ECE 156B: HOMEWORK 4 PHASE 1

Vicki Chen
University of California, Santa Barbara

1 Introduction

In this assignment, we are given the task to generate classifications. We are given a dataset which contains 777 samples and each sample has 100 features. This dataset is different from previous experiments since it comprises of two classes of samples that are either positive or negative. there are no feature names and outputs are simply dependent on the amount of times positives show. The goal for this experiment is to uncover the cause for some special property observed on our sample data.
The task to classify this dataset is a lot harder since there is only one case of positive sample out of the 777 samples given. Training an imbalanced dataset is a task we have not yet encountered in this class. The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets. Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

Codalab Username: icxiv
Score: 46%

2 Method

2.1 Setup

My first step was to analyze the data and feature correlations. I ran a simple classification algorithm to classify the dataset first. Then I evaluated the results through applying a confusion matrix, which shows the correct and incorrect predictions for each class.

```
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print('Confusion matrix:\n', conf_mat)
```

```
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

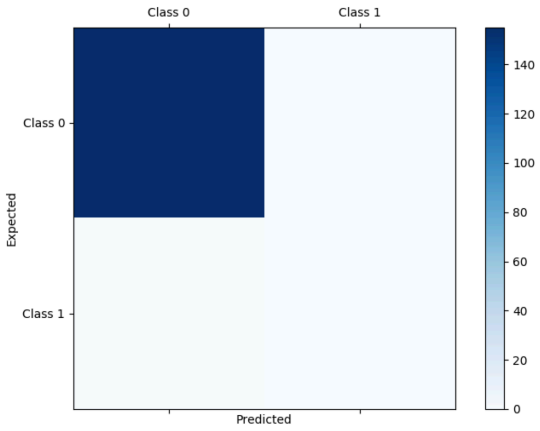
In the first row, the first column indicates how many classes 0 were predicted correctly, and the second column, how many classes 0 were predicted as 1. In the second row, we note that all class 1 entries were erroneously predicted as class 0

Therefore, the higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.

Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Accuracy of a model = (TP+TN) / (TP+FN+FP+TN)

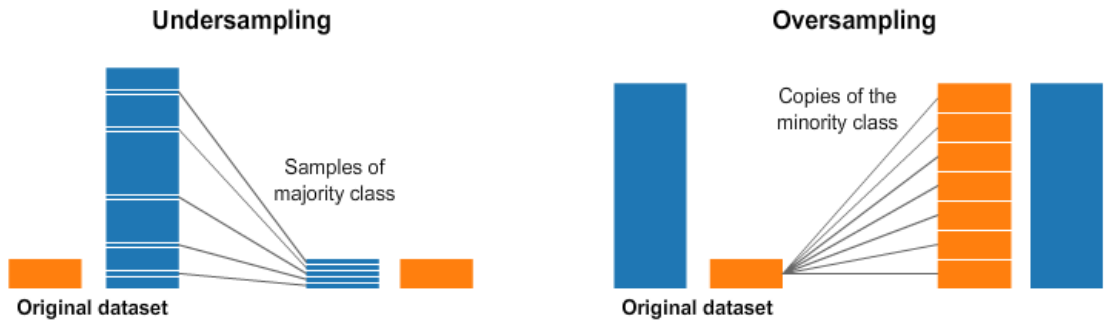


