Machine Learning

CSI 5155

Classification of Online Shoppers Purchasing Intention

Assignment 1

Authors
Alexandra Sklokin (300010511)

Professor
Dr. Herna Viktor

October 4, 2021

Contents

1 Introduction

The purpose of this assignment is to perform an exploration to find the best model for the classification problem on online shopping data. The target feature is a boolean variable *Revenue* which represents whether a client completes or abandons their online purchase. The classifiers to be trained are K-nearest neighbors (KNN), Support Vector Machine (SVM), Decision Tree, and Random Forest.

The following report documents my methodology, and results for this assignment. Sections marked with a letter, for example (a), correspond to a requirement in the assignment instructions.

2 Dataset

The *online_shoppers_intension* dataset contains 12,330 entries in 18 attributes (10 categorical and 7 numerical), one of which is the target feature *Revenue*. This is a unbalanced classification problem, as 84.5% (10,422) of the dataset is of the negative class (False), meaning that these clients did not complete their purchase. The remaining 1,908 entries belong to the positive class (True), indicating these users did complete their purchase.

3 Methodology

For this assignment I used *jupyter notebooks*, Excel spreadsheets, and the *sklearn Python* library.

Please find all of the code for this assignment in my git repository[1].

4 Preprocessing (a)

4.1 Cleaning the Data

In this dataset, none of the entries are *null* or unknown, therefore I did not need to clean the data. All 12,330 entries of the data were used for this experiment.

4.2 Feature Selection

I did not begin this assignment by feature transformations, as described in the assignment document, since 1-of-C coding of the categorical features would

[1] https://github.com/alexandrasklokin/CSI5155/tree/main/Assignment1

have increased the dimensionality of the dataset. Instead, I began by performing feature selection, and then was able to proceed with feature transformations.

4.2.1 SelectKBest

My first step was to use *SelectKBest* from the *sklearn* library to find the best ten features. The following features were selected by the algorithm (in no particular order):

- *Administrative, Administrative_Duration, Informational Informational_Duration, ProductRelated, ProductRelated_Duration, PageValues, SpecialDay, Month, VisitorType*

4.2.2 Correlation Matrix

I proceeded by using the correlation matrix (refer to Table 1) to determine which pairs of features were most correlated. I found very high correlation (absolute values greater than 0.5) between the following pairs of features:

- *ExitRates* and *BounceRates* (0.91)
- *ProductRelated* and *ProductRelated_Duration* (0.86)
- *Administrative* and *Administrative_Duration* (0.6)
- *Informational* and *Informational_Duration* (0.62)

Again using the correlation matrix, I now ranked the features, based on which were most correlated to the target feature *Revenue*, but only selecting one of the two features from the above pairs of correlated features. For example, *ExitRates* was more correlated to *Revenue* than *BounceRates*, therefore *ExitRates* was included in the list of best features and *BounceRates* was not. The following were the best features selected (ordered from most correlated to target features to least):

- *PageValues, ExitRates, ProductRelated, Administrative, VisitorType, Informational, SpecialDay, Month, Weekend, Browser, OperatingSystems, Region, TrafficType*

4.2.3 Dimension Reduction

It turns out that the lists, generated from *SelectKBest* and the correlation matrix method, have some elements in common. The *SelectKBest* features include pairs of highly-correlated features, which I did not want to include for training due

