Handling Class Imbalance with oversampling for predicting annual income: Duplication vs. synthetic data

Manar Attar (2631465), Majd Al Ali (2659280), Imane Akhyar (2667107), Duaa Ashtar (2696234) and **Raihan Karim Ishmam (2694796)**

Abstract

According to Pew Research Center, 71% of the world population has low or poor income(1). This huge imbalance in income distribution makes it hard for machine learning algorithms to make accurate predictions. In this paper, we decided to train different classifiers using an imbalanced dataset and optimize performance by attempting to balance the data. The aim is to predict the income, for which we use a dataset from the U.S. Census Bureau(2). To deal with the imbalance, two variants of oversampling: random oversampling (duplication) and synthetic oversampling (SMOTE) are compared to see which method performs better for different classifiers. the comparison is done by examining performance metrics such as f1-score and roc-AUC score. We observe an improvement for these scores using both oversampling methods, with slight advantage when using SMOTE-oversampling.

Keywords: *Class Imbalance, Predicting Income, Deep Learning, Classification Algorithms, Machine Learning Models, Decision Trees, Random Forest, Support Vector Machine, Naive Bayes, K-Nearest Neighbours, Logistic Regression*

1 Introduction

1.1 Motivation

One of the significant elements of our daily lives is our financial activity and status. From travel to food to shelter, all are highly dependent on and run by our finances (payments, rents, bills, etc.). Moreover, the most common source of providing this financial support is their income if we focus on adults. This makes income prediction an area of interest for many organizations and researchers. One of the most reliable datasets that we can use, including the data relevant to the income, is the census dataset. To find a dataset of this type, we turned to two very well-known web repositories: Kaggle and the UCI - Machine Learning Repository . We initially aimed to find our datasets from the ongoing competitions in Kaggle, but we could not find an appropriate one matching our project. So we moved to the UCI repository, where we found a data set that contains weighted census data extracted from the 1994 and 1995 population surveys conducted by the U.S. Census Bureau(2). The data set contains demographic and employment-related variables, which would provide us with a set of reasonably relevant features.

1.2 Paper outline

The essential element of this project is using machine learning to predict the $annual$ income of the various instances (adults). We decided to approach the problem as a classification problem rather than a regression one, as this would let us focus more on the statistical aspects of the performance. Choosing regression might have led to more detailed predicted values for the income, but that might introduce issues with the overfitting when we try to improve the accuracy, so we opted to go with classification. The machine learning classifier that we are developing is designed to do binary classification, again due to the reasons above. We chose to go with a specified value of (50,000 USD) to split the annual income range into two classes, so the instances are classified as, with an annual income either greater than 50,000 USD or less.

The dataset that we are using is comparatively significant, with around 32,500 instances. We are planning to use several different training models like K - Nearest Neighbours, Logistic Regression, Support Vector Machines, etc., to train our machine learning classifier to compare their performance on the training the data. As we have a significant imbalance in the data, which will be explained in the later section, we mainly aim to balance the training data and observe the outcome of the change in the classifier's performance on the test data.

2 Data inspection

The primary dataset consists of 17 classes, including the target class(Income), 32560 instances/rows. The target class **Income** has a remarkable class imbalance regarding the number of instances, where:22654 belongs to the low-income class($\leq 50k$ as annual income) vs. the high-income class 7508($> 50k$ as annual income). Since the data originate from the USA, most instances in the native country class are the instance the USA. The Sex class is binarily split into Male and Female, and the size of the Male value is twice as large as the Female value.The age class included in this dataset ranged between 17 and 90, even though the dataset covers adults' income.There are two classes about the education levels($Education\&Education-num$). Both cover 16 levels starting at preschool and ending at the doctorate. The only main difference between those two classes is that Education data is categorical and Education-num is numerical.

