{x}}{\sigma} \quad (1)$$

In this paper, standardization was completed utilizing python Standard Scaler class from preprocessing module in scikit-learn library.

D. *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)

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.

2) 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* and *Fig. 4* obtained.

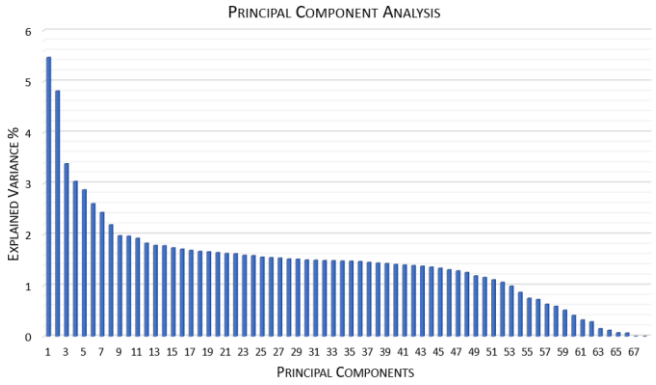

Fig. 3 PCA Individual Component Variance

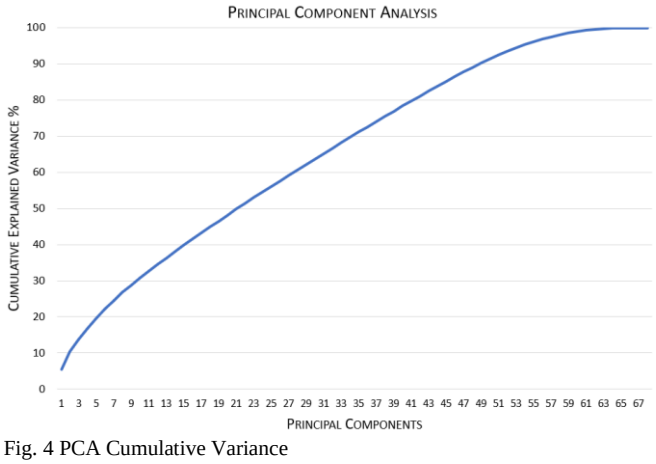

Fig. 4 PCA Cumulative Variance

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

3) 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* and *Fig. 6* (Hill and Lewicki 2006).

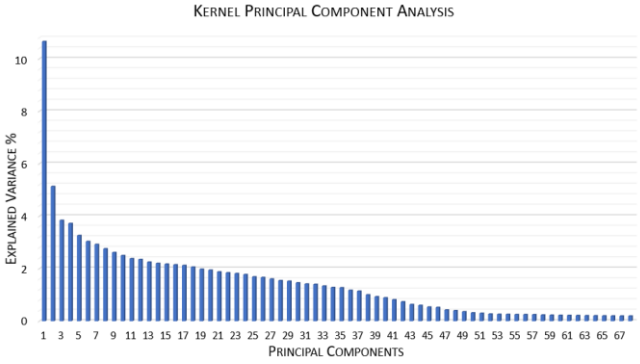

Fig. 5 Kernel PCA Individual Component Variance

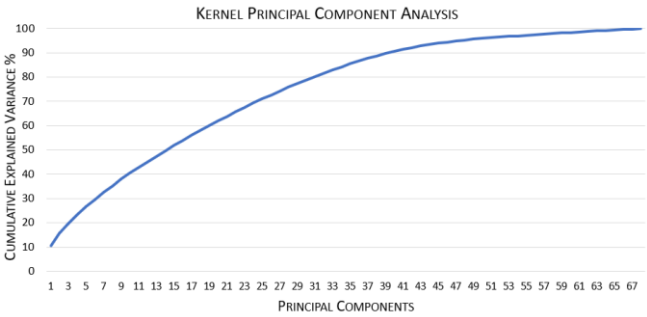

Fig. 6 Kernel PCA Cumulative Variance

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.

IV. Machine Learning Classification Techniques

A. *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).

B. *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 *Fig. 8*. Figures are for example purposes and do not represent online shoppers dataset.

