Prediction of Flight Cancellation

Project Report
submitted in the fulfilment of the requirements for the course

CSCI-B 565 DATA MINING

BY

RATAN TEJASWI VADAPALLI
DURGA SINDHU ANIMALLA

Under the guidance of

Dr. YUZHEN YE

Professor of Informatics and Computer Science

Luddy School of Informatics, Computing and Engineering

Indiana University, Bloomington.

Abstract

It is quite annoying to find out in the eleventh hour that a flight one is about to catch has been cancelled. Flight cancellations have proven to be a passenger's nightmare and pretty much spoils anyone's plans. Cancellations are costly to the airline as much as they are to the passengers. But no significant progress has been made to accurately predict if a flight will be cancelled or not since there are many factors which affect flight cancellations. This project is an attempt at predicting flight cancellations based on some of those factors using various classification models. A successful prediction of the cancellations of flights can greatly help passengers save a lot of time, effort and money and also be helpful to the airlines in planning a better resource allocation and prevent wastage of valuable resources due to unforeseen circumstances.

Keywords

Flight Cancellation, Classification, Decision Tree, Bagging Classifier, AdaBoost Classifier, SMOTE, Random Under Sampler, ADASYN

TABLE OF CONTENTS

Introduction	4
Methods	4
Results	10
Discussion	16
References	17

1.Introduction

Air transport has been a boon to mankind as it significantly reduced travel time compared to other modes of transport and increased convenience. However, even flights have plaguing issues like flight cancellations and flight delays with flight cancellations being the worse of both since they incur losses to both the passenger and the airline. Flights might be cancelled due to plethora of reasons like bad weather, mechanical issues, airports being overcrowded by flights sometimes etc. Since the factors causing the delays are so unpredictable and diverse, it is challenging to precisely predict flight cancellations as we must take many factors into play. But, if we can predict cancellations with the help of Machine Learning algorithms, passengers can make changes to their plans beforehand and it would help reduce wastage of resources and time. The data used in this project is taken from the Flight Status Prediction 2021 dataset of Kaggle website, which has data of flights in the year 2021. The data obtained from the dataset had a very skewed distribution of the target. The records with cancelled flights were only 1.75% of the total number of records. This introduced various challenges in the model like overfitting. Hence, multiple approaches were tried in this project which will be explained below.

2.Methods

There are various methods and techniques used in this project. The data obtained from the dataset was not fully usable initially so, there was some Data Preprocessing done on the data.
This data was then split into train and test data with test size 0.5 which means that half of the data was used for training the model and the other half for testing. Test size was chosen as the test size. Then, classification techniques like Decision Tree classifier, Bagging classifier and Adaboost Classifier were used.

2.a) Dataset

The dataset used was the Flight Status Prediction 2021 dataset from Kaggle. This dataset has information about all flights like cancellations and delays In the USA during the year 2021. The dataset has 6.3 million records and 61 attributes.

- 15 attributes out of 61 were selected for the prediction and one other target attribute.
- Out of these 3 were categorical data, and the remaining were numerical data.
- The target attribute has a skewed distribution with 0.11 million True values and 6.2 million False values which is shown in Fig 2.a.2
- The dataset had null values.

```
In [220]: print(flight_data.shape)

(6311871, 61)
```