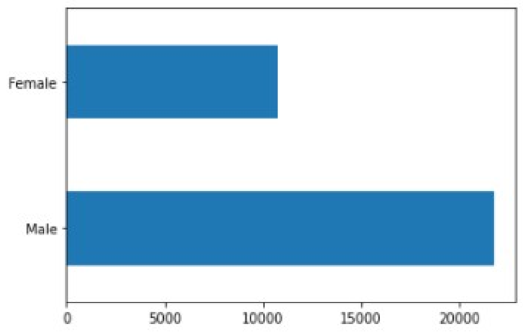


Figure 1: Male and Female distribution

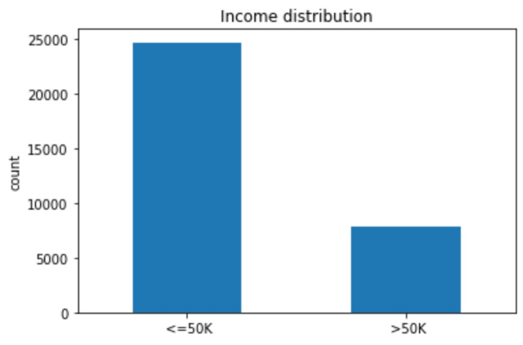


Figure 2: Low-income class and high-income class distribution

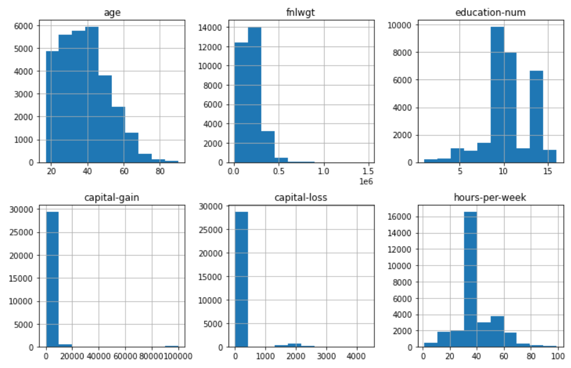


Figure 3: distributions all numerical features

3 Research Question

Other income predictors out there(3)(4) use machine learning algorithms to predict the income in various contexts, so the research prospect is directed on improving the classifier's performance. Something familiar in most real-world datasets is huge class imbalances, which makes it a primary area of interest. Generally, undersampling is an excellent method to deal with is and is commonly used. However, data samples are not very big (5), so undersampling is no longer a viable option. The left option is oversampling, which the research will focus on. There are different variants of oversampling. This paper compares the performance of two in dealing with class imbalance: $random\ oversampling$ and $SMOTE$, leading to optimization of the overall performance. This leads to the research question is:

How does generating corrupt data (SMOTE) compare to duplicating (random oversampling) in dealing with a class imbalance to improve the overall performance of a machine earning classifier?

4 Data preprocessing

4.1 Missing values

Following the data inspection, we noticed some instances with missing values, most likely due to missing inputs or other errors in the data collection stage of creating the dataset. We moved into the data preprocessing stage of the project, planning to deal with the missing values. There were around 2000 instances with missing values out of 32,560, which roughly forms 6$ of the entire database. The missing values belong to three out of the 16 classes we have: the $person'sworkclass$, $occupation$, and $nativecountry$. Despite the missing data being from only three classes, the deletion of these classes would result in a reduced number of features giving us less emphasis on the data and its properties. This might also lead to the deletion of a feature crucial to the prediction. However, there was a relatively high number of instances, so all the instances with the missing values were removed.

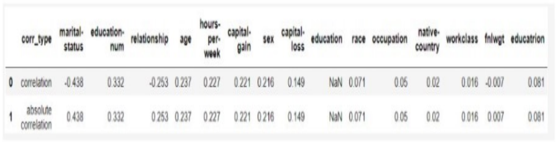
4.2 Sorting the data

The classifier's target is to predict the income, so the target class $income$ was parted from the rest of the classes, which would be forming the features. The features are split into $categorical$ and $numerical$ data to deal with them accordingly. The $numerical$ data were scaled to 1-0 using $min - maxscaling$ to have consistency in all the numerical classes. Initially, an attempt was made to map the $categorical$ into numerical data by assigning them to discrete numerical values. However, a ranking problem occurred upon mapping as not all of the categorical data is ordinal (data that can be ranked based on their properties). There are some nominal data too. One-hot encoding was applied to quantify the categorical data in a non-ordinal way to solve this. One-hot encoding converts the class into a matrix with each distinct category in the class being a column, which eventually leads to each category being a class/feature with 1s if an instance is of that category and 0s for the rest of the categories. The data is now good to be fed into a machine learning classifier.

4.3 Feature selection

The data is now processed and ready, but the features selection is still needed as not all features are relevant, which might cause the classifier to become more lenient to misclassification. We removed the $nativecountry$ and the $race$ before proceeding to the correlation perspective as they bring no added value to the classification. The reason behind it is that as the data is from the U.S. Census Bureau, there is an overwhelming majority from the U.S., which might lead to having a negative effect on the learning of our classifier. Our choice is also backed up from further research on the relevancy of these features for income prediction using $ExtraTreeClassifier$ where $nativecountry$ and $race$ are found as the least relevant features(6).