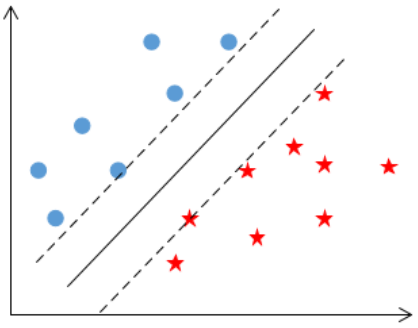

Fig 7 SVM Linear Separation

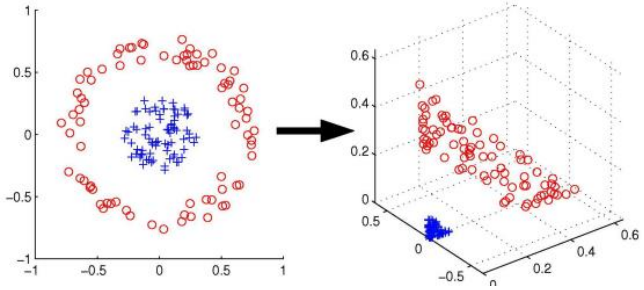

Fig. 8 SVM Non-linear Separation Using Kernel

C. *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 (Grąbczewski 2014).

2) 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 (Grąbczewski 2014). For online shopper's classification ,ADA Boost was used with Random Forest classifier.

3) 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).

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, 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

A. *Support Vector machines*

Using python libraries SVC classifier, support vector machine model was trained and results presented in *Fig. 9* and *TABLE 4*.

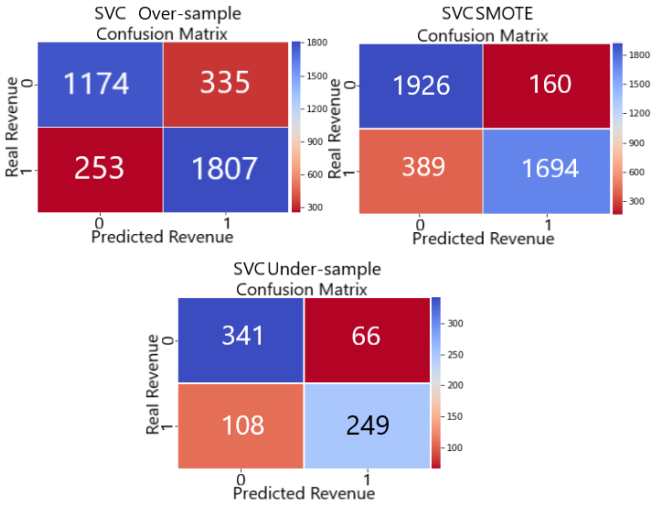

Fig. 9 Confusion Matrices SVM Classifier

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) for modeling function and producing results, IX.A.8) for source code using SMOTE, IX.A.9) for source code using over-sampling, IX.A.10) for source code using under-sampling, IX.A.11) for grid search.

B. *K-Nearest Neighbor*

Using python libraries K-Nearest Neighbor classifier was trained and results presented in *Fig. 10*, *Fig. 11* and *TABLE 5*

