MLBP

July 21, 2019

1 Using Voting Classfier and Sampling for Unbalanced Classfication Problem

1.0.1 Abstract

Music genre classification is the task of assigning genres to music. Not only, a music can belong to more than one category but there can be lot of variations based on artist, geographical locations etc which can make it diffcult to identify the genre of music based on accoustic properties. We were provided with an class imbalanced dataset of music meta features. We reduce the dimension of dataset by removing collinearity. Then we resample from the original dataset to create new dataset with balanced classes. We compare several classfiers using cross validation and finally employ a voting classfier to make the prediction. We achieve a test accuracy of 96% on the training data and 65% accuracy on Kaggle.

1.0.2 Keywords

`Voting Classfier`, `Resampling`, `Collinearity`, `Class Imbalance`.

1.0.3 Introduction

Music genre classfication is not only diffcult for machines but can also be diffcult for humans.Two different person can percieve the same music belonging to different genre. Also, we also listen to music and there is some new music coming out everyday from new artists so there are new songs coming up with unknown genres. Also, these music can have lot of variations depending upon the the artist, region, musical instruments etc.

Researchers have tried to identify music genres using timeseries because music is actually time series data with variation in timbre, pitch and rythm. In order to improve the result, some researchers have tried bag of words approach by taking the help of lyrics to identify the genre of song. Now, researchers are taking the help of the state of the art deep neural networks for classfying the genre. Gaussian Mixture Models are also an ideal choice.

Music genre classfication can help companies like spotify recommend new music to their users depending on their playlist. Identifying new genres can help music companies to reset their direction according to the trend because music always keep changing from generation to generation. Extended application can include identifying artists and piracy.

The data is split into two datasets: a training data set with 4363 songs, and a test set dataset with 6544 songs. Each song has 264 features, and there are 10 possible classes in total. The dataset is a custom subset of the Million Song Dataset, and the labels were obtained from AllMusic.com. So, this is a multiclass classfication problem. Let's find out more as we explore the data!

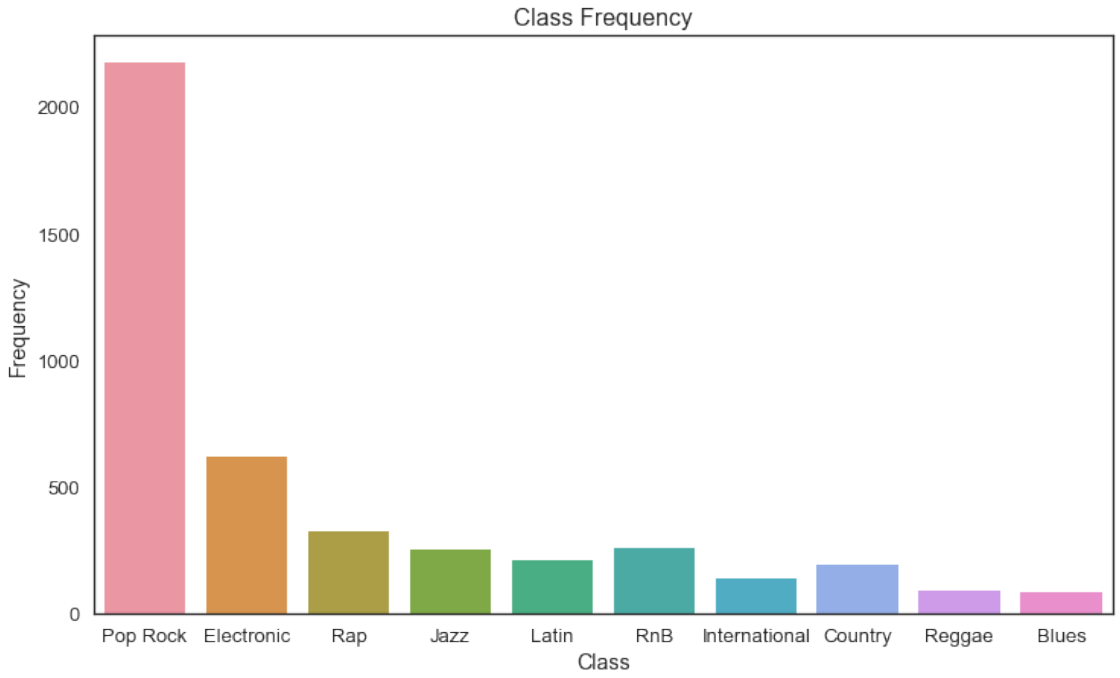
```
In [1]: import pandas as pd
        import numpy as np
        import os
        import seaborn as sns
        import matplotlib.pyplot as plt
        from imblearn.over_sampling import SMOTE
        plt.style.use('ggplot')
        import seaborn as sns
        sns.set(style="white", context="talk")
```

```
In [2]: X = pd.read_csv('Data/train_data.csv',header=None)
        y = pd.read_csv('Data/train_labels.csv',header=None).values.ravel()
        test = pd.read_csv('Data/test_data.csv',header=None)
```