The correlation plot can give a good overview of the feature's relevancy with the target class and the features themselves. However, the focus will primarily be on correlating the individual features against the target class. There could also be a negative correlation. We decided to take the absolute value of the comparison to rank the relevancy based on how high the correlation value is.



	corr_type	marital-status	education-num	relationship	age	hours-per-week	capital-gain	sex	capital-loss	education	race	occupation	native-country	workclass	fnlwgt	educatrion
0	correlation	-0.438	0.332	-0.253	0.237	0.227	0.221	0.216	0.149	NaN	0.071	0.05	0.02	0.016	-0.007	0.081
1	absolute correlation	0.438	0.332	0.253	0.237	0.227	0.221	0.216	0.149	NaN	0.071	0.05	0.02	0.016	0.007	0.081

Figure 4: Correlation results

The correlation plot was plotted with the classes being sorted based on the highest absolute correlation value. To ensure the machine learning classifier can focus on the important features and not get disturbed by the un-related features, we narrowed our feature space to the six most relevant features. The final chosen features excluding the target class are $maritalstatus$, $educational$, $relationship$, age, $hours$, and $gain$.

5 ScikitLearn Models And Hyperparameters Optimization

5.1 K Nearest neighbor (KNN)

The K- Nearest Neighbours is a non-parametric data classification method used to solve classification and regression models. For any unclassified instance **p** , the algorithm considers the k number of closest instances around **p**. Then, the instance **p** is classified with the class with the highest frequency among the k nearest instances chosen. The k-nearest-neighbor is an example of a "lazy learner" algorithm because it does not generate a data set model beforehand. However, it is simple and yet very effective in many cases. The success of classification is very much dependent on the value of k, which can be determined by running the algorithm many times with different k values and choosing the one with the best performance(7) and is the approach we are going to adapt.

5.1.1 Optimizing KNN

For obtaining the best k value for KNN, K-values between 1-60 were tested. The best f1-score was for the k=22 as demonstrated by the graph below.

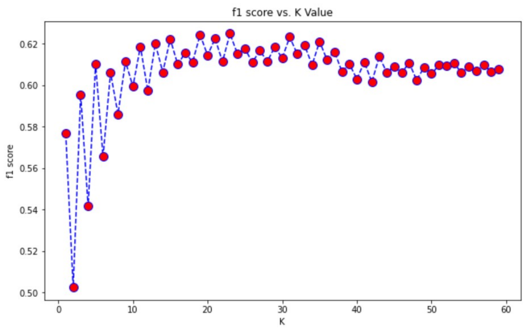


Figure 5: Test results for the K-values

5.2 Decision trees

A decision tree represents a recursive pattern of the instance space and is one of the most popular approaches for representing classifiers. The idea is simply an inverted tree staring from a *root* with no incoming connection, but all other nodes have precisely one incoming connection. The nodes may have two or more outgoing connections, and at each of them, a decision is made utilizing the dataset's features contributing to a better-formed classification. Decisions are made from a starting point, the root of the tree, working its way through the nodes with the connections being the attribute values. The classification is finally done when the algorithm has reached a leaf of the tree (8). The *leaves* are the dead-ends where the node has no outgoing connections. There are multiple variants of the decision trees, from which we chose to work with the $CART$ algorithm as $ScikitLearn$ uses this method for its decision trees, and we are circulating Scikit learn for this project.

5.3 Random forest

Random forest is a supervised learning algorithm that deals with multi-class problem(9). It randomly creates and merges multiple random decision trees into one "forest.". One of the most significant advantages of the random forest is that it is less sensitive for the training data than decision trees. The decision tree often fails to generalize, whereas the random forest doesn't. A new dataset has to be built from the original data of the same size(9). That could be done by Bagging(Bootstrap Aggregation), randomly sampling from the data set with replacement. To train decision trees on the new datasets, we randomly select a subset of features for each tree and use them for the training; this will lower the correlation between trees and cause more variation. For the final predictions, there are different approaches. Each tree uses a local majority vote for classification problems, whereas, in regression problems, the forecast is accomplished by a local averaging.