Fig 2.a.1 Shape of the data

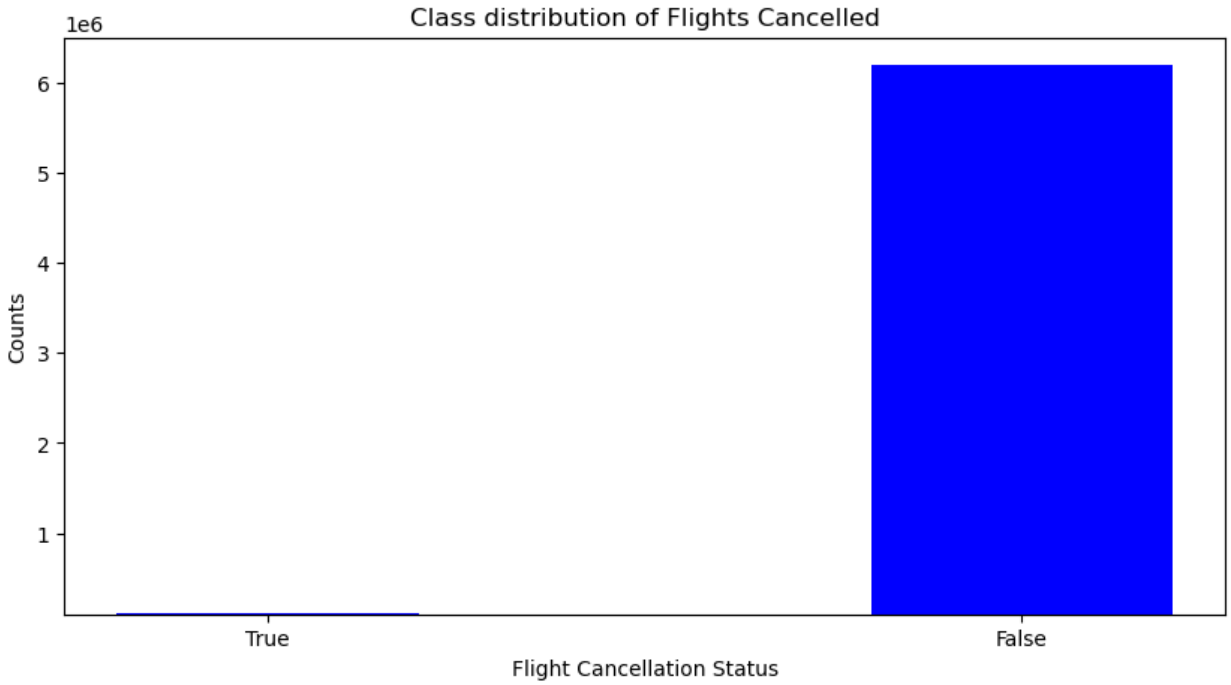

Fig 2.a.2 Class Distribution of the Target attribute

2.b) Data Preprocessing

The data preprocessing methods used are handling missing data, integer encoding, feature selection and sampling. The following was done:

1. There was missing data found in multiple columns but the decision to handle it was only taken after checking if the features which are useful for the project have missing data. Since required attributes had missing data, they were filled with the mode of the column. This was done using the fillna() method as shown in Fig 2.b.1

```
In [258]: flight_data_mode = flight_data.fillna(flight_data.mode(axis=0, numeric_only=False, dropna=True).iloc[0])

In [260]: flight_data_mode.isnull().any().any()

Out[260]: False

In [261]: flight_data_mode.isnull().any()

Out[261]: FlightDate            False
          Airline               False
          Origin                False
          Dest                  False
          Cancelled             False
                                ...
          ArrDel15              False
          ArrivalDelayGroups    False
          ArrTimeBlk            False
          DistanceGroup         False
          DivAirportLandings    False
          Length: 61, dtype: bool
```

Fig 2.b.1 Figure showing null values filled with mode of the column

2. Some required attributes like Airline, Origin and Destination were of the type object in the dataframe and hence were not understandable by the model. Hence, they were converted into integers where each unique airline or an airport will get an integer value. No built-in method was used for this but replacement of values was done normally.
3. On analyzing the data, it is found out that existence of some attributes like Departure Delay were automatically causing an easy decision for the models and even though the model predicts that the flight is not cancelled for all cancelled flights, it would not affect the accuracy as the number of records with cancelled flights is very less. Hence, only some features were selected to be used in the data as shown in Fig 2.b.2

```
In [481]: req_cols = ['Airline','Origin','Dest','DistanceGroup','DestStateFips','DestAirportSeqID','DestAirportID',
                      'DestCityMarketID','OriginWac','DayOfWeek','DayofMonth','Month','Quarter','Distance','CRSDepTime']
```

Fig 2.b.2 Figure showing the selected features

4. Three types of sampling techniques were used. Firstly, records with flight cancelled which is the minority class was oversampled using the SMOTE technique as shown in Fig 2.b.3. The second technique was where the minority class samples were oversampled with SMOTE and sampling strategy 0.1 and the majority class was undersampled using RandomUnderSampler with sampling strategy 0.5 as shown in Fig 2.b.4. The third technique was where the minority class samples were oversampled using ADASYN as shown In Fig 2.b.5.

```
In [379]: from imblearn.over_sampling import SMOTE
          oversample = SMOTE()
          X, Y = oversample.fit_resample(X, Y)

In [380]: from collections import Counter
          count = Counter(Y)
          print(count)

          Counter({False: 6200853, True: 6200853})
```