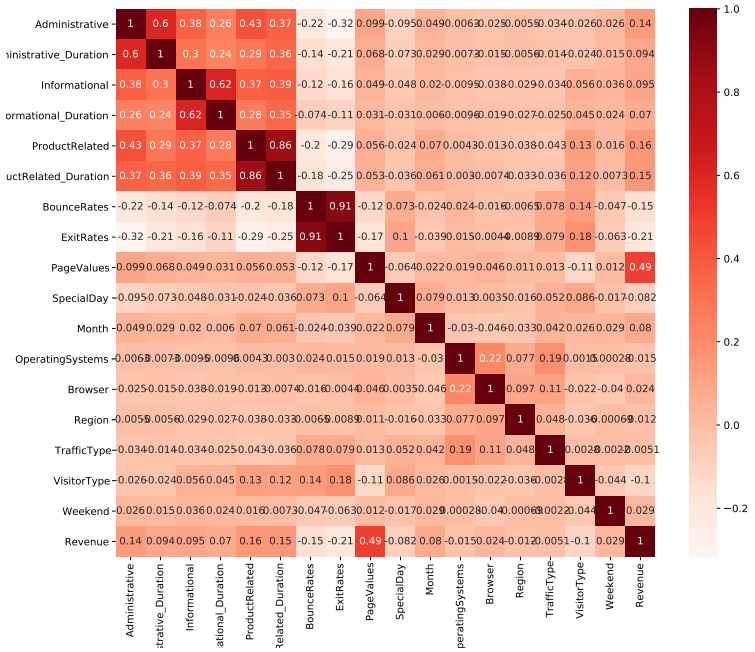


Figure 1: Correlation Matrix.

to the multicollinearity conundrum. Instead, I used the features selected from the correlation matrix. Now I could proceed to further dimension reduction.

Using some preliminary testing, I determined that the Random Forest classifier achieved the highest F1-score for non-prepossessed data, therefore this could be used to select the number of features to retain for training. The highest F1-Score was achieved with the first 8 features (refer to Figure 11).

Thus, at the end of the full feature selection process, I have selected the following 8 features:

- *PageValues, ExitRates, ProductRelated, Administrative, VisitorType, Informational, SpecialDay, Month*

4.3 Feature Transformations

For the purpose of this experiment, I used the two methods proposed by the Sakar's paper for feature transformations [1].

For categorical features (features with discrete or non-numerical values), I used 1-of-C coding, where each categorical feature is transformed into multiple

features, one for each unique entry in the original feature. For example, if feature *A* has entries *a,b,c,b*, then the new feature set would be *A_a, A_b, A_c*. The original feature *A* is removed from the new feature set.

For the target feature *Revenue*, I encoded *True* entries to 1, and *False* to 0, so that I could use tools which required numerical based (not string) values.

For numerical features (features with continuous values), I standardized the entries. The entries are scaled so that for each feature the mean of all entries is equal to 0.0, and the standard deviation is 1.0.

4.4 Preprocessed Data

The data, following all of the preprocessing, can be found in the *preprocessed-Data.csv* Excel sheet, in my git repository.

5 The Models (b)

5.1 Training

As suggested in the assignment, I trained the models with a data split of 70% (8631) entries for training, and 30% (3699) testing. I used a stratified training-testing split to make sure that the training and testing sets also contained the unbalanced distribution of negative to positive classes. I only used the 8 best features for both training and hyperparameter tuning.

5.2 Hyperparameter Tuning

I began with hyperparameter tuning. This is done to determine the best parameters in each of the classifier to use for the training step. It was determined that the best cross-validation results were obtained from the following parameter values:

- **KNN**

  metric: 'manhattan', n_neighbors: 15, weights: 'uniform'

- **SVM**

  kernel: 'linear'

- **Decision Tree**

  criterion: 'gini', max_depth: 4, min_samples_leaf: 3, min_samples_split: 2

- **Random Forest**

  bootstrap: True, max_depth: 10, max_features: 8, min_samples_leaf: 5, min_samples_split: 8, n_estimators: 100

5.3 Resampling

This dataset is highly imbalanced. The majority class is *False* and minority is *True*. Two methods of resampling were performed for this assignment. Oversampling is when the minority class is resampled to match the number of occurrences of the majority class. Undersampling is when we sample from the majority class, and only take as many entries as in the minority class.

Resampling should be done only in the training set, as in practice, unseen data might follow the observed unbalanced distribution. The testing data should be representative of the true population, thus it is left out during resampling.

In general, resampling has two major issues. Oversampling the minority class can lead to overfitting, since certain entries of the original data are oversampled. Undersampling the majority class can result in underfitting, as the model is being trained with less of the provided data.

5.4 Evaluation

Since we are dealing with an unbalanced classification problem, I will pay the MOST attention to the average weighted F1-Score, and the average AUC-PR.