5.3.1 Optimizing Decision Trees & Random Forest classifiers

For optimizing these two classifiers, different values for each one at the next hyperparameters were tested:

Criterion: this is a measure for calculating the information gain to decide the quality of a split, the possible values for this hyperparameter is "gini" and entropy." The scores were obtained using "gini" for decision trees and "entropy" for the random forest. The formulas for calculating "gini" and "entropy" criteria are illustrated by the figure below, where "p(ci)" is the probability of a class "ci" in a node.

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

Figure 6: Formulas for "gini" and "entropy"

Max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or contain less than min_samples_split (the following parameter) samples. The Values 1-100 were tested, the best score was obtained using a value "10" for decision trees and "20" for

the random forest.
Min_sample_split: is the minimum number of samples required to split an internal node. Values between 1 and 10 were tested. The best scores were for the "4" for decision trees and "2" for the random forest.
$Splitter$: The strategy used to choose the split at each node in Decision trees. For this parameter, the value "best" was used for choosing the best possible split.
$n\_estimators$: The number of trees in the forest for the random forest classifier, the values (100,200,300,400,500) was tested, and the best scores were obtained using the value "200" for this parameter.

5.4 Logistic regression

Logistic regression is a supervised classification algorithm used to predict the probability of a target variable(10). The predictions are based on a set of independent variables. Moreover, logistic regression predicts discrete values, unlike linear regression. There are several types for logistic regression to distinguish; binary multinomial logistic regression. In Binary logistic regression, we use the sigmoid function to compute the probability of belonging to one of two classes. The output of the sigmoid function is a value between 0 and 1(10). In the case of multinomial logistic regression, the independent variable has more than two categories, and the softmax function is used to compute the possible outputs. An advantage of multinomial logistic regression is that it does not assume independence between classes(11).

5.5 Support vector machine

Support vector machines (SVMs) are sparse kernel machine learning techniques primarily used for supervised binary classification. It is one of the most popular machine learning approaches known for robustness, good generalization ability, and unique global optimum solutions(12). SVMs generate a linear separator between the classes that maximize the margin (from the nearest points). But as it is a kernel technique, it maps the data into a higher-dimensional space to solve the task as a convex optimization problem(12). SVMs allow N points to be on the wrong side of the separator; often, the value of N is small. These are called soft SVMs, and when the value of N is 0, they are called hard SVMs. SVMs can also be used for multi-classification problems by using n binary classifiers, but we won't be using that as our classifier is binary.

5.5.1 Optimizing Logistic regression Support vector machine classifiers

C: Inverse to regularization strength, it is seen as the weight of the training data. A higher C-value means a higher attention/weight of the training data. This value must be a positive float. Values 20 values in the log-space between -4 and 4 $logspace(-4, 4, num = 20)$ were tested, the best scores were obtained using "C=206.914" for Logistic Regression and "C=143.450" for the Support Vector Machine classifier.

$Penalty$: In logistic regression, this is also a form of regularization; it is a way to penalize logistic regression models for having too many variables. This minimizes the effect of the variables that have less contribution. For this hyperparameter, the values "L1", "L2", "'elasticity," and "'none'" were tested. The best performance is for using "L2" as a penalty model.

$Solver$: As explained previously, solving Logistic Regression is an optimization problem, the built-in solvers in Scikit-learn are ('lbfgs,' 'newton-cg,' 'liblinear', 'sag', 'saga') and the default solver is $lbfgs$. It's known that sag and $saga$ are the best for larger datasets like the one used in this research. However, all solvers were tested to obtain the best scores. Therefore, the sag solver performed the best and was selected as a value for this hyperparameter.

max_iter: is also a hyperparameter for the Logistic Regression classifier; it represents the maximum number of iterations taken for the solver. The values (100,200,300,400,500,1000,5000) were tested, The best values were obtained with max_iter = "100"

$Kernel$: The kernel hyperparameter for the SVM classifier specifies the kernel type to be used in the algorithm. It helps perform calculations in higher dimensions. The available kernels are ('linear,' 'poly,' 'rbf,' 'sigmoid,' 'precomputed'); the best performance was obtained using a "Linear" kernel.

5.6 Naive Bayes

Naive Bayes methods are supervised learning algorithms based on Bayes' theorem with a naive assumption of independence between features(13). The assumption is called the conditional probability. Naive Bayes assumes that the input belongs to the class with the higher probability, which could be determined using the law above. Important advantages of Naive Bayes are that it could be used for large and small data, and it is considered a fast algorithm.(13)(14).