Data Analysis and Preprocessing

```
In [88]: fig = plt.figure(figsize=(12,7))
         unique, counts = np.unique(y, return_counts=True)
         class_names = ['Pop Rock','Electronic','Rap','Jazz','Latin','RnB','International','Cou
         ax = sns.barplot(class_names,counts)
         ax.set_title('Class Frequency')
         ax.set_xlabel('Class')
         ax.set_ylabel('Frequency')
```

```
Out[88]: Text(0,0.5,'Frequency')
```



Problems with the Dataset The dataset has class imbalance problem which is evident from the above plot. Class imbalance is a major problem for classfication task and we can find many practical real life examples with it. One of such cases might be that the number of fradulent transactions is 1-2% compared to healthy transaction which can make it diffcult to detect fraud. However, we can tackle the class imbalance problem by adopting many strategies. * **Collect More Data** - This isn't possible for our case * **Use Different Performance metrics and algorithms** - We have tried different classfication algorithms in our project. As far as performance metric was concerened, AUC ROC can be used. AUC ROC stands for area under the receiver operating characteristic curve which can be used to measure the ability of a classfier to distinguish between different classes. However, in a multiclass classfication problem, it can be diffcult to interptret these curves. So, we resort to accuracy and log loss as suggested in the MLBP competition. * **Resampling the Dataset** - When training our model, we should have the training set with balanced classes. This can be done by oversampling the minority class or undersampling the majority class to have a class balance. Oversampling is preferred in when we have less data and undersampling is preferred when we have more data. * **Generating Synthetic Samples** - SMOTE or the Synthetic Minority Over-sampling Technique can create synthetic samples from minority class by adding some perturbation/random noise between two samples. * **Ensemble Methods** - Ensemble methods mainly comprise of Stacking, Boosting and Bagging. In stacking, the first level of classfiers are trained on the dataset and then second level of classfier, uses the prediction made by first level of classfiers to make the final prediction. In Bagging, we average the response of n models whereas in boosting, we take the weighted average of these n models. * **Regularization** - We can penalize the classfier more when it makes mistake in predicting the label for minority class.

Apart from class imbalance, the dataset also suffers from **Collinearity** which we found out by using **Linear Discriminant Analysis (LDA)** on the dataset. Collinearity is the condition in which one column or feature of the dataset can be predicted with the help of other columns. Collinearity can be tackled by calculating **variance inflation factor (VIF)**. Higher the VIF value, higher the collinearity. VIF is calculated for each explanatory variable/column and the column with highest value is removed. We keep repeating the process until collinearity is removed. **However, the process of removing collinearity was very slow, so we performed it seperately and have added the code in the appendix. We couldn't demonstrate the effect because removing collinearity from the test data was also very slow and we couldn't complete in time.**