The following describe all of the possible metrics I could use for evaluating classification models. It is important to note that weighted statistics take into consideration the distribution of negative to positive in the target feature. Since the test set is imbalanced, we would like to consider weighted statistics, rather than macro statistics. All statistics are averaged over 10 models for each classifier. ROC and PR curves are produced for models with the highest weighted F1-score, for each classifier.

1. **Accuracy** is the proportion of correctly made predictions. This metric provides a good summary of performance over both binary classes, however may not provide good evidence when in the unbalanced case. This may not be a good metric for the imbalanced classification task.

2. **Precision** describes the proportion of predictions which were actually correct, for each label.

3. **Recall** describes the proportion of the actual responses which are identified correctly, for each label.

   Precision and Recall are often 'counter' one another. When we attempt to improve precision, we might negatively affect recall, and vice versa. If the model is liberal (ie. covers more samples), then you will have a high recall, but lower precision. If you have a conservative model (ie. only makes very sure predictions), then precision will be high, but recall will be low. Therefore, these two metrics should be consider together.

4. **F1-Score** attempts to brings together the precision and recall, for each sample:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{1}$$

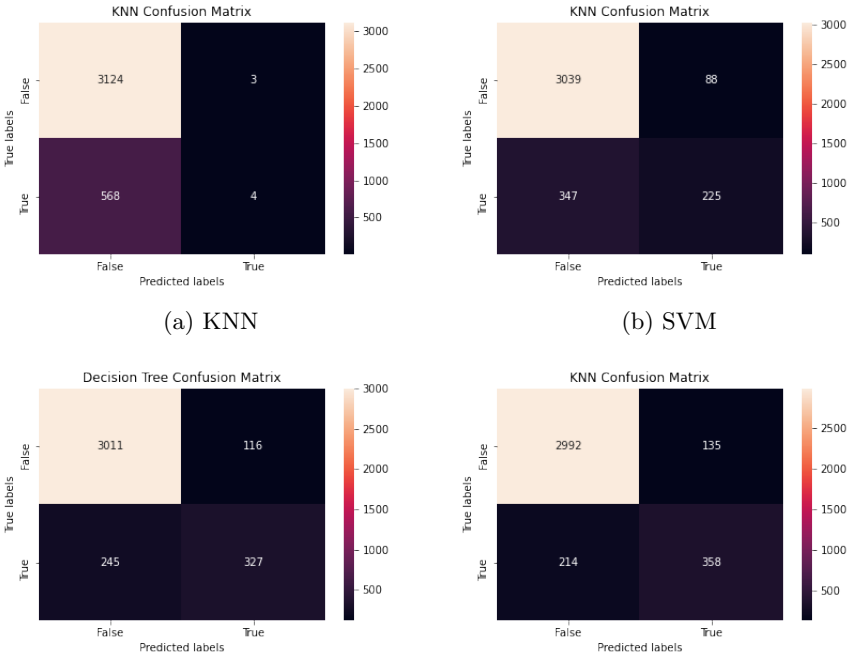
   This is the harmonic average of the Precision and Recall, therefore it gives equals weights to both. This metric will come in handy for the unbalanced problem, where accuracy will not be good evidence.

5. **AUROC** or Area under ROC curve measure the degree of separability. That is, how capable a model is in distinguishing between binary response classes. We would like the AUROC to be closer to 1.0. If AUROC is around 0.5, then the model is not significantly better than random guessing, and under 0.5 is a worse model. AUROC is not perfectly suitable for the imbalanced case, where every wrong prediction makes a significant difference to the ROC curve.

6. **AUC-PR** or Area under Precision-Recall curve measure. This can be better suited for the imbalanced problem.