Fig 2.b.3 Figure showing SMOTE oversampling and class counts

```
In [398]: from imblearn.under_sampling import RandomUnderSampler
          oversample = SMOTE(sampling_strategy=0.1)
          undersample = RandomUnderSampler(sampling_strategy=0.5)

In [399]: from imblearn.pipeline import Pipeline
          steps = [('o', oversample), ('u', undersample)]
          pipeline1 = Pipeline(steps=steps)

In [401]: X, Y = pipeline1.fit_resample(X, Y)

In [402]: from collections import Counter
          count = Counter(Y)
          print(count)

          Counter({False: 1240170, True: 620085})
```

Fig 2.b.4 Figure showing SMOTE oversampling and under sampling using RandomUnderSampler

```
In [423]: from imblearn.over_sampling import ADASYN

In [424]: oversample = ADASYN()
          X = flight_data_mode[req_cols]
          Y = flight_data_mode.Cancelled
          X, Y = oversample.fit_resample(X, Y)

In [427]: counter = Counter(Y)
          print(counter)

          Counter({False: 6200853, True: 6196105})
```

Fig 2.b.5 Figure showing oversampling using ADASYN

SMOTE stands for Synthetic Minority Oversampling Technique which is an approach for oversampling of imbalanced datasets for classification. This does the duplication of samples from the minority class from the train set before it is fit into the model. This helps in balancing the distribution of the class but does not provide any additional information for the prediction.

Random Under Sampler does under sampling by random picking of the samples from the majority class and deletes them from the training set and we can specify it to do it with or without replacement. This is done to achieve a balanced distribution.

ADASYN stands for Adaptive Synthetic Sampling Approach for Imbalanced Learning which is a synthetic sampling approach which generates more samples from where the minority class density is less and less samples from where the density is high in the minority class. This is a modified version of the SMOTE technique for achieving better balance in the distribution.

2.c) Classification

Initially, the data was trained and tested on the three classification techniques used in this project before sampling and the models were observed to be overfitting and was seen to predict only the majority class for the whole test set but still achieved a very high accuracy due to the imbalanced distribution. Then, even on trying different things like only using some records from the dataset didn't do much good and the models were only predicting the majority class. But, after applying sampling techniques to the data, there were some changes in the results. The three classification techniques used are the Decision Tree classifier, Bagging classifier and the Adaboost classifier. The observations and results of each classifier will be discussed below.

1. Decision Tree Classifier:
   It is a supervised machine learning algorithm which makes decisions based on a set of rules. In this project, four different data were fit into the model and four different predictions were made. The max depth of the tree was kept being 4. This was done because thought the training error reduces if max depth is increased, the accuracy of the test data will take a hit. We have taken 'entropy' as the criterion as the results obtained using entropy criterion are better compared to gini criterion though gini criterion is faster in reaching a decision. The implementation is as shown in Fig 2.c.1.

```
In [471]: dec_tree_clsfr = DecisionTreeClassifier(criterion="entropy",max_depth = 4)
          dec_tree_clsfr = dec_tree_clsfr.fit(X_train,y_train)
          y_pred_dec = dec_tree_clsfr.predict(X_test)
```

Fig 2.c.1 Figure Showing Decision Tree classifier implementation

2. Bagging Classifier:
   Bootstrap aggregation, also known as bagging, is an ensemble learning method which helps in increasing accuracy and the performance of a machine learning algorithm. It is also known to reduce variance in a noisy dataset. The base estimator used in the bagging classifier was Decision Tree. The sampled were taken with replacement which is the default for the bagging classifier. Even the features were taken with replacement. Default values were taken for all the parameters of the bagging classifier as shown in Fig 2.c.2.

```
In [480]: bag_clsfr = BaggingClassifier()
          bag_clsfr.fit(X_train, y_train)
          y_pred = bag_clsfr.predict(X_test)
```

Fig 2.c.2 Figure Showing Bagging Classifier Implementation

3. Adaboost Classifier:
   Adaptive Boosting, also known as Adaboost, is a meta-estimator which is a supervised learning method. It is used to increase the performance of the machine learning algorithm. Decision Tree was used as the base estimator in the Adaboost Classifier. All the parameters of the Adaboost Classifier were set to the default values. Adaboost is strong against overfitting and hence we used it in this project as shown in Fig 2.c.3.

```
In [359]: ada_clsfr2 = AdaBoostClassifier()
          ada_clsfr2.fit(X_train, y_train)
          y_pred_ada1 = ada_clsfr2.predict(X_test)
```

Fig 2.c.3 Figure Showing Adaboost Classifier Implementation

3.Results

The following were the observations for each of the classification model with the different sampling techniques.