Data Preprocessing and cleaning

```
In [97]: # Just for demonstration

         data_without_collinearity = pd.read_csv('col_free.csv')
         data_without_collinearity.shape
```

```
Out[97]: (4363, 137)
```

```
In [98]: data_without_collinearity.head()
```

```
Out[98]:    Unnamed: 0       24      25       26      27       28      29       30  \
         0           0   580.90  2149.8  1543.60  1046.4  1588.90  1388.5  1527.40
         1           1  2038.80  2753.7  3077.10  1737.4  1074.10  1132.7  1457.20
         2           2  1018.50  3062.3  2967.10  4394.1  2321.00  2436.1  2200.30
         3           3  1282.60  2919.8  2120.50  2346.2  1860.30  1762.5  1427.00
```

```
4           4   461.41  1111.4   815.69   528.6   440.54   337.4   360.16

        31       32    ...          254      255       256       257  \
0  1135.00  1244.20    ...     0.216490  0.36548  0.093584  0.166870
1  2895.60  1763.70    ...     0.100670  0.14739  0.102560  0.213040
2  5039.50  2160.60    ...     0.126760  0.36321  0.114200  0.223780
3  1215.40   962.47    ...     0.096479  0.28950  0.074124  0.201580
4   277.18   260.94    ...     0.138340  0.38266  0.079402  0.063495

        258       259       260       261       262       263
0  0.083426  0.118090  0.089792  0.074371  0.073162  0.059463
1  0.082041  0.080967  0.076450  0.052523  0.052357  0.055297
2  0.100770  0.186910  0.067270  0.061138  0.085509  0.049422
3  0.049032  0.130210  0.045800  0.080885  0.148910  0.042027
4  0.053717  0.086750  0.062090  0.048999  0.033159  0.070813

[5 rows x 137 columns]
```

After removing collinearity, the dataset had 137 features. However, we haven't actually used it in final methodology. (P.S.: We forgot to transform the test data after fitting and transforming the training data. By the time, we realised our mistake, it was too late to complete the task on time. Bad Luck!) We tried different standardization techniques and got the best result with **Quantile Transformer**. Quoting sklearn's documentation, "This method transforms the features to follow a uniform or a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is therefore a robust preprocessing scheme."

```
In [7]: from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.preprocessing import Normalizer
        from sklearn.preprocessing import RobustScaler
        from sklearn.preprocessing import QuantileTransformer
```

```
(4363, 137)
```

We tried different sampling techniques but none of them improved our results. They increased our training accuracy to 98% and test accuracy to 96% but the model didn't generalize well on the kaggle data.

```
In [38]: from imblearn.over_sampling import SMOTE, ADASYN
         from imblearn.combine import SMOTEENN
         from imblearn.combine import SMOTETomek
         from imblearn.under_sampling import ClusterCentroids
         from imblearn.under_sampling import RandomUnderSampler

         # Undersampling techniques
         #X_gen, y_gen = ClusterCentroids().fit_resample(X, y)
         #X_sel, y_sel = RandomUnderSampler().fit_resample(X, y)
```

```
        # Oversampling techniques
        #X_smote, y_smote = SMOTE().fit_resample(X, y)
        #X_adasyn, y_adasyn = ADASYN().fit_resample(X, y)

        # Combination of oversampling and undersampling
        #X_smoteenn, y_smoteenn = SMOTEENN().fit_resample(X, y)
        X_smotetomek, y_smotetomek = SMOTETomek().fit_resample(X, y)
```

In [40]:
```
# Transforming the collinearity removed data

transformer = QuantileTransformer()
transformed_train = transformer.fit_transform(X_smotetomek)
transformed_test = transformer.transform(test)
```

In [41]:
```
transformed_train.shape
```

Out[41]:
```
(21762, 264)
```

1.0.4 Methodology

Below, we experimented with different methodology one after the other: * Remove collinearity * Apply SMOTETomek. SMOTE, discussed earlier, alone can result in noisy samples. So we apply SMOTE to the dataset and clean it with T-Link Algorithm. https://www.omicsonline.org/open-access/classification-of-imbalance-data-using-tomek-link-tlink-combined-with-random-undersampling-rus-as-a-data-reduction-method-2229-8711-S1111.pdf * Apply Quantile transformation * Compare single classfiers with stratfied k fold cross validation * Combine multiple classfiers to from voting classfier using soft voting