5.6.1 Optimizing Naive Bayes

$Var\_smoothing$: Zero probabilities in Naive Bayes can cause calculation problems. To solve this, smoothing replaces the zero probabilities with a float close to zero. var_smoothing is the portion of the most significant variance of all features that are added to variances for calculation stability. By optimizing this hyperparameter, we can find a value for smoothing that provides the best prediction scores. One hundred sample-values in the log-space between 0 and 9 were tested logspace(0,-9, num=100), best was results were obtained using the value "3.511e-05".

6 Result

6.1 Hyperparameter optimization results

The optimization was done on the pre-processed data before balancing the target class (income); this decision has two reasons. The first is to observe the effect of optimization on the accuracy of the prediction. Second, to use fixed hyperparameters values that deliver the optimal performance of each classifier as a baseline for comparing the performance of the classifiers after using different balancing methods. Note that all results shown in the roc plots and the score tables are the average of repeating the prediction ten times for each classifier. As illustrated by the AUC plots below, the default hyperparameters' best-performing classifiers were Naive Bayes, Random Forest, Logistic Regression, Decision Trees, KNN, and SVC respectively. After optimization, the performance improved slightly for all classifiers. The best performing classifiers after the optimization were Decision Trees, Naive Bayes, Random Forest, Logistic Regression, SVC, and KNN respectively.

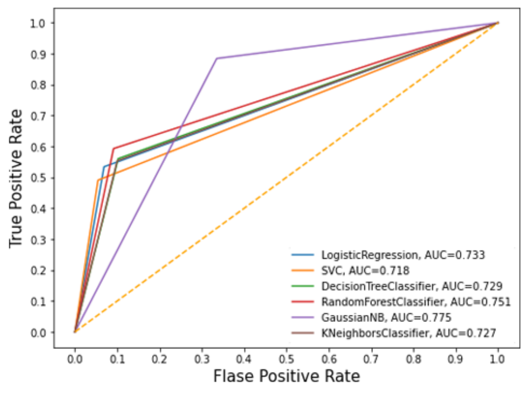


Figure 7: roc-AUC plot (default hyperparameters)

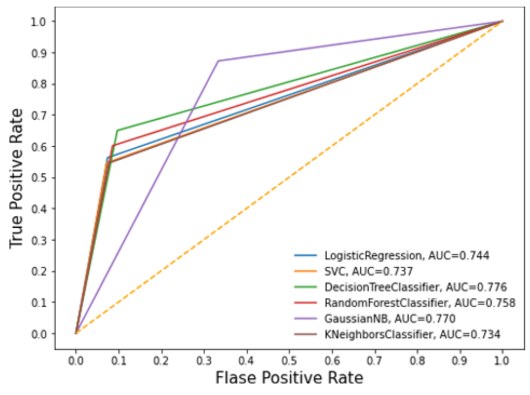


Figure 8: roc-AUC plot (optimized hyperparameters)

6.2 Sampling Results

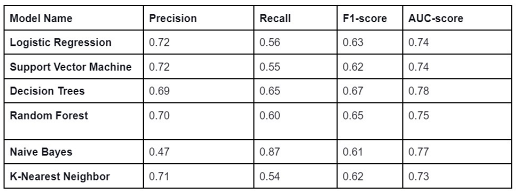
As mentioned previously, the optimized models are used as a baseline for comparing the different sampling methods. The values of Precision, Recall, F1-score, and AUC-score have been computed for all classifiers using Random oversampling, SMOTE oversampling, and without using any oversampling method. The process of splitting the training data, training, and prediction was repeated ten times, and the averages of the scores are displayed in figure X and figure X1. An example of the 10-runs scores is given in the appendix.

6.3 Random oversampling (with replacement)

The models' computations were performed on balanced data using the random oversampling method. Despite the data class imbalance in the data, oversampling applies changes the class distribution where it transforms the minority class by duplicating the instances and uplifting the amount of data in the minority class. The duplicates allow the classifier to recognize more features from the minority class, increasing the prediction chances. Compared with unbalanced data, the precision values slightly decrease in all the models, whereas a notable increase occurs in the recall values. This improves the F1-score regarding most models except for DT and NB, which retain identical scores. Regarding AUC-scores, all models vary between 1% for NB and 8% for Logistic Regression.