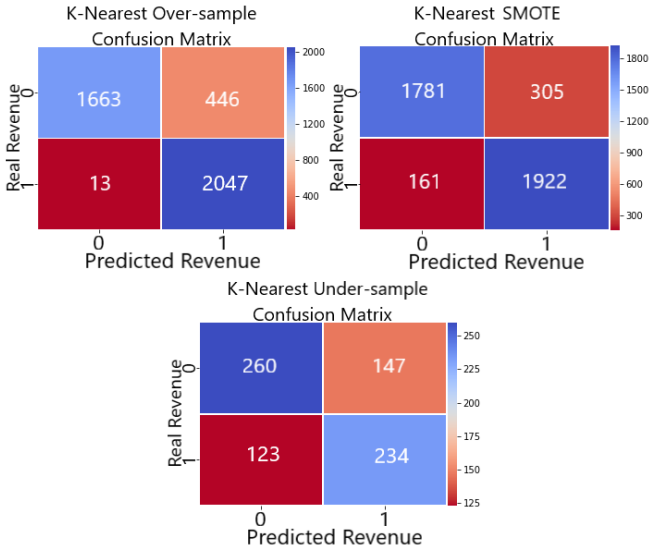

Fig. 10 Confusion Matrices K-Nearest Classifier

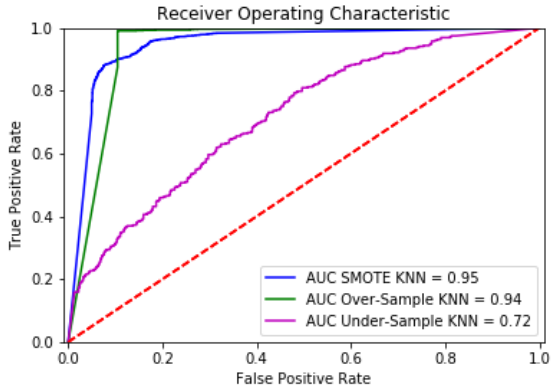

Fig. 11 ROC Curve K-Nearest

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.

C. *Random Forest*

Random forest algorithm from the scikit-learn repository in python was trained, tested and results obtained. Confusion matrices presented in *Fig. 12*, ROC curve in *Fig. 13* and results summary presented in *TABLE 6*.

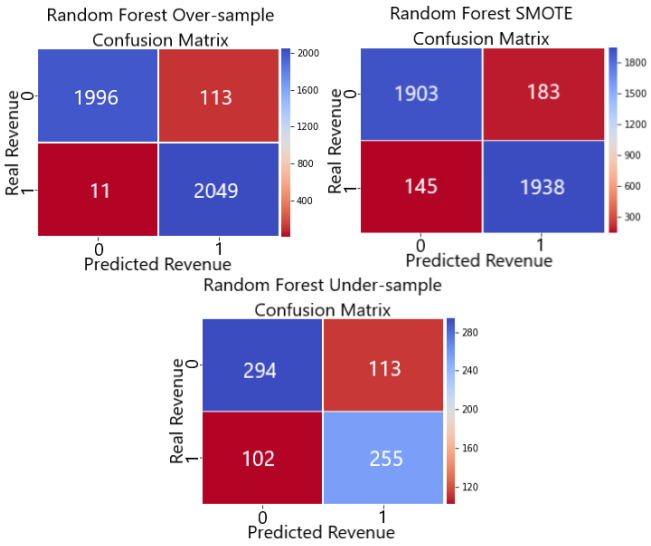

Fig. 12Confusion Matrices Random Forest

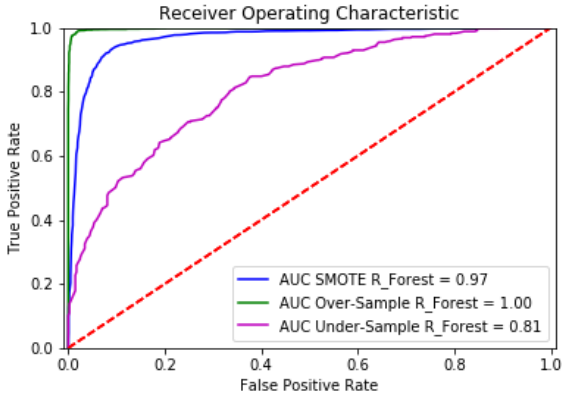

Fig. 13 ROC Curve Random Forest

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.