The best results were obtained with the methodology below: * Apply SMOTETomek. SMOTE, discussed earlier, alone can result in noisy samples. So we apply SMOTE to the dataset and clean it with T-Link Algorithm. https://www.omicsonline.org/open-access/classification-of-imbalance-data-using-tomek-link-tlink-combined-with-random-undersampling-rus-as-a-data-reduction-method-2229-8711-S1111.pdf * Apply Quantile transformation * Compare single classfiers with stratfied k fold cross validation * Combine multiple classfiers to from voting classfier using soft voting

For model selection, we initially benchmark different classfiers on the transformed dataset.

Performance of Single Classfier

In [17]:
```
# Multiclass classification
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors.nearest_centroid import NearestCentroid
from sklearn.neural_network import MLPClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier

# Multiclass classification as one vs one
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier

# Multiclass classification as one vs all
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
kernel = 1.0 * RBF(1.0)
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import Perceptron
from sklearn import linear_model

# Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
```

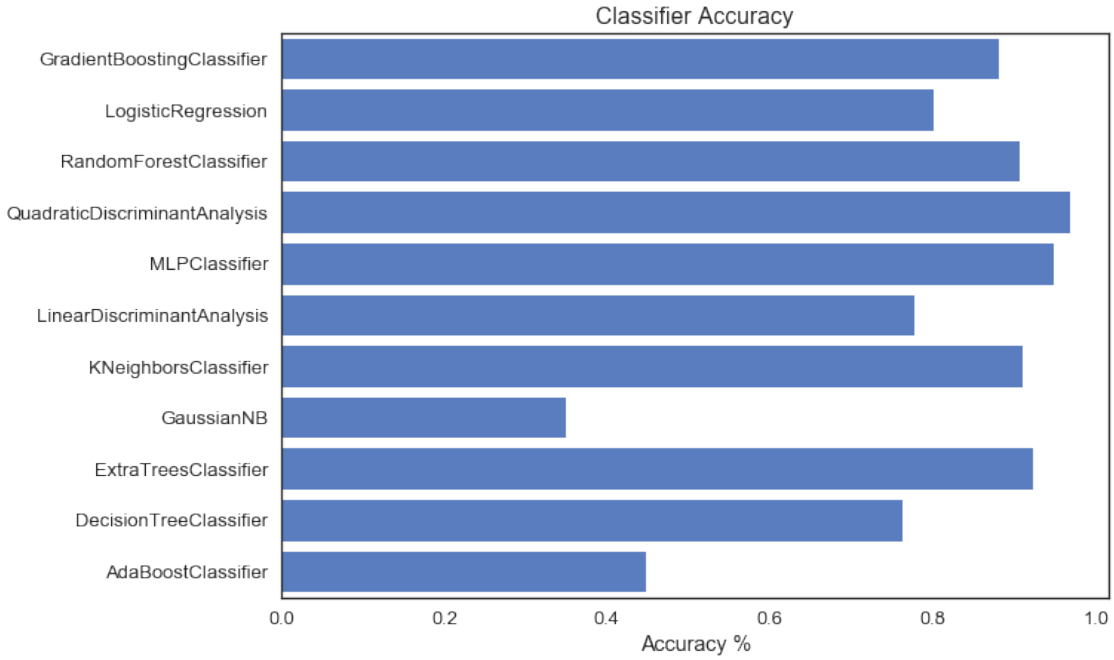
Let's calculate the classfication metric on these models to get an idea how they perform on the dataset. While testing, we found out that many out the classfier don't support predicting probablities which reduced the number of classfiers used in the code section below. Also, we are using stratified split so that in each fold the percentage of samples for each class remains balanced. Shuffling, however, doesn't guarantee that all the folds will be different.