3.a) Decision Tree Classifier

The following are the confusion matrices for the 4 cases of data with Decision Tree classifier:

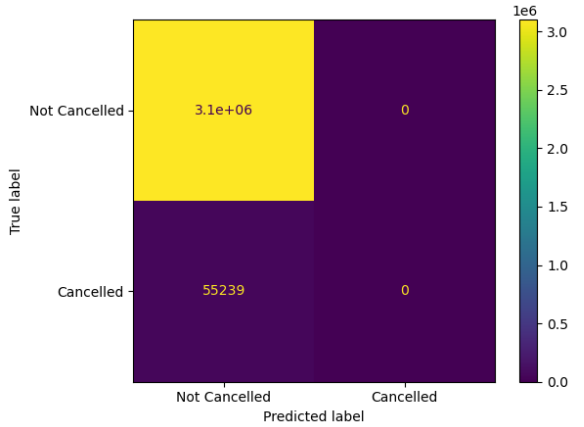

Fig 3.a.1 Data with no sampling

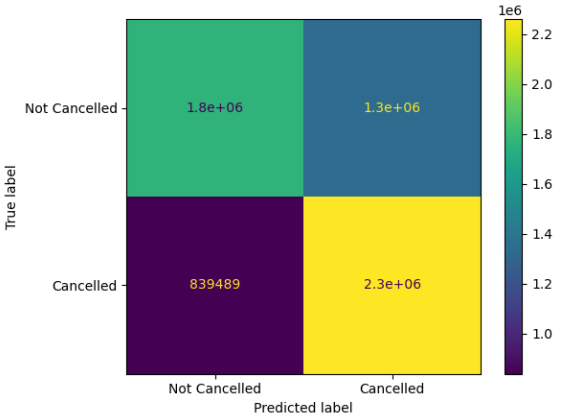

Fig 3.a.2 Data oversampled with SMOTE

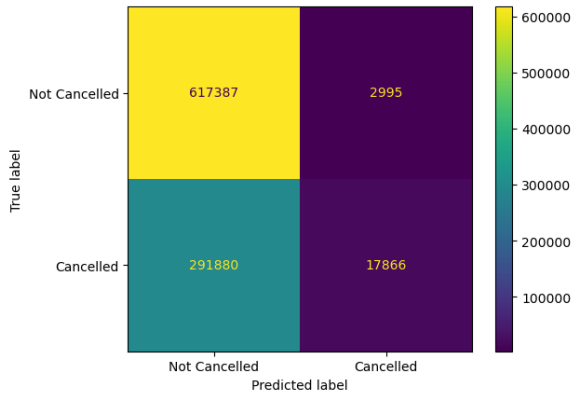

Fig 3.a.3 Data oversampled with SMOTE and Undersampled with RandomUnderSampler

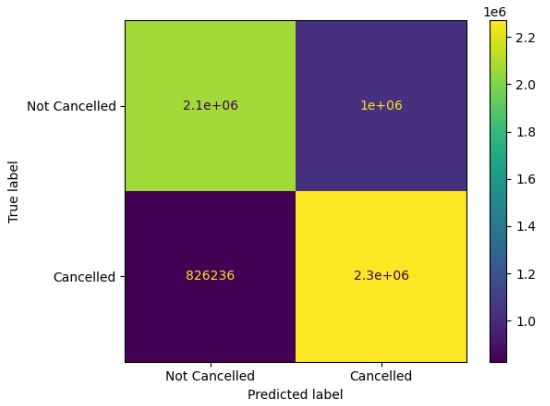

Fig 3.a.4 Data oversampled with ADASYN

Figure 3.a.1 shows that the false positives and true negatives are 0 indicating that the model is predicting one class for the whole test set or train set might have only one class. The model after fitting with data after sampling show that the false positives and true negatives are 0, indicating that the model is predicting both true and false for the target attribute. The precision, recall, f1 score and the accuracy results for all 4 sets of data with the decision tree model are as below:

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_dec, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.000
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_dec,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.000
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_dec, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.000
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_dec))
```

```
Accuracy: 0.9824967933443517
```

Fig 3.a.5 Results with no sampling

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_dec1, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.675
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_dec1,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.628
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_dec1, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.729
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_dec1))
```

```
Accuracy: 0.6487605818102767
```

Fig 3.a.6 Results of SMOTE oversampling

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_dec2, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.108
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_dec2,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.856
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_dec2, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.058
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_dec2))
```

```
Accuracy: 0.6829737412485163
```

Fig 3.a.7 Results with Over Sampling And Under Sampling

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_dec3))
```

```
Accuracy: 0.6439721744640903
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_dec3,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.609
```

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_ada3, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.710
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_ada3, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.733
```

Fig 3.a.8 Results with ADASYN oversampling

The results show that with the decision tree classifier model, the best balance of accuracy, precision, f1 score and recall with the ADASYN oversampling. Even oversampling with SMOTE showed very similar results. Though the accuracy was higher with oversampling with smote and undersampling with RandomUnderSampler, it had bad precision, f1 score and recall.

3.b) Bagging Classifier

The following are the confusion matrices for the 4 cases of data with Bagging classifier:

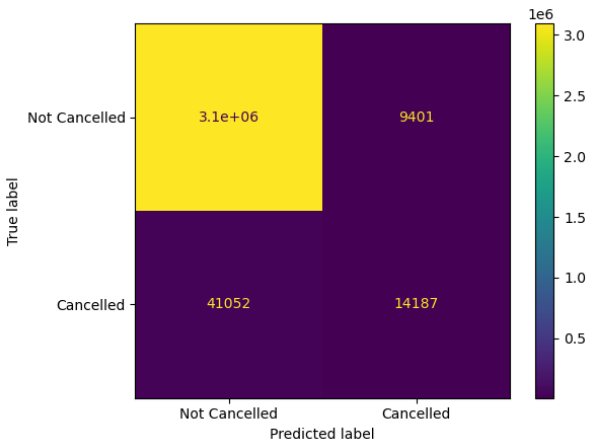

Fig 3.b.1 Data with no sampling

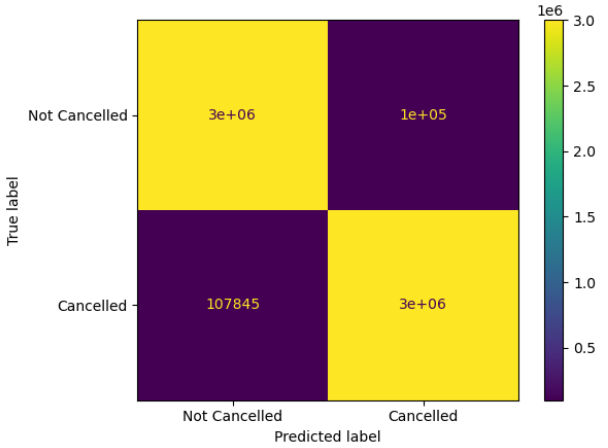

Fig 3.b.2 Data oversampled with SMOTE

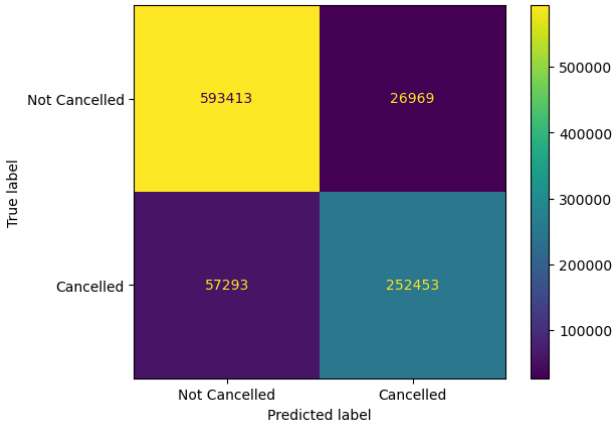

Fig 3.b.3 Data oversampled with SMOTE and Undersampled with RandomUnderSampler

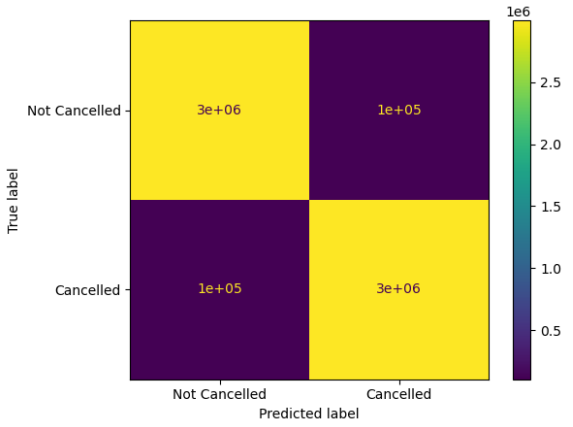

Fig 3.b.4 Data oversampled with ADASYN

We observe from figures 3.b.1, 3.b.2, 3.b.3, 3.b.4 that false positives and true negatives are not 0 in any case unlike in a decision tree with data without sampling. The precision, recall, f1 score and the accuracy results for all 4 sets of data with the bagging classifier model are as below:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```
```
Accuracy: 0.9840133006499499
```
```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred,average = 'binary')
print('Precision: %.3f' % prec)
```
```
Precision: 0.601
```
```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred, average='binary')
print('Recall: %.3f' % recall)
```
```
Recall: 0.257
```
```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred, average='binary')
print('F-Measure: %.3f' % score)
```
```
F-Measure: 0.360
```

Fig 3.b.5 Results with no sampling

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_bag1))
```
```
Accuracy: 0.9664060734869864
```
```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_bag1,average = 'binary')
print('Precision: %.3f' % prec)
```
```
Precision: 0.968
```
```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_bag1, average='binary')
print('Recall: %.3f' % recall)
```
```
Recall: 0.965
```
```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_bag1, average='binary')
print('F-Measure: %.3f' % score)
```
```
F-Measure: 0.966
```