6 Experimental Results for Imbalanced Case

6.1 Confusion Matrices and F1-Scores (c)



(a) KNN (b) SVM

(c) Decision Tree (d) Random Forest

Figure 2: Confusion Matrices for Classifiers (with highest weighted F1-Score)

Model	Accuracy	F1-Score		Recall		Precision		AUROC	AUC-PR
		Macro	Weighted	M	W	M	W		
KNN	0.8443	0.4615	0.7751	0.5009	0.8442	0.5680	0.7597	0.6038	0.2148
SVM	0.8782	0.7038	0.8608	0.6660	0.8782	0.8019	0.8645	0.7789	0.5114
DT	0.8939	0.7810	0.8895	0.7630	0.8939	0.8095	0.8890	0.9209	0.7027
RF	0.9031	0.8010	0.8994	0.7810	0.9031	0.8268	0.8979	0.9333	0.7532

Table 1: Average Scores over 10 Models per Classifier

6.2 ROC and PR Curves (d)

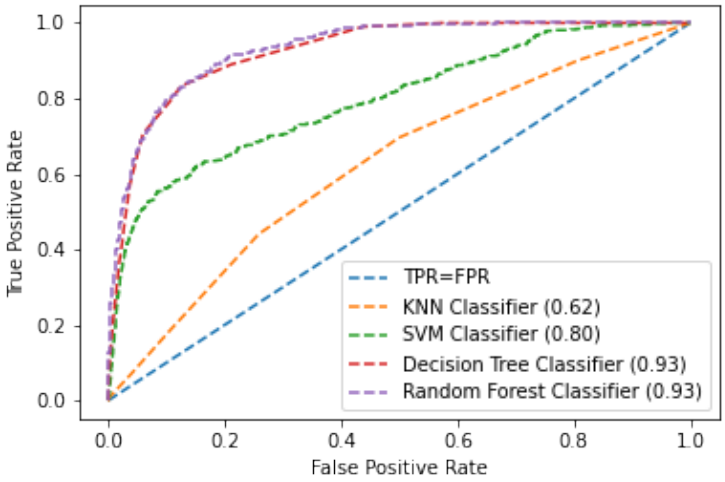


Figure 3: ROC Curve for Classifiers (with highest F1-Score)

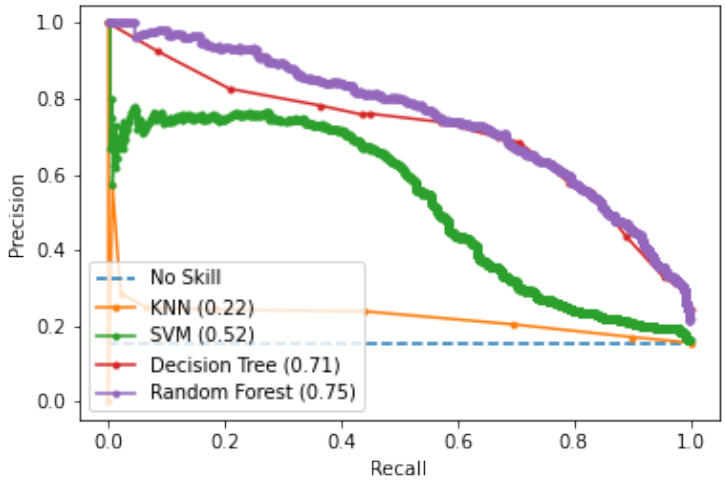
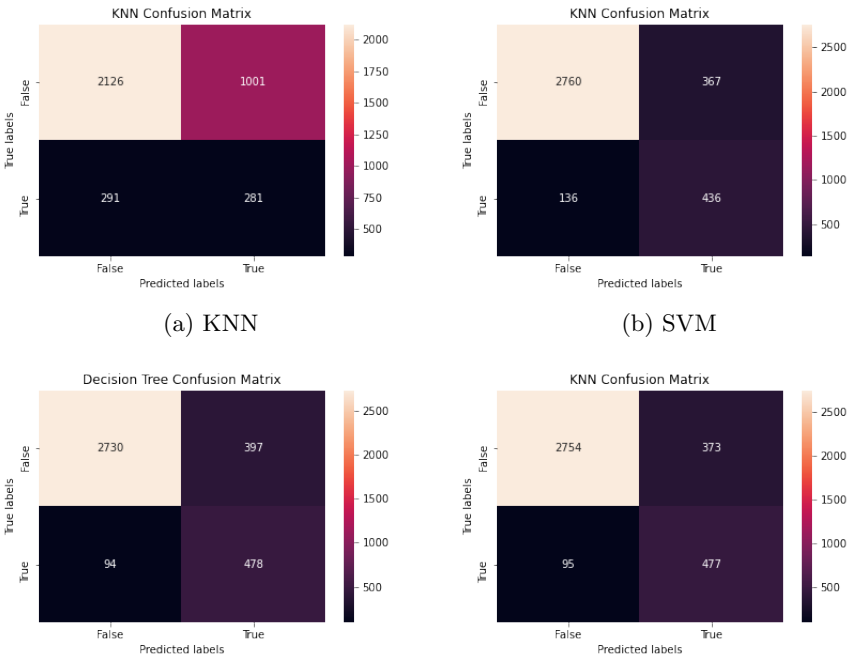