```
0    776
1      1
```

By generating a confusion matrix, we can see that the data is extremely imbalanced. This can generate problems when we're performing clustering algorithms, so the first step is to clean up the data by applying resampling.

2.2 Resampling

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and / or adding more examples from the minority class (over-sampling).



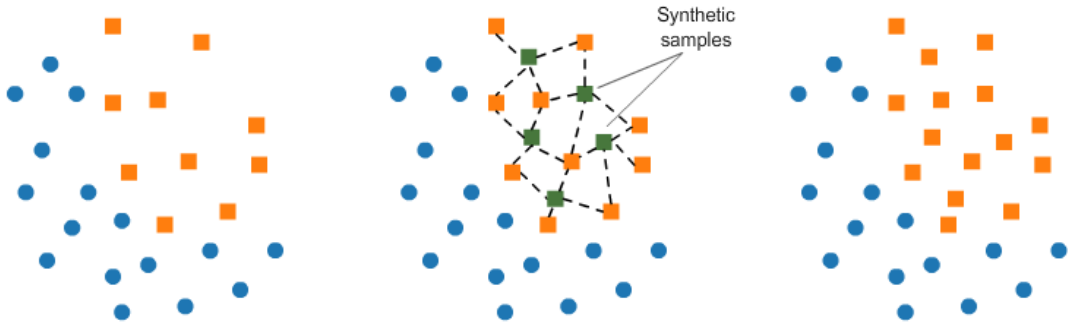
Despite the advantage of balancing classes, these techniques also have their weaknesses (there is no free lunch). The simplest implementation of over-sampling is to duplicate random records from the minority class, which can cause overfitting. In under-sampling, the simplest technique involves removing random records from the majority class, which can cause loss of information.

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
pass
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_resampled, y_resampled,
test_size = 0.2, random_state=0)

from imblearn.over_sampling import SMOTE
smote = SMOTE(ratio='minority')
X_sm, y_sm = smote.fit_sample(X, y)
plot_2d_space(X_sm, y_sm, 'SMOTE over-sampling')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
```

I tested out two different Resampling methods. The first one is RandomOverSampling which works by duplicating some of the original samples of the minority class. The second is SMOTE (Synthetic Minority Oversampling Technique) which consists of synthesizing elements for the minority class, based on those that already exist. It works randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.



2.3 Classification

After resampling the data, I ran the Naïve Byes classification on the dataset since it's an easy way of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem.

```
def mymodel(Xt, X, Y):
    Yp = np.ones(Xt.shape[0])
    ros = RandomOverSampler(random_state=0)
    X_resampled, y_resampled = ros.fit_resample(X, y)
    smt = SMOTETomek(ratio='auto')
    X_smt, y_smt = smt.fit_sample(X_resampled, y_resampled)
    plot_2d_space(X_smt, y_smt, 'SMOTE + Tomek links')
    clf = GaussianNB()
    model = clf.fit(X_smt,y_smt)
    Yp = model.predict(Xt)
    return Yp
```

3 Results

After submitting my model to codalab, the score I got back was 46%. When I performed cross validation on the dataset on my local environment, I was able to get the following percentages for test/training data:

KNN test data accuracy: 1.000000000

KNN train data accuracy: 1.000000000

Linear SVM test data accuracy: 1.000000000

Linear SVM train data accuracy: 1.000000000

Naive Bayes test data accuracy: 1.000000000

Naive Bayes train data accuracy: 1.000000000

LDA test data accuracy: .967845659

LDA train data accuracy: .979049154

This proved that there is a significant difference it how my actually algorithm is classifying the data and my model is extremely inaccurate.

4 Conclusion

4.1 Next Phase

The first phase of this experiment was difficult at first because of the lack of knowledge I had on working with imbalanced data sets. For the next phase, I want to test out

several different factors that might play a role in reshaping/reconstructing the dataset so that I can develop a model with higher accuracy. Here are some fixes and goals I will be looking into:

1. **Resample differently.** Oversample from the minority class and under-sample from the majority class, so I get a more balanced dataset.
2. **Improve Accuracy**. My main goal for the next phase will be to raise the accuracy score by continuously adding ways to improve the imbalanced dataset.
3. **Try different metrics.** other than correct vs wrong prediction. I will try Confusion Matrix or ROC curve. Accuracy is divided into sensitivity and specificity and models can be chosen based on the balance thresholds of the values.
4. **Anomaly Detection** techniques and models are often used as well.

References

https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html