Fig 3.b.6 Results of SMOTE oversampling

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_bag2,average = 'binary')
print('Precision: %.3f' % prec)
```
```
Precision: 0.903
```
```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_bag2, average='binary')
print('Recall: %.3f' % recall)
```
```
Recall: 0.815
```
```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_bag2, average='binary')
print('F-Measure: %.3f' % score)
```
```
F-Measure: 0.857
```
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_bag2))
```
```
Accuracy: 0.9094081674780246
```

Fig 3.b.7 Results with Over Sampling And Under Sampling

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred,average = 'binary')
print('Precision: %.3f' % prec)
```
```
Precision: 0.966
```
```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred, average='binary')
print('Recall: %.3f' % recall)
```
```
Recall: 0.967
```
```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred, average='binary')
print('F-Measure: %.3f' % score)
```
```
F-Measure: 0.967
```
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```
```
Accuracy: 0.9665651847816215
```

Fig 3.b.8 Results with ADASYN oversampling

The results of the bagging classifier show that on the data with no sampling, even bagging classifier struggled on the precision and recall scores while having a very high accuracy as seen in Fig 3.b.5. Here, we can infer that it had issues like Decision Tree's but performed better than the decision tree. But on sampling the data we can see that the precision, recall and f1 scores have seen a significant bump up while maintaining a better accuracy than the decision tree. Data oversampled with SMOTE, and ADASYN fit into the models performed very similarly and they showed very good results. Even data oversampled with SMOTE and undersampled with RandomUnderSampler was much better than the original but not as good as SMOTE and ADASYN as we observe from figures 3.b.6, 3.b.7 and 3.b.8.

3.c) Adaboost Classifier

The following are the confusion matrices for the 4 cases of data with Adaboost classifier:

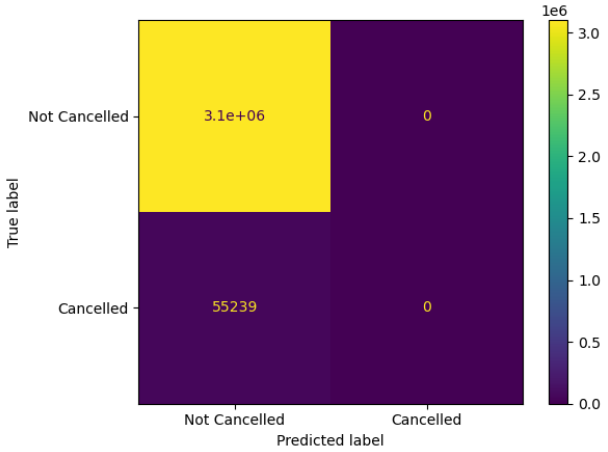

Fig 3.c.1 Data with no sampling