Figure 4: Precision-Recall Curve for Classifiers (with highest F1-Score)

7 Experimental Results from Oversampling (e)

7.1 Confusion Matrices and F1-Scores (c)



(a) KNN (b) SVM

(c) Decision Tree (d) Random Forest

Figure 5: Confusion Matrices for Classifiers (with highest F1-Score), with Oversampling

Model	Accuracy	F1-Score		Recall		Precision		AUROC	AUC-PR
		Macro	Weighted	M	W	M	W		
KNN	0.6401	0.5252	0.6868	0.5741	0.6406	0.5424	0.7723	0.6051	0.2242
SVM	0.8424	0.7501	0.8549	0.8068	0.8424	0.7240	0.8805	0.8878	0.6182
DT	0.8547	0.7747	0.8674	0.8471	0.8547	0.7427	0.8971	0.9170	0.6891
RF	0.8690	0.7883	0.8786	0.8465	0.8690	0.7573	0.8992	0.9323	0.7449

Table 2: Average Scores over 10 Models per Classifier, with Oversampling

7.2 ROC and PR Curves (d)

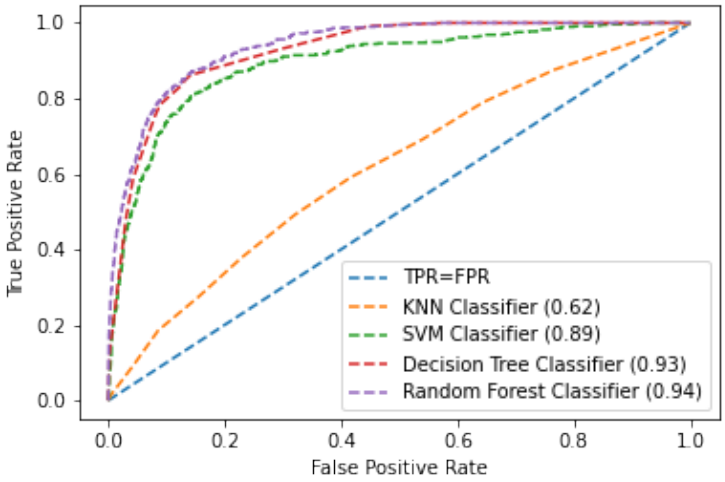


