Name: Adel Hamad Alkhamisy

Miner2 username: AA
Accuracy=0.76 and rank=59 on miner

The approach that I used is as follow:

Step 1: preprocessing part

First, I analyzed the training dataset which consist of a sparse binary matrix. Each instance(row) start with 0 (inactive) or 1 (active) and this represents the class label. Class label and features are separated by tab. Only the number of non-zero features (columns with 1 binary value) are provided, and the number of zero features are omitted. For example, if we found number 100000 in instance 1, this means that feature (column) 100000 has binary value 1. However, each instance has few numbers, and there are 100000 features in training dataset. Also, the data are imbalanced only 78 active and the rest which are 722 inactive. **Second**, in order to process the data correctly. I imported the train and test data into two dataframes train_data and test_data using read_table() method from pandas library. **Third**, I stored the class label in Y numpy array for easy processing using np.array(train_data[0]) because the class label in train_data dataframe column 0. **Fourth**, I stored the feature list in x which is pandas Series using train_data[1]. Now, each element in the X Series is a string holding the values of features for a specific instance. **Fifth**, I did the same for test data except that the test data does not have a class label. **Then**, I checked for Null values to make sure there are no null values in the data.

After that, I developed a function called transform_to_sparse() to expand the data set to its full features list including features that have 0 values using coo_matrix((entries, (r, c)) method. This function receives a Series X or test that developed earlier and creates three lists entries, row, and col and iterate over their instances. The index of instance is stored in i. In each instance, it splits the string that contains the features list by space and convert each number data type from string to integer using map() method and store them in **features**. **Then**, the function iterates over the **features** and appends 1 to entries list, appends i (index of instance) to row list, and feature – 1 to col list. With the help of coo_matrix((entries, (r, c)) method that create a sparse matrix in coordinate format using three arrays: entries that represents entries of the matrix, row that represents row indices of matrix entries and col that represents the column indices of the matrix entries. Finally, I created x_sparse and test_sparse by calling transform_to_sparse() and pass x and test as arguments. Now, x_sparse is a sparse matrix of 800x100000 and test_sparse is 350x100000 sparse matrix.

Step 2: balance the imbalanced data:

I tried three methods from imblearn library which offers many features for dealing with imbalanced data.

1. Random Undersampling:

   In this strategy the majority class is reduced randomly with or without replacement. I found that sampling_strategy=1 without replacement which means the majority class will be equal to minority class after resampling is the best in this situation.

2. Random Oversampling:

In this strategy the minority class is increased randomly by picking samples with replacement. However, this methods of sampling made my classifiers prone to overfitting. See result tables.

3. Oversampling SMOTE:

   In this strategy the minority class is increased by creating Synthetic Minority Over-sampling. To create synthetic samples, the number of nearest neighbors will be employed. However, this produced poor results. See result tables.

Step 3: feature reduction technique:

I used singular value decomposition which decompose sparse matrix into tree matrices M=UΣV* where A is n*m matrix and U is n*r matrix and Σ is r*r diagonal matrix and V* is r*m matrix. The diagonal matrix represents feature importance in an ordered manner and the most important one is on the top. SVD is a feature reduction technique that looks for the most important parts of the features and it produce a new feature list that represent the original matrix but with reduced number of features. However, it produced poor results in this situation.

Step 4: Classifiers:

I implemented three classifiers, and the best one was logistic regression.

1. **Logistic regression:** The method of modeling the probability of a discrete result given an input variable is called logistic regression. I tried with many different parameters and the best results obtained when class_weight=1, max_iter=100 and with UnderSampling. The Random OverSampling and SMOTE made the classifier prone to overfitting. See result tables. The performance of it on f1 score was between 0.76-0.91 and on miner2 was between 0.48-0.76.

2. SVM:

Support-vector machines are supervised learning models that try to find the best hyperplane that separate data points. It is used for classification and regression. However, in this study, SVM gave very high f1 score 0.73-0.99, but very bad score on miner2 between 0.11-0.55 which means it is prone to overfitting.

3. Decision Tree:

Decision Tree are supervised learning models used for classification and regression. The goal is to learn simple decision rules from data attributes to develop a model that predicts the value of a target variable. However, it produced a good f1 scores 0.71-0.90, but poor score on miner2 between 0.44-0.69 which means it is prone to overfitting.

DecisionTree								
max_depth	6	4	4	3	4	4	3	3
class_weight	-	-	-	-	-	{0:1,1:2}	-	{0:1,1:2}
Random UnderSampling	-	-	-	-	-	Yes	Yes	Yes
Random OverSampling	Yes	Yes	Yes	Yes	Yes	-	-	-
SMOTE sampling	-	-	-	-	-	-	-	-
SVD	-	-	4	20	100	30	20	25
Runtime (Seconds)	0.22	0.16	0.01	0.007	0.008	0.006	0.010	0.002
F1_Score	0.90	0.83	0.83	0.95	0.96	0.79	0.71	0.80
Miner2 Score	0.59	0.63	0.44	0.71	0.67	0.47	0.57	0.69

SVM				
C	0.1	0.1	0.005	0.0005
kernel	Linear	Linear	Linear	Linear
class_weight	-	-	-	-
Random UnderSampling	Yes	Yes	-	-
Random OverSampling	-	-	-	Yes
SMOTE sampling	-	-	Yes	-
SVD	-	Comp=10	-	-
Runtime (Seconds)	0.14	0.01	6.8	5.8
1_Score	0.73	0.76	0.93	0.99
Miner2 Score	-	-	0.24	0.55

Logistic Regression								
C	1	1	0.10	0.10	0.10	0.10	0.10	1
class_weight	1	{0:1, 1:2}	{0:1, 1:2}	{0:1, 1:2}	{0:1, 1:3}	{0:1, 1:2}	-	{0:1, 1:2}
max_iter	100	100	100	100	100	100	100	100
Random UnderSampling	Yes	Yes	Yes	-	Yes	Yes	-	-
Random OverSampling	-	-	-	Yes	-	-	Yes	Yes
SMOTE sampling	-	-	-	-	-	-	-	-
SVD	-	-	-	-	-	Comp=2 or 10	-	-
Runtime (Seconds)	0.21	0.26	0.24	0.24	0.26	0.25	0.44	0.016
F1_Score	0.76	0.76	0.78	0.79	0.85	0.67	0.98	0.91
Miner2 Score	0.73	0.76	0.70	0.68	0.54	-		0.48