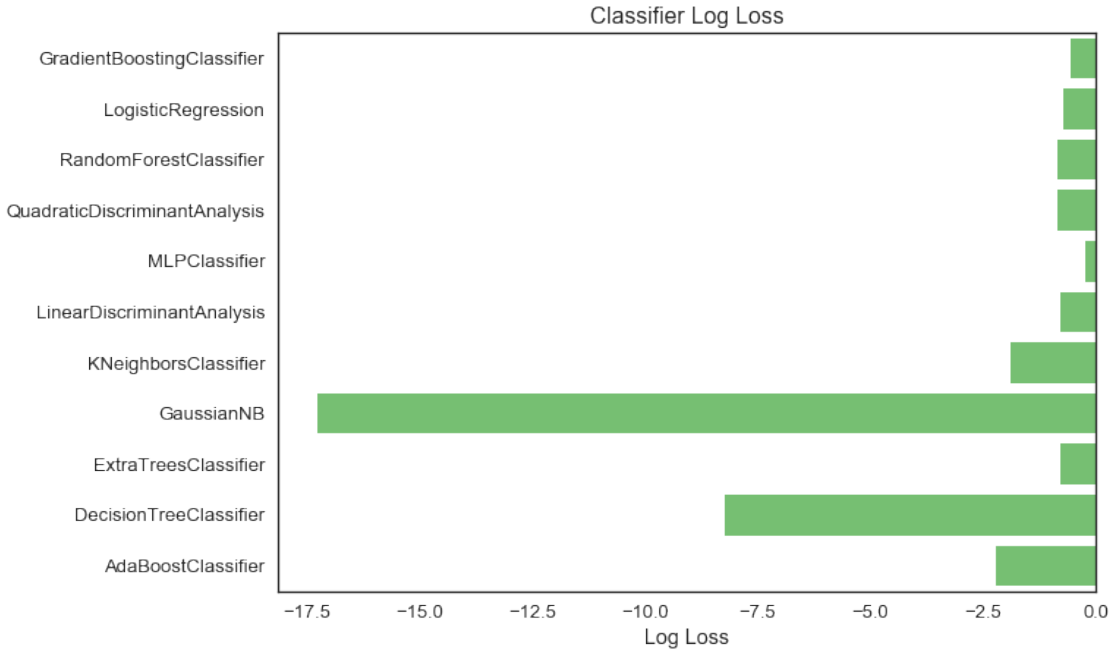
```
In [48]: from sklearn.model_selection import StratifiedShuffleSplit
         from sklearn.model_selection import cross_validate
         np.random.seed(2)

         classfiers = [GradientBoostingClassifier(),LogisticRegression(),RandomForestClassifie
                       QuadraticDiscriminantAnalysis(), MLPClassifier(),LinearDiscriminantAnal
                       KNeighborsClassifier(),GaussianNB(),ExtraTreesClassifier(),DecisionTree
                       AdaBoostClassifier()]

         sss = StratifiedShuffleSplit(n_splits=2, test_size=0.3, random_state=0)
         criterion = ['accuracy','neg_log_loss']
         estimators = []

         classfier_name = []
         accuracy_scores = []
         neg_log_loss_scores = []
         for clf in classfiers:
             print(str(clf.__class__.__name__))
             score = cross_validate(clf,transformed_train,y_smotetomek,scoring=criterion,cv=ss
             classfier_name.append(str(clf.__class__.__name__))
             accuracy_scores.append(np.mean(score['test_accuracy']))
             neg_log_loss_scores.append(np.mean(score['test_neg_log_loss']))
```

```
GradientBoostingClassifier
LogisticRegression
RandomForestClassifier
QuadraticDiscriminantAnalysis
MLPClassifier
LinearDiscriminantAnalysis
KNeighborsClassifier
GaussianNB
ExtraTreesClassifier
DecisionTreeClassifier
AdaBoostClassifier
```