Figure 6: ROC Curve for Classifiers (with highest F1-Score), with Oversampling

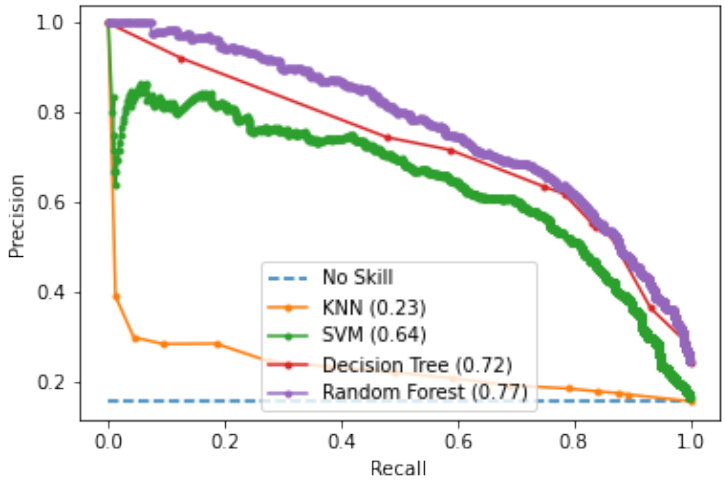
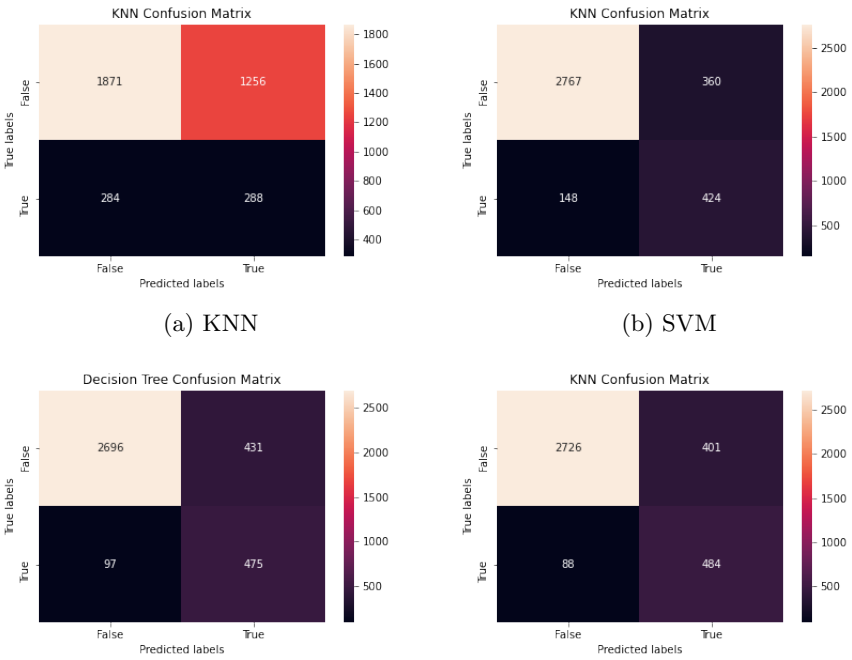


Figure 7: Precision-Recall Curve for Classifiers (with highest F1-Score), with Oversampling

8 Experimental Results from Undersampling (f)

8.1 Confusion Matrices and F1-Scores (c)



(a) KNN (b) SVM

(c) Decision Tree (d) Random Forest

Figure 8: Confusion Matrices for Classifiers (with highest F1-Score), with Undersampling

Model	Accuracy	F1-Score		Recall		Precision		AUROC	AUC-PR
		Macro	Weighted	M	W	M	W		
KNN	0.5651	0.4850	0.6251	0.5602	0.5651	0.5318	0.7682	0.5851	0.1980
SVM	0.8507	0.7584	0.8615	0.8102	0.8507	0.7321	0.8830	0.8858	0.6284
DT	0.8453	0.7660	0.8600	0.8470	0.8453	0.7347	0.8962	0.9171	0.6789
RF	0.8570	0.7798	0.8699	0.8567	0.8570	0.7468	0.9011	0.9334	0.7346