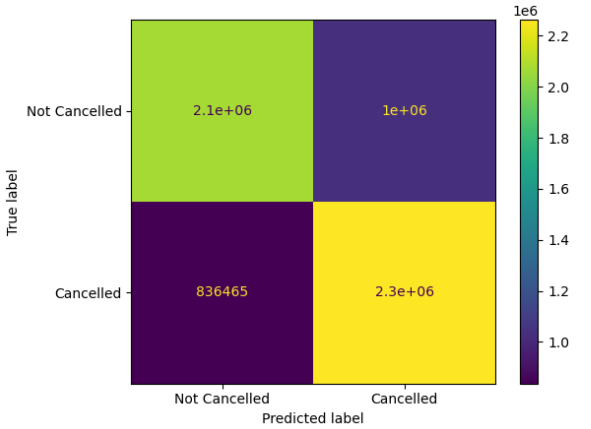

Fig 3.c.2 Data oversampled with SMOTE

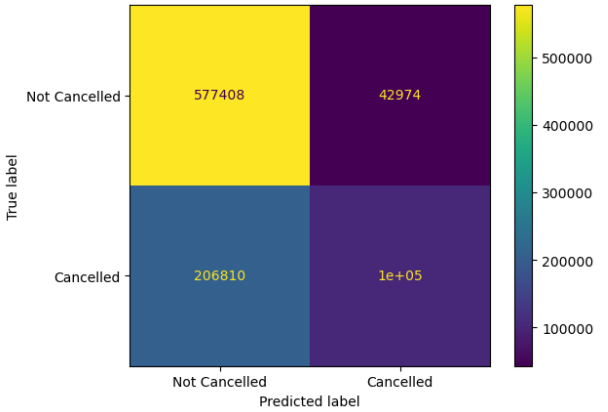

Fig 3.c.3 Data oversampled with SMOTE and Undersampled with RandomUnderSampler

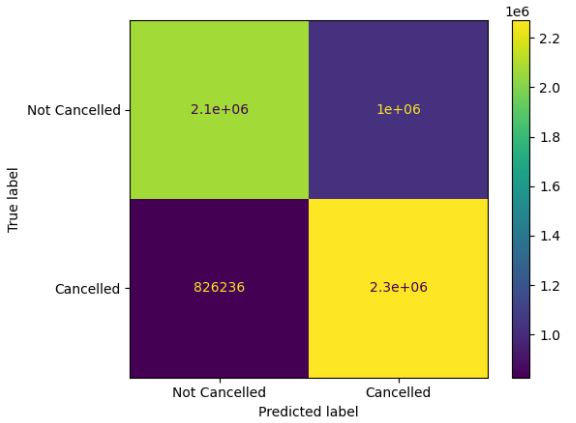

Fig 3.c.4 Data oversampled with ADASYN

From figure 3.c.1, we observed that the model fit with the original data showed the same issue as the decision tree classifier where the false positives and true negatives are zero indicating that the model might have predicted only one class and the minority class might not have been there in the train set. This was mitigated in the data with sampling as we see in figures 3.c.2, 3.c.3, 3.c.4. The right inferences on this can be made from the results below.

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_ada1,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.000
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_ada1, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.000
```

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_ada1, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.000
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_ada1))
```

```
Accuracy: 0.9824967933443517
```

Fig 3.c.5 Results with no sampling

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_ada1, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.708
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_ada1,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.686
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_ada1, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.730
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_ada1))
```

```
Accuracy: 0.6984214913657847
```

Fig 3.c.6 Results of SMOTE oversampling

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_ada2, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.452
```

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_ada2,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.705
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_ada2, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.332
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_ada2))
```

```
Accuracy: 0.7314520152065093
```