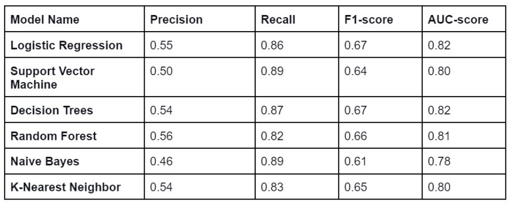
6.4 SMOTE

SMOTE is an enhanced version of random oversampling. Instead of duplicating the data of the minority class, this approach generates new realistic corrupted data and simultaneously keeps the values within the same region of the minority class. Compared to the outcomes of the models pre and post SMOTE, all the models show significant improvements overall in the AUC rate(around 10% on average), except for Naive Bayes, which surprisingly has a slight increase in the rate after performing the SMOTE on the data. The graph below shows detailed records of the AUC rates for all models.



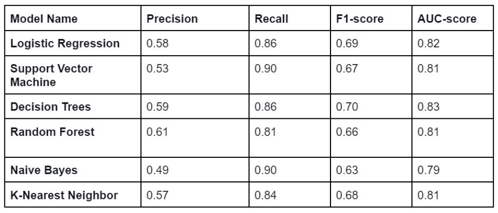
Model Name	Precision	Recall	F1-score	AUC-score
Logistic Regression	0.72	0.56	0.63	0.74
Support Vector Machine	0.72	0.55	0.62	0.74
Decision Trees	0.69	0.65	0.67	0.78
Random Forest	0.70	0.60	0.65	0.75
Naive Bayes	0.47	0.87	0.61	0.77
K-Nearest Neighbor	0.71	0.54	0.62	0.73

Imbalanced data (optimized models)



Model Name	Precision	Recall	F1-score	AUC-score
Logistic Regression	0.55	0.86	0.67	0.82
Support Vector Machine	0.50	0.89	0.64	0.80
Decision Trees	0.54	0.87	0.67	0.82
Random Forest	0.56	0.82	0.66	0.81
Naive Bayes	0.46	0.89	0.61	0.78
K-Nearest Neighbor	0.54	0.83	0.65	0.80

Balanced using Random Oversampling



Model Name	Precision	Recall	F1-score	AUC-score
Logistic Regression	0.58	0.86	0.69	0.82
Support Vector Machine	0.53	0.90	0.67	0.81
Decision Trees	0.59	0.86	0.70	0.83
Random Forest	0.61	0.81	0.66	0.81
Naive Bayes	0.49	0.90	0.63	0.79
K-Nearest Neighbor	0.57	0.84	0.68	0.81

Balanced using SMOTE

Figure 9: Comparison of model performance & sampling method

7 Conclusion

Six machine learning classifiers and two oversampling methods were tested to predict income using a census dataset, and compare the performance of the different minority oversampling methods. There were repeated runs for all the models, where we used a split ratio of 80% training data and the remaining 20% test data. While the test data was being withheld to compare the models in terms of performance, we attempted to optimize the models using grid search for the best roc-AUC score. For the hyperparameter tuning, we used cross-validation on our training data with cv = 5. For predictions done on the over-sampled data using SMOTE, The overall recall for all models stayed almost the same, with minor fluctuations of +/- 1% in comparison with predictions done with Random Oversampling. Furthermore, the results showed an improvement of 1% on the AUC score for all the models except the random forest and logistic regression, and both stayed nearly the same. As a final overview, we plotted the confusion matrices for the duplication and the SMOTE to compare the overall change in the performance. And meeting our expectations, we evidenced improvement in performance while using SMOTE for all the models, resulting from a decrease in the number of false negatives that correspond to misclassifications in the majority class, and an increase in the true positive rate [confusion matrices for all the models can be found in the appendix].

Due to the scarcity of time, we could not check if the results would generalize to other imbalanced datasets. Regarding future research prospects, studies can be done to check if SMOTE will still outperform Random oversampling on different datasets. Furthermore, other oversampling methods could be compared to SMOTE, like ADASYN or K-Mean SMOTE, to have more emphasis on the conclusion.

References

[1] R. Kochhar, *A global middle class is more promise than reality*. Routledge India, 2020.

[2] U. M. Learning, "Adult census income." `https://www.kaggle.com/uciml/adult-census-income`, 2017.

[3] A. Lazar, "Income prediction via support vector ma-

chine..” https://doi.org/10.1109/icmla.2004.1383506, 2004.

[4] J. R. de Castro Vieira, F. Barboza, V. A. Sobreiro, and H. Kimura, “Machine learning models for credit analysis improvements: Predicting low-income families’ default.” http://dx.doi.org/10.1016/j.asoc.2019.105640, 2019.