D. 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*, ROC curve *Fig. 15* and results summary in *TABLE 7*.

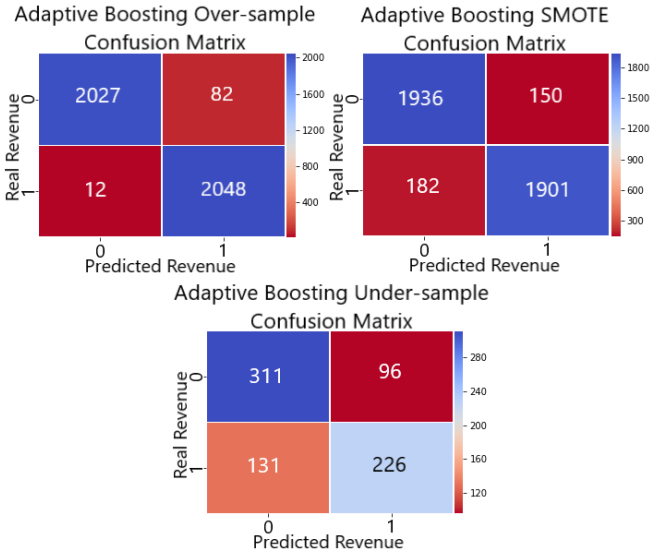

Fig. 14 Confusion Matrices Adaptive Boosting

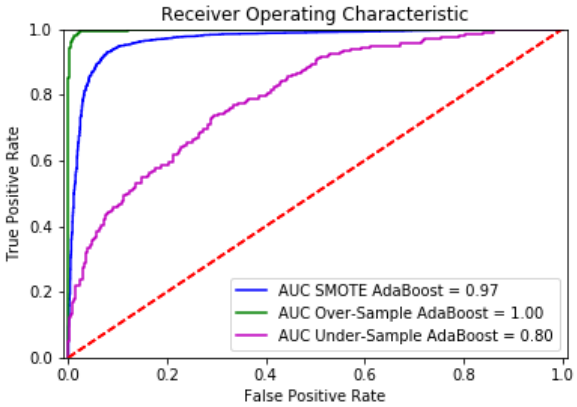

Fig. 15 ROC Curve Adaptive Boosting

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.

E. 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*, ROC curve in *Fig 17* and results summary in *TABLE 8*

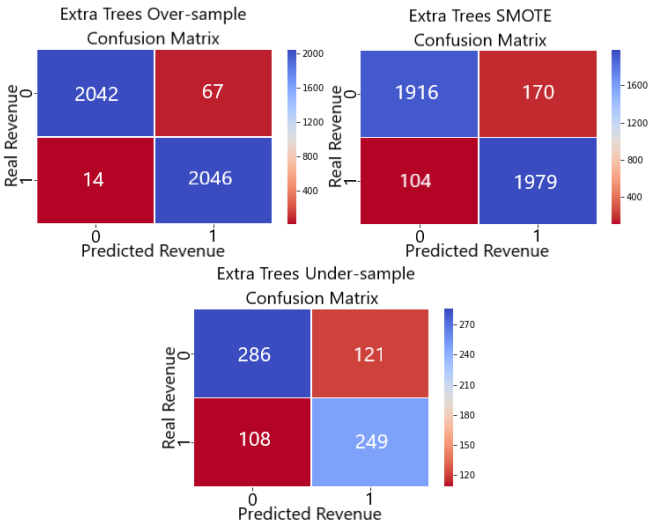

Fig. 16 Extremely Randomized Trees

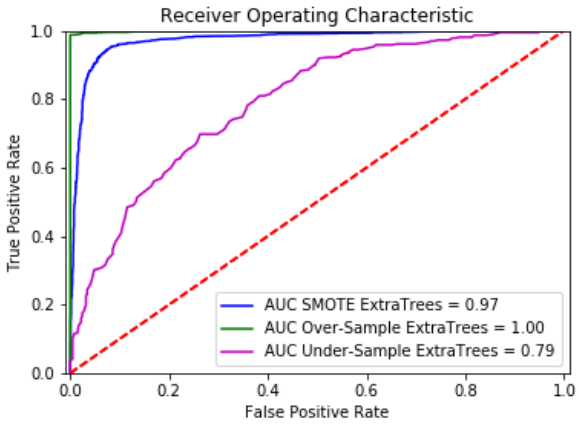

Fig 17 ROC Curve Extremely Randomized Trees

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.