Fig 3.c.7 Results with Over Sampling And Under Sampling

```
from sklearn.metrics import precision_score
prec = precision_score(y_test,y_pred_ada3,average = 'binary')
print('Precision: %.3f' % prec)
```

```
Precision: 0.687
```

```
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred_ada3, average='binary')
print('F-Measure: %.3f' % score)
```

```
F-Measure: 0.710
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred_ada3, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 0.733
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_ada3))
```

```
Accuracy: 0.7000746473449374
```

Fig 3.c.8 Results with ADASYN oversampling

The results in the figure 3.c.5 show that the accuracy is 98.2% whereas the precision, recall and f1 score are 0. This is similar to what is observed with the Decision Tree Classifier where only one class of the target variable is being predicted. But the result after sampling the data is better as the precision, recall and f1 score are not 0. We observe from figures 3.c.6 and 3.c.8 that SMOTE and ADASYN oversampling did a better job that doing SMOTE oversampling and undersampling using RandomUnderSampler as we see from figure 3.c.7 as though accuracy is lower, SMOTE and ADASYN had a significantly better recall, precision and f1 scores. So, it is safe to assume that SMOTE and ADASYN oversampled data fit into the model produced better predictions.

4.Discussion

From all the observations above, it is seen that Bagging classifier was the best model for the current dataset which had a very huge number of records and where the target variable had a very skewed distribution. Even before sampling the data, it was better than the other two classifiers, and on sampling, showed a very good improvement over the other models. The sampling techniques which stood out were the ADASYN and SMOTE techniques. On applying oversampling using ADASYN to the minority class, we saw a significant improvement in the precision, recalls and f1 scores of all the three models and that data with Bagging Classifier showed the best overall result. On oversampling the minority class with SMOTE and fitting into the models showed very close results to the ADASYN model and SMOTE applied to data fit in the Bagging Classifier was the next best combination.

This shows that having very few records of the minority class was causing the train set to have a very few records or even no records of the minority class so that the model is on a whole ignoring the minority class and predicting only the majority class for the whole test set and even though it is predicting a wrong label for the minority class in the test set, the accuracy numbers are very high due to the number of records in the majority class being relatively so huge. Hence, oversampling of the majority class has proven to be a very good way of tackling the class imbalance problem. We have seen that undersampling the majority class was producing mediocre results as some of the necessary information might be lost while some records of the majority class are dropped from being considered.

5.References

1. Yu Yanying, Hai Mo and Li Haifeng, “A Classification Prediction Analysis of Flight Cancellation Based on Spark”, 7th International Conference on Information Technology and Quantitative Management (ITQM 2019)
2. Jia Yi, Honghai Zhang, Hao Liu, Gang Zhong and Guiyi Li, “Flight Delay Classification Prediction Based on Stacking Algorithm”, Journal of Advanced Transportation, vol.2021, Article ID 4292778, 10 pages, 2021. https://doi.org/10.1155/2021/4292778
3. N.V. Chawla, K.W Bowyer, L.O. Hall and W.P Kegelmeyer, “SMOTE: Synthethic Minority Over-sampling Technique”, Journal of Artificial Intelligence Research, Volume 16, pages 321-357, 2002. https://doi.org/10.48550/arXiv.1106.1813.
4. Z.Shu, “Analysis of Flight Delay and Cancellation Prediction Based on Machine Learning Models”, 2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), 2021, pp. 260-267, doi: 10.1109/MLBDBI54094.2021.00056.
5. Ansari, Ahlam and Shaikh, Ashad and Mapkar, Salim and Khan, Maaz, Cancellation Prediction for Flight Data Using Machine Learning (April 8, 2019). 2nd International Conference on Advances in Science & Technology (ICAST) 2019 on 8th, 9th April 2019 by K J Somaiya Institute of Engineering & Information Technology, Mumbai, India, Available at SSRN: https://ssrn.com/abstract=3367683 or http://dx.doi.org/10.2139/ssrn.3367683
6. Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.
7. Decision Tree Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
8. Bagging Classiifer: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#:~:text=A%20Bagging%20classifier.,to%20form%20a%20final%20prediction.
9. Adaboost Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
10. SMOTE: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
11. RandomUnderSampler: https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html
12. ADASYN: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html