Sklearn uses neg_log_loss as scoring scheme in order to be consistent with other scoring schemes. Values such as accuracy that need to be maximized are kept positive whereas log loss which needs to be minimized is kept negative. So, lower the value of neg_log_loss, the better the classfier is able to predict the classes.

```
In [90]: fig = plt.figure(figsize=(10,7))

         sns.set_color_codes("muted")
         sns.barplot(x=accuracy_scores, y=classfier_name, color="b")

         plt.xlabel('Accuracy %')
         plt.title('Classifier Accuracy')
         plt.show()

         fig = plt.figure(figsize=(10,7))

         sns.set_color_codes("muted")
         sns.barplot(x=neg_log_loss_scores, y=classfier_name, color="g")

         plt.xlabel('Log Loss')
         plt.title('Classifier Log Loss')
         plt.show()
```





Based on the above plot, we choose classfiers which perform well on both metrics. The results are for the best methodology process in which we have used resampling and applied transformation.

```
In [59]: accuracy_scores
```

```
Out[59]: [0.8808393322101393,
          0.8019604839944862,
          0.9061111962015622,
          0.9679123908714964,
          0.9494562720171542,
          0.7782202481237556,
          0.9096339408791545,
          0.3493643743299127,
          0.9231122683412467,
          0.7621381528564865,
          0.4474651554602542]
```

Voting Classfier

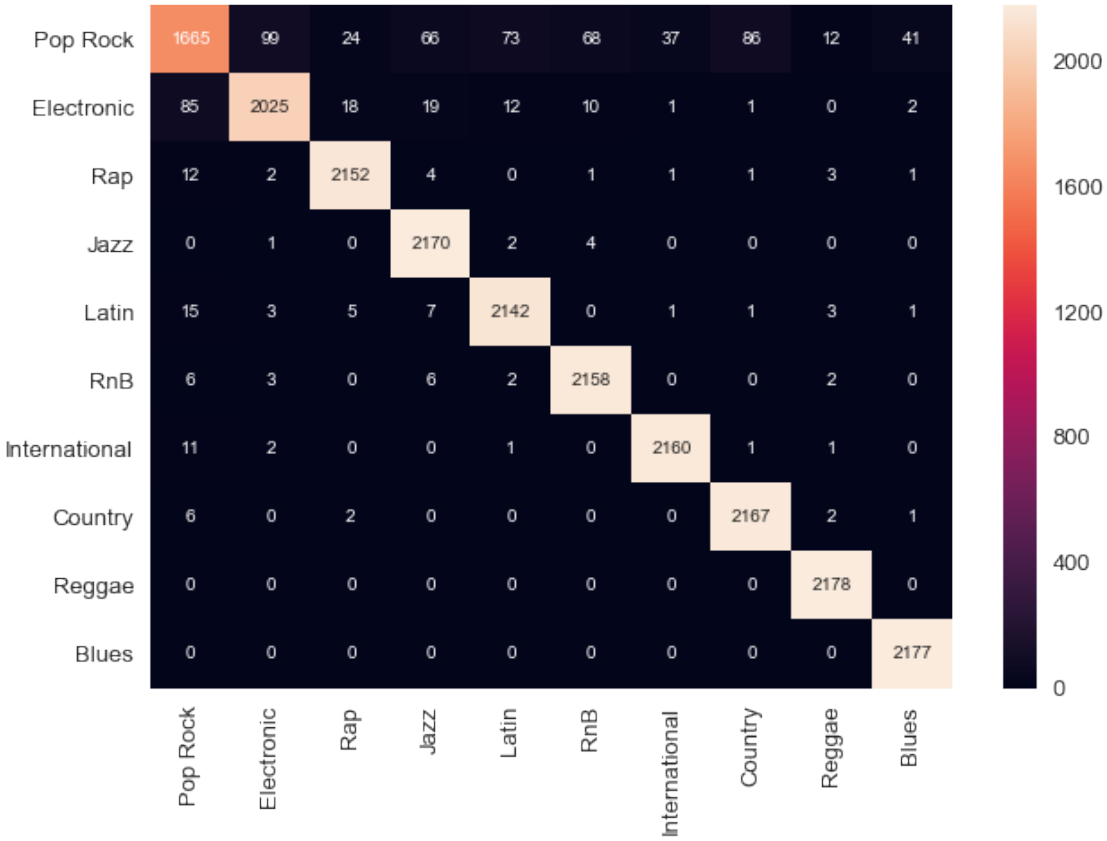
```
In [60]: from sklearn.ensemble import VotingClassifier
         from sklearn.model_selection import cross_val_predict

         classfiers2 = [QuadraticDiscriminantAnalysis(), MLPClassifier(),GaussianNB(),\
                        DecisionTreeClassifier(),AdaBoostClassifier()]
         classfiers3 = [x for x in classfiers if x not in classfiers2]
         list_of_estimators = []
         for clf in classfiers:
             list_of_estimators.append((str(clf.__class__.__name__),clf))

         voting_classfier = VotingClassifier(estimators=list_of_estimators,voting='soft')
         y_pred = cross_val_predict(voting_classfier, transformed_train, y_smotetomek, cv=3)
         conf_mat = confusion_matrix(y_smotetomek, y_pred)
```

```
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
```

```
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/neural_network/mu
  % self.max_iter, ConvergenceWarning)
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear.")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/neural_network/mu
  % self.max_iter, ConvergenceWarning)
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear.")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
```

```
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/neural_network/mu
  % self.max_iter, ConvergenceWarning)
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear.")
```

```
In [92]: class_names = ['Pop Rock','Electronic','Rap','Jazz','Latin','RnB','International','Cou
         df_cm = pd.DataFrame(np.array(conf_mat), index=class_names, columns=class_names )
         fig = plt.figure(figsize=(10,7))
         ax = sns.heatmap(df_cm,annot=True, fmt="d")
```




```
In [61]: score = cross_validate(voting_classfier,transformed_train,y_smotetomek,scoring=criter
```

```
In [80]: score['test_accuracy'].mean()
```

```
Out[80]: 0.9598713432378618
```

```
In [83]: score['test_neg_log_loss'].mean()
```

```
Out[83]: -0.4477984899021737
```