VI. 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* 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.

VII. 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.

VIII. 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://public.eblib.com/choice/publicfullrecord.aspx?p=4388301> [7 December 2018]

Grąbczewski, 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://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-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-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.ensemble.AdaBoostClassifier.html> [13 December 2018]

Statista (2018) *Global Retail E-Commerce Market Size 2014-2021* [online] available from <https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/> [13 December 2018]

IX. Appendix

A. Source code

1) Libraries that were imported as part of the experiment

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import confusion_matrix,recall_score,precision_recall_curve
from sklearn.metrics import matthews_corrcoef, auc,roc_curve,roc_auc_score,classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from imblearn.over_sampling import SMOTE
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
import math
import time
```

2) Importing Dataset, Pot Class Imbalance and Prepare for Over/Under-sampling

```
data = pd.read_csv('online_shoppers_intention.csv')
#plot class imbalances
sns.countplot("Revenue",data=data)
data['Revenue'].replace(False, 0, inplace=True)
data['Revenue'].replace(True, 1, inplace=True)
Count_False_purchase = len(data[data["Revenue"]==0])
Count_True_purchase = len(data[data["Revenue"]==1])
true_indices= np.array(data[data.Revenue==1].index)
false_indices = np.array(data[data.Revenue==0].index)
```

3) Functions Defined and Used for Over/Under-Sampling

```
def undersample(false_indices,true_indices,times):#times denote the normal data = times*fraud data
    Normal_indices_undersample = np.array(np.random.choice(false_indices,(times*Count_True_purchase),replace=False))
    undersample_data= np.concatenate([true_indices,Normal_indices_undersample])
    undersample_data = data.iloc[undersample_data,:]
    print("the non-purchase proportion is :", len(undersample_data[undersample_data.Revenue==0])/len(undersample_data.Revenue))
    print("the purchase proportion is :",len(undersample_data[undersample_data.Revenue==1])/len(undersample_data.Revenue))
    print("total number of records in resampled data is:",len(undersample_data.Revenue))
    return(undersample_data)

def oversample(false_indices,true_indices,times):#times denote the normal data = times*fraud data
    Fraud_indices_oversample = np.array(np.random.choice(true_indices,(1*Count_False_purchase),replace=True))
    oversample_data= np.concatenate([false_indices,Fraud_indices_oversample])
    oversample_data = data.iloc[oversample_data,:]
    print("the false transacation proportion is :", len(oversample_data[oversample_data.Revenue==0])/len(oversample_data.Revenue))
    print("the true transacation proportion is :",len(oversample_data[oversample_data.Revenue==1])/len(oversample_data.Revenue))
    print("total number of record in resampled data is:",len(oversample_data.Revenue))
    return(oversample_data)

def smote(data1):
    data_test_X = data1[:,:-1]
    data_test_y = data1[:,-1:]
    os = SMOTE(random_state=0)
    os_data_X,os_data_y=os.fit_sample(data_test_X,data_test_y)
    os_data = np.column_stack((os_data_X, os_data_y))
    return(os_data)
```

4) Model and Model1 Functions Used for Training, Testing, Validation, and Metrics Calculation

```
def model(model,features_train,features_test,labels_train,labels_test, train, labels):
    clf= model
    clf.fit(features_train,labels_train.ravel())
    pred=clf.predict(features_test)
    cnf_matrix=confusion_matrix(labels_test,pred)
    print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[1,0]))
    plt.figure(1,figsize=(6,3))
    tpr1 = cnf_matrix[1,1,]
    tnr1 = cnf_matrix[0,0] # no. of normal transaction which are predited normal
    fpr1 = cnf_matrix[0,1] # no of normal transaction which are predicted fraud
    fnr1 = cnf_matrix[1,0] # no of fraud Transaction which are predicted normal
    print("TP",tpr1) # no of fraud transaction which are predicted fraud
    print("TN",tnr1) # no. of normal transaction which are predited normal
    print("FP",fpr1) # no of normal transaction which are predicted fraud
    print("FN",fnr1) # no of fraud Transaction which are predicted normal
    sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5,fmt='g')
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_Revenue")
    plt.ylabel("Real Revenue")
    plt.show()
    print("\n----------Revenueification Report--------------------------------------")
    print(classification_report(labels_test,pred))
    # This is the AUC
    probs = clf.predict_proba(features_test)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(labels_test, preds)
    plt.figure(2)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    #next three lines based on colour required comment or uncommend the relevan tine
    plt.plot(fpr, tpr, 'b', label = 'AUC SMOTE AdaBoost = %0.2f' % roc_auc)
    #plt.plot(fpr, tpr, 'g', label = 'AUC Over-Sample AdaBoost = %0.2f' % roc_auc)
    #plt.plot(fpr, tpr, 'm', label = 'AUC Under-Sample AdaBoost = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.01, 1.01])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
    print('========== metrics ============')
    print('AUC         :', roc_auc)
    cv = cross_val_score(clf, train, labels.ravel(), cv = 10)
    print('========== cross validation ============')
    print("Accuracy: %0.2f (+/- %0.2f)" % (cv.mean(), cv.std() * 2))
    accur = (tpr1 + tnr1)/(tpr1+fpr1+fnr1+tnr1)
    print('Accuracy          : ',accur)
    precis = tpr1/(tpr1+fpr1)
    print('Precission        : ',precis)
    sp = tnr1/(tnr1 + fpr1)
    print('Specificity       : ',sp)
    se = tpr1/(tpr1+fnr1)
    print('Sensitivity/Recall: ',se)
    f1 = 2*(precis * se)/(precis + se)
    print('F1 Score          : ',f1)
    gm = math.sqrt(se * sp)
    print('G Measure         : ',gm)
    mcc = matthews_corrcoef(labels_test, pred)
    print('Mathews Corr Coef : ',mcc)
    print('===============================')
    print('TP: ',tpr1,' | TN: ',tnr1)
    print('FP: ',fpr1,' | FN: ',fnr1)
    print('===============================')
    return(cnf_matrix,clf)
```