Table 3: Average Scores over 10 Models per Classifier, with Undersamping

8.2 ROC and PR Curves (d)

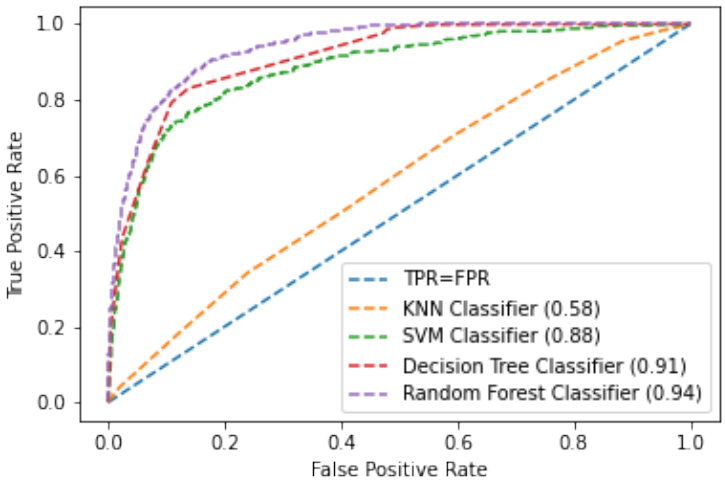


Figure 9: ROC Curve for Classifiers (with highest F1-Score), with Undersampling

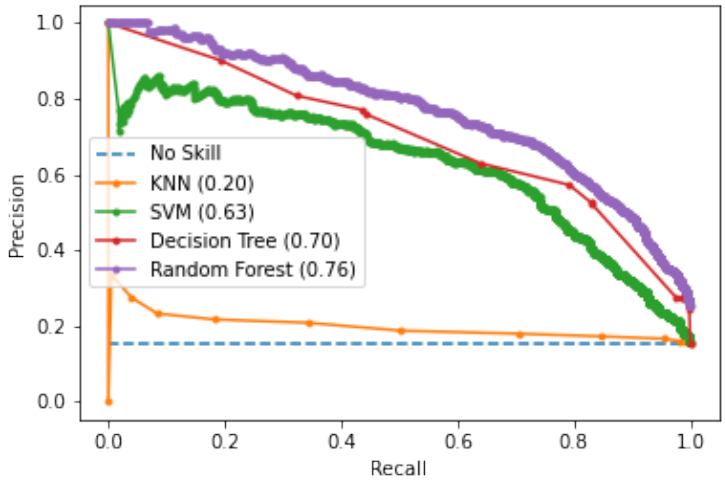


Figure 10: Precision-Recall Curve for Classifiers (with highest F1-Score), with Undersampling

9 Discussion

9.1 Best Models (g)

In each of the three experiments (no resampling, oversampling, and undersampling), the **Random Forest** models had the best performance. Referring to Tables 1, 2, and 3, we see that the Random Forest classifier outperformed all of the other models, in virtually all of the metrics. Following from the ROC curves (Figures 3, 6, and 9) and the PR curves (Figures 4, 7, and 10), the Random Forest has the highest AUROC and AUC-PR values.

The second best model was the Decision Tree. These results were very close but just bellow those for Random Forest. With a bit more restrictive rounding, we could say that the Decision Tree and Random Forest have basically the same performance, in all of the metrics.

The next best classifier is the SVM classifier. Although the performance is very comparable to that of the Decision Tree and Random Forest, the SVM classifier took hours to train, whereas all other models took minutes (refer to Table 4). This information is not negligible, especially in the case of needing real-time training and predictions, as in the reference paper. For this reason, I would not consider this model the best.

The worst classifier, by far, was the KNN algorithm. Even after parameter tuning, this algorithm may be too simple for this classification problem, resulting in underfitting.

	KNN	SVM	Decision Tree	Random Forest
Parameter Tuning	1 min	10 mins	1 min	2 mins
Training (x10)	1 min	3 hrs	<1 min	2 mins
Training (x10, with oversampling)	1 min	5 hrs	<1 min	5 mins
Training (x10, with undersampling)	1 min	2 hrs	<1 min	1 mins

Table 4: Approximation of Time Taken for Different Algorithms. Training for 10 models per classifier.

9.2 Comparison of Results Before and After Balancing (g)

In general, the ranking of the models did not change before and after resampling. The ranking remains (from best to worst): (1) **Random Forest**, (2) Decision Tree, (3) SVM, and (4) KNN, for oversampling and undersampling.

The best metrics were achieved when the training set was not resampled

upon (refer to Table 5). There was not an improvement in any of the metrics from no resampling to resampling. This may be due to overfitting or underfitting when resampling. The worst results came from undersampling, which is to be expected, since there is less data in the training set for the model to learn from.

Resampling	Best Model	Accuracy	F1-Score		AUROC	AUC-PR
			Macro	Weighted		
None	Random Forest	0.9031	0.8010	0.8994	0.9333	0.7532
Oversampling	Random Forest	0.8690	0.7883	0.8786	0.9323	0.7449
Undersampling	Random Forest	0.8570	0.7798	0.8699	0.9334	0.7346

Table 5: Best Classifiers, with and without Resampling

10 Comparison with Reference Paper (h)

Here I will compare the methods and results of my experiment to that of Sakar's paper[1].