Here, with stratified cross validation, we are achieving test accuracy of nearly 96% and a neg_log_loss of -0.44. **On Kaggle, we achieved accuracy of 0.65461 and a log loss of 0.18243.**

Predicting from final Model

```
In [57]: eclf = voting_classfier.fit(transformed_train,y_smotetomek)
         accuracy_pred = eclf.predict(transformed_test)
         log_prob_pred = eclf.predict_proba(transformed_test)
```

```
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
/Users/rohan/.virtualenvs/userelicitation/lib/python2.7/site-packages/sklearn/discriminant_anal
  warnings.warn("Variables are collinear")
```

CSV Generation for Kaggle

```
In [58]: accuracy_sub = []
         i = 1
         for value in accuracy_pred:
             df={'Sample_id':i,'Sample_label':value}
             i = i+1
             accuracy_sub.append(df)
         accuracy_dataframe = pd.DataFrame(accuracy_sub)
         accuracy_dataframe.to_csv('accuracy.csv',columns=['Sample_id','Sample_label'],index=Fa

         log_loss = []
         z = 1
         for value in log_prob_pred:
             df={'Sample_id':z,'Class_1':value[0],'Class_2':value[1],'Class_3':value[2],'Class_
                'Class_5':value[4],'Class_6':value[5],'Class_7':value[6],'Class_8':value[7],'Cl
                'Class_10':value[9],}
             z = z+1
             log_loss.append(df)
         log_loss_dataframe = pd.DataFrame(log_loss)
```

```
log_loss_dataframe.to_csv('log_loss.csv',columns=['Sample_id','Class_1','Class_2','Cl
                                                  'Class_5','Class_6','Class_7','Class_
```

1.0.5 Discussion and Conclusion

Most of the real life problems are like finding needle in a haystack because of imbalanced classes. We have already proposed several techniques to handle unbalanced classes. First of all, removing collinearity from the data increased our test accuracy from 65% to around 76% in cross validation. Unfortunately, we couldn't show the results because the process was very slow. Adding the quantile transformation further increased the test accuracy to around 78% in cross validation. Feature scaler, Robust scaler and Quantile scaler provided us nearly same results.