```
def model1(model,features_train,features_test,labels_train,labels_test, train, labels):
    clf= model
    clf.fit(features_train,labels_train.ravel())
    pred=clf.predict(features_test)
    cnf_matrix=confusion_matrix(labels_test,pred)
    print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[1,0]))
    plt.figure(1,figsize=(6,3))
    tpr1 = cnf_matrix[1,1,]
    tnr1 = cnf_matrix[0,0] # no. of normal transaction which are predited normal
    fpr1 = cnf_matrix[0,1] # no of normal transaction which are predicted fraud
    fnr1 = cnf_matrix[1,0] # no of fraud Transaction which are predicted normal
    print("TP",tpr1) # no of fraud transaction which are predicted fraud
    print("TN",tnr1) # no. of normal transaction which are predited normal
    print("FP",fpr1) # no of normal transaction which are predicted fraud
    print("FN",fnr1) # no of fraud Transaction which are predicted normal
    sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5,fmt='g')
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_Revenue")
    plt.ylabel("Real Revenue")
    plt.show()
    print("\n----------Revenueification Report--------------------------------------")
    print(classification_report(labels_test,pred))
    cv = cross_val_score(clf, train, labels.ravel(), cv = 10)
    print('========== cross validation ============')
    print("Accuracy: %0.2f (+/- %0.2f)" % (cv.mean(), cv.std() * 2))
    accur = (tpr1 + tnr1)/(tpr1+fpr1+fnr1+tnr1)
    print('Accuracy          : ',accur)
    precis = tpr1/(tpr1+fpr1)
    print('Precission        : ',precis)
    sp = tnr1/(tnr1 + fpr1)
    print('Specificity       : ',sp)
    se = tpr1/(tpr1+fnr1)
    print('Sensitivity/Recall: ',se)
    f1 = 2*(precis * se)/(precis + se)
    print('F1 Score          : ',f1)
    gm = math.sqrt(se * sp)
    print('G Measure         : ',gm)
    mcc = matthews_corrcoef(labels_test, pred)
    print('Mathews Corr Coef : ',mcc)
    return(cnf_matrix,clf)
```

5) Categorical Data encoding Function

```
def data_encoding(Undersample_data):
    labelencoder_X = LabelEncoder()
    Undersample_data['Month'] = labelencoder_X.fit_transform(Undersample_data['Month'])
    Undersample_data['VisitorType'] = labelencoder_X.fit_transform(Undersample_data['VisitorType'])
    Undersample_data['Weekend'] = labelencoder_X.fit_transform(Undersample_data['Weekend'])
    Undersample_data = Undersample_data.values
    onehotencoder = OneHotEncoder(categorical_features = [10,11,12,13,14,15])
    Undersample_data = onehotencoder.fit_transform(Undersample_data).toarray()
    Undersample_data = np.delete(Undersample_data,[0,10,18,31,40,60],axis=1)
    return (Undersample_data)
```

6) Splitting Data into Train and Test Sets Function