[5] C. F. Caiafa, Z. Sun, T. Tanaka, P. Marti-Puig, and J. Solé-Casals, “Machine learning methods with noisy, incomplete or small datasets.” https://doi.org/10.3390/app110941, 2021.

[6] N. Chakrabarty and S. Biswas, “A statistical approach to adult census income level prediction",” in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 207–212, 10.1109/ICACCCN.2018.8748528, 2018.

[7] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, “Knn model-based approach in classification.” https://doi.org/10.1007/978-3-540-39964-3_62, 2003.

[8] O. Maimon and L. Rokach, “Data mining and knowledge discovery handbook.” https://doi.org/10.1007/0-387-25465-X_9, 2005.

[9] M. Pal, “Random forest classifier for remote sensing classification.” https://doi.org/10.1080/01431160412331269698, 2005.

[10] T. Point, “Machine learning - logistic regression.” https://bit.ly/36KXjPk.

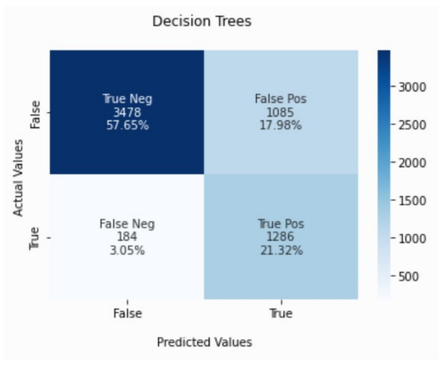
[11] B. Krishnapuram, L. Carin, M. A. Figueiredo, and A. J. Hartemink, “Sparse multinomial logistic regression: Fast algorithms and generalization bounds,” vol. 27, pp. 957–968, 10.1109/TPAMI.2005.127, IEEE, 2005.

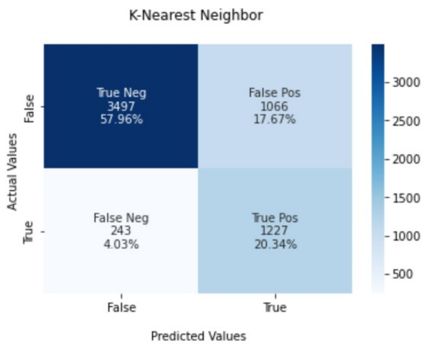
[12] M. Awad and R. Khanna, “Support vector machines for classification.” https://doi.org/10.1007/978-1-4302-5990-9_3, 2015.

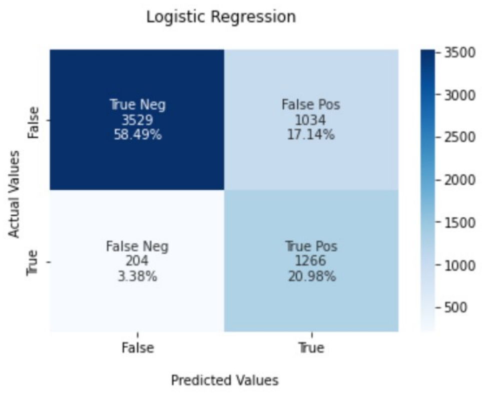
[13] S. Learn, “1.9. naive bayes.” https://scikit-learn.org/stable/modules/naive_bayes.html.

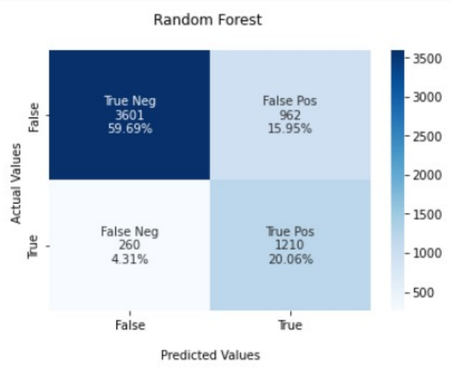
[14] A. Navlani, “Knuth: Computers and typesetting.” https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn.