We also tried to further increase the performance by sampling from the dataset. Undersampling and Oversampling method such as SMOTE provided an test accuracy of around 57% and 60% in cross validation. The resampling of the data increased the dataset size which was again a bottleneck for us. Also, as were using voting classfier, it was very slow because it had to fit all the classfiers. We could have selected a subset of classfiers with hard voting but it didn't offer much improvement. The voting classfier also required to be soft voting in order to predict probablities for each class(log loss).

Our kaggle performance was also not very impressive because of the bias-variance trade off. Using all the different kind of techniques such as removing collinearity, resampling dataset and using voting classfier made our model complex. Complex models have low bias but high variance. Bias is the error and variance is the variablity of the prediction. Our complex model performed very well in cross validation which is 96% test accuracy but didn't generalize well on the test data.

Regarding the metrics, Accuracy assigns equal severity to all errors in classfication. In the problem of unbalanced class, we would like to penalize the classfier more when it makes a wrong prediction regarding the minority class. Log loss suffers from the same pitfall as accuracy. Although, it uses class prediction for each class but weights each type of misclassification equally. Log loss can be generalized to take into account misprediction costs adjusted by class weights. AUCROC curve is better but diffcult to interpret for multiclass classfication problems.

We have many suggestions for improvement to the above project: * We could have optimized each classfier using grid search. * We could have use semi-supervised learning of the test data. We would use the voting classfier to predict the test data labels. Then, re-train the voting classfier based on training data and test data. * We also wanted to try other techniques such as Bagging.

1.0.6 References

- https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
- https://www.kaggle.com/ffisegydd/sklearn-multicollinearity-class
- http://scikit-learn.org/stable/documentation.html
- https://imbalanced-learn.readthedocs.io/en/stable/

1.0.7 Appendix

Code for removing collinearity

```
In [ ]: import numpy as np
        import pandas as pd
        from subprocess import check_output
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import Imputer
from statsmodels.stats.outliers_influence import variance_inflation_factor

X = pd.read_csv('Data/train_data.csv',header=None)
y = pd.read_csv('Data/train_labels.csv',header=None)

from statsmodels.stats.outliers_influence import variance_inflation_factor

class ReduceVIF(BaseEstimator, TransformerMixin):
    def __init__(self, thresh=5.0, impute=True, impute_strategy='median'):
        # From looking at documentation, values between 5 and 10 are "okay".
        # Above 10 is too high and so should be removed.
        self.thresh = thresh

        # The statsmodel function will fail with NaN values, as such we have to impute
        # By default we impute using the median value.
        # This imputation could be taken out and added as part of an sklearn Pipeline.
        if impute:
            self.imputer = Imputer(strategy=impute_strategy)

    def fit(self, X, y=None):
        print('ReduceVIF fit')
        if hasattr(self, 'imputer'):
            self.imputer.fit(X)
        return self

    def transform(self, X, y=None):
        print('ReduceVIF transform')
        columns = X.columns.tolist()
        if hasattr(self, 'imputer'):
            X = pd.DataFrame(self.imputer.transform(X), columns=columns)
        return ReduceVIF.calculate_vif(X, self.thresh)

    @staticmethod
    def calculate_vif(X, thresh=5.0):
        # Taken from https://stats.stackexchange.com/a/253620/53565 and modified
        dropped=True
        while dropped:
            variables = X.columns
            dropped = False
            vif = [variance_inflation_factor(X[variables].values, X.columns.get_loc(va

            max_vif = max(vif)
            if max_vif > thresh:
                maxloc = vif.index(max_vif)
                X = X.drop([X.columns.tolist()[maxloc]], axis=1)
                dropped=True
```

```
        return X

transformer = ReduceVIF()

X = transformer.fit_transform(X, y)
```