```
def data_prepration(x):
    x_features= x[:,:-1]
    x_labels=x[:,-1:]
    x_features_train,x_features_test,x_labels_train,x_labels_test = train_test_split(x_features,x_labels,test_size=0.2)
    print("length of training data")
    print(len(x_features_train))
    print("length of test data")
    print(len(x_features_test))
    return(x_features_train,x_features_test,x_labels_train,x_labels_test)
```

7) LDA, PCA, KPCA Functions

```
def lda(x_train, y_train, x_test, components):
    lda = LDA(n_components = components)
    y_train = y_train[:,0]
    x_train = lda.fit_transform(x_train, y_train)
    return (x_train, x_test)
#PCA function
def pca(x_train, x_test, components):
    pca = PCA(n_components = components)
    x_train = pca.fit_transform(x_train)
    x_test = pca.transform(x_test)
    explained_variance = pca.explained_variance_ratio_
    #print variance of features
    plt.plot(np.cumsum(explained_variance))
    plt.title('Resampled Data, all variables')
    plt.xlabel('number of components')
    plt.ylabel('cumulative explained variance');
    return (x_train, x_test, explained_variance)

#KPCA function
def kpca(x_train, x_test, cross_val_data, components, kernel):
    kpca = KernelPCA(n_components = components, kernel = kernel)
    x_train = kpca.fit_transform(x_train)
    x_test = kpca.transform(x_test)
    cross_val = kpca.transform(cross_val_data)
    ex_var = np.var(x_train, axis=0)
    explained_variance = ex_var / np.sum(ex_var)
        #print variance of features
    plt.plot(np.cumsum(explained_variance))
    plt.title('Resampled Data, all variables')
    plt.xlabel('number of components')
    plt.ylabel('cumulative explained variance');
    return (x_train, x_test, cross_val, explained_variance)
```

8) Running SMOTE Scenario

```
# data preparation
data1 = data_encoding(data)
smote_data = smote(data1)
smote_over_train,smote_over_test,smote_over_labels_train,smote_over_labels_test=data_prepration(smote_data)
#data_features_train,data_features_test,data_labels_train,data_labels_test=data_prepration(data1)

# feature normalisation
sc_X = StandardScaler()
smote_over_train = sc_X.fit_transform(smote_over_train)
smote_over_test= sc_X.transform(smote_over_test)

# k-fold cross validation on whole SMOTE data
cross_val_feat_smote = smote_data[:,:-1]
cross_val_label_smote = smote_data[:,-1:]
cross_val_feat_smote = sc_X.transform(cross_val_feat_smote)

#Kernel PCA pass (train data set, test data set, number of components, kernel used surounded by quotes)
smote_over_train, smote_over_test, cross_val_feat_smote1, kpca_rate_smote = kpca(smote_over_train, smote_over_test, cross_val_feat_smote, 38, 'rbf')

# Classifiers to be used
clf = SVC(C = 4, kernel='rbf', gamma = 0.3)
clf = KNeighborsClassifier(n_neighbors = 3, weights = 'distance', metric = 'minkowski', p = 2)
clf = DecisionTreeClassifier()
clf = RandomForestClassifier(n_estimators=110)
clf = ExtraTreesClassifier(n_estimators = 2000)
bs = RandomForestClassifier()
clf = AdaBoostClassifier(base_estimator = bs, n_estimators=123, algorithm = 'SAMME') #2023, 63 11,2072

# run classifier through models and capture time
start = time.time()
cnf,clf = model(clf,smote_over_train,smote_over_test,smote_over_labels_train,smote_over_labels_test, cross_val_feat_smote1, cross_val_label_smote)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)

# for SVM only
start = time.time()
cnf,clf = model1(clf,smote_over_train,smote_over_test,smote_over_labels_train,smote_over_labels_test, cross_val_feat_smote1, cross_val_label_smote)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)
```

9) *Running Random Over-sampling Scenario*

```
# use to crate data set oversampling positive class at random with replacement
over_data = oversample(false_indices,true_indices,1)
over_data = data_encoding(over_data)
#split data into train and test
over_data_train,over_data_test,over_labels_train,over_labels_test=data_prepration(over_data)