10.1 Feature Selection

In Sakar's paper, the authors also used a filter-based feature selection. Using the mRMR statistic, which aims to maximize the relevance between the selected set of features while avoiding redundancy, they ranked the features. This is similar to my correlation matrix method, where I ranked the features based on their correlation to the target feature. My ranking matches the ranking in the 'Correlation' column of their Table 10. I also attempted to reduce redundancy when selecting my top 13 best features, by removing pairs of highly correlated features.

The authors proceed to reduce the feature space by feeding the top 10 features selected by the mRMR filter into a Multi Layer Perceptron (MLP) model, using oversampled data. At the end, they find that including the top 6 features gives the highest possible accuracy and F1-Score:

- *PageValue, Month, ExitRates, Weekend, Informational duration, Region*

Although our final list of features are not identical, my list does contain their top three features. In comparison, I fed my ranked features into a RandomForest algorithm, and obtained these top 8 features:

- *PageValues, ExitRates, ProductRelated, Administrative, VisitorType, Informational, SpecialDay, Month*

10.2 Feature Engineering

As specified in the assignment instructions, I followed Sakar's procedure for feature engineering: 1-of-C coding of categorical features, and standardization of numerical features. Although they did not provide examples of their transformed features, I would expect our results to be similar.

10.3 Algorithms and Resampling

In Sakar's paper, the authors used three main algorithm types: tree-based (Decision Tree and Random Forest), Multi Layer Perceptron, and SVM. They used different hyperparameters for each group of classifiers and obtained accuracies and F1-scores. For each classifier, their training is repeated 100 times with random training/validation partitions.

In my assignment, I used four classifiers: KNN, SVM, Decision Tree, and Random Forest. For each classifier, my training is repeated 10 times with random training/validation partitions. I used hyperparameter tuning on preprocessed data to select the best classification parameters.

For resampling, we both used oversampling of minority class in the training set, while withholding the testing set, to balance the data. Additionally, I attempted undersampling of the majority class.

10.4 Results

In the paper, the best F1-score for no resampling and oversampling are obtained with the MLP, with 20 and 10 neurons in the hidden layer, respectively (refer to Table 6). The authors see significant improvement of the F1-Score after resampling, in all the classifiers.

Referring to Table 5, I obtained a higher F1-Score than the authors with the Random Forest algorithm than their MLP, for the no resampling case. I obtained slightly higher results with the Random Forest model for the oversampling case as well. There are many factors which could have contributed to this, such as the feature selection (our feature sets were different), the preprocessing methods, and our hyperparameters.

The authors did not do undersampling, therefore those results could not be compared.

Model	Parameters	Accuracy	F1-Score
No Resampling			
MLP	#neurons=20	0.8792	0.58
Tree-Based	Random Forest	0.8951	0.58
SVM	kernel=rbf	0.8614	0.53
Oversampling			
MLP	#neurons=10	0.8724	0.86
Tree-Based	C4.5	0.8234	0.82
SVM	kernel=RBF	0.8488	0.82

Table 6: Sakar's Best Classification Results[1]

11 Conclusion

In conclusion, I was able to get good results for this classification problem. First the feature set was preprocessed through feature selection, and feature transformations. Then, I used parameter tuning to select the best hyperparameters for training. Following training, I obtained different metrics for comparing the models, as well as the ROC and PR curves. I also attempted resampling methods to see if this would result in better predictions. In each case (no resampling, oversampling, and undersampling), the Random Forest classifier had the best metrics. Resampling did not improve my F1-scores. My classification results were very slightly higher than those obtained in Sakar's paper.

12 References

[1] C. Okan Sakar et al. "Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks". In: *Neural Computing and Applications* (May 2018). URL: https://link.springer.com/article/10.1007/s00521-018-3523-0.

A Feature Selection

Figure 11: F1-Scores of Random Forest Classification, for the purpose of feature selection.