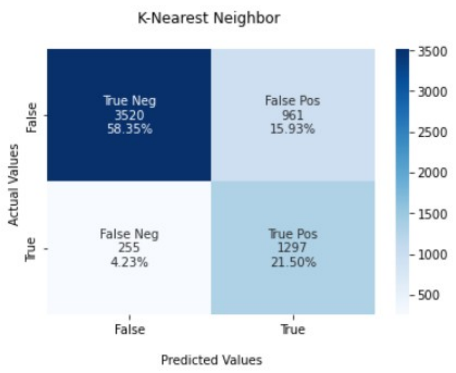
A Appendix

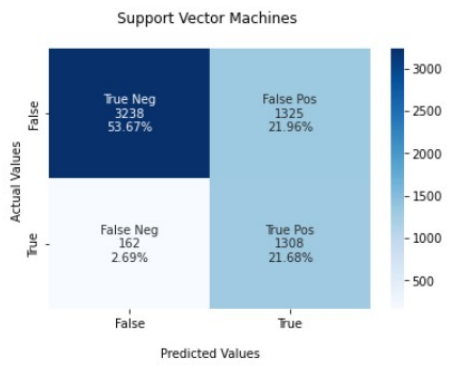


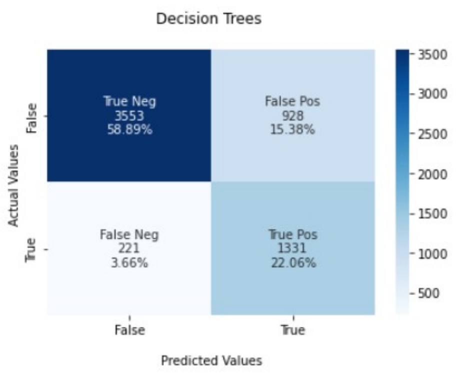


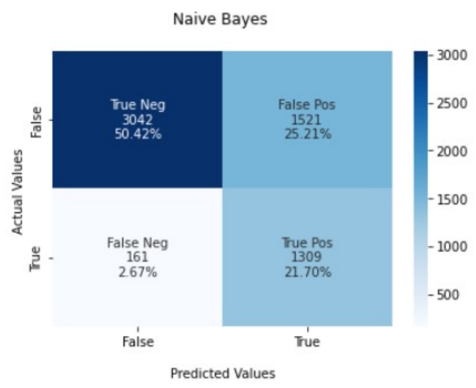


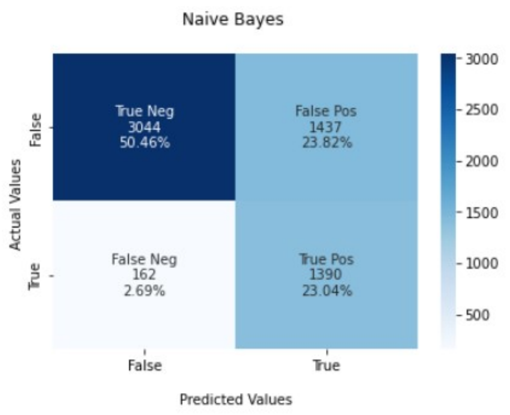




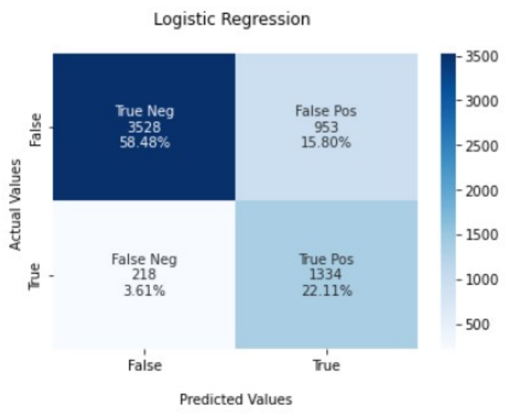


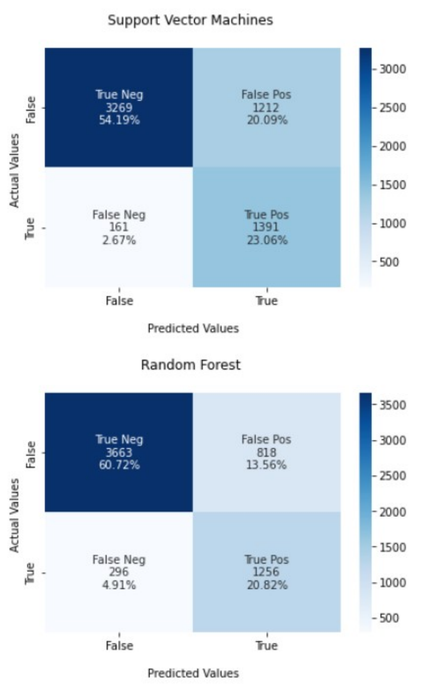






Appendix 1: Confusion matrices (Random Oversampling)





Appendix 2: Confusion matrices (SMOTE))