# scale data
sc_X2 = StandardScaler()
over_data_train = sc_X2.fit_transform(over_data_train)
over_data_test= sc_X2.transform(over_data_test)

# k-fold cross validation on whole oversample data
cross_val_feat_over = over_data[:,:-1]
cross_val_label_over = over_data[:,-1:]
#normalise same as training, test data
cross_val_feat_over = sc_X2.transform(cross_val_feat_over)

# run KPCA use none to find the results
over_data_train1, over_data_test1, cross_val_feat_over1, kpca_rate_over = kpca(over_data_train, over_data_test, cross_val_feat_over, 38, 'rbf')

#Classifiers to use
clf = SVC(C = 10, kernel='rbf', gamma = 1)
clf = KNeighborsClassifier(n_neighbors = 3, weights = 'distance', metric = 'minkowski', p = 2)
clf = RandomForestClassifier(n_estimators=123)
clf = ExtraTreesClassifier(n_estimators = 144)
bs = RandomForestClassifier()
clf = AdaBoostClassifier(base_estimator = bs, n_estimators=123, algorithm = 'SAMME') #2023, 63 11,2072

#pass data to classifier and get results
start = time.time()
cnf,clf = model(clf,over_data_train1,over_data_test1, over_labels_train, over_labels_test, cross_val_feat_over1, cross_val_label_over)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)
# for SVM only
start = time.time()
cnf,clf = model1(clf,over_data_train1,over_data_test1, over_labels_train, over_labels_test, cross_val_feat_over1, cross_val_label_over)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)
```

10) *Running Under-sample Scenario*

```
# Undersample data preparation
under_data = undersample(false_indices,true_indices,1)
under_data1 = data_encoding(under_data)

while (len(under_data1[1,:]) <= 68):
    under_data = undersample(false_indices,true_indices,1)
    under_data1 = data_encoding(under_data)

under_data_train,under_data_test,under_labels_train,under_labels_test=data_prepration(under_data1)
#Scale the data
sc_X3 = StandardScaler()
under_data_train = sc_X3.fit_transform(under_data_train)
under_data_test= sc_X3.transform(under_data_test)

# k-fold cross validation on whole oversample data
cross_val_feat_under = under_data1[:,:-1]
cross_val_label_under = under_data1[:,-1:]
#normalise same as training, test data
cross_val_feat_under1 = sc_X3.transform(cross_val_feat_under)

# run KPCA
under_data_train1, under_data_test1, cross_val_feat_under1, kpca_rate_under = kpca(under_data_train, under_data_test, cross_val_feat_under1, 38, 'rbf')

# Classifiers to be used
clf = SVC(C = 4, kernel='rbf', gamma = 1)
clf = KNeighborsClassifier(n_neighbors = 5, weights = 'distance', metric = 'minkowski', p = 2)
clf = DecisionTreeClassifier()
clf = RandomForestClassifier(n_estimators=145)
clf = ExtraTreesClassifier(n_estimators = 162)
bs = RandomForestClassifier()
clf = AdaBoostClassifier(base_estimator = bs, n_estimators=84, algorithm = 'SAMME') #2023, 63 11,2072

# running classifiers and validation results
start = time.time()
cnf,clf = model(clf,under_data_train1,under_data_test1,under_labels_train,under_labels_test, cross_val_feat_under1, cross_val_label_under)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)

#start = time.time()
cnf,clf = model1(clf,under_data_train1,under_data_test1,under_labels_train,under_labels_test, cross_val_feat_under1, cross_val_label_under)
elapsed_time = (time.time() - start)
print(" Time taken : ", elapsed_time)
```

11) Grid Search

```
parameters = {"n_neighbors":[3, 5], "weights": ['distance'], "metric":['minkowski'], "p":[1, 2]}
parameters = {"n_estimators":[125, 123]}
parameters = {"n_estimators":[87, 85, 90, 86, 84]}
parameters = {"n_estimators":[100, 110, 120, 150]}
grid_search = GridSearchCV(clf,
                           parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search = grid_search.fit(under_data_train1, under_labels_train.ravel())
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
```