In [21]:
```
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import decomposition
import matplotlib.pyplot as plt
from sklearn2pmml.pipeline import PMMLPipeline
from sklearn_pandas import DataFrameMapper
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import *
from feature_selector import FeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.pipeline import Pipeline
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE
```

CS422 Project - Purvaj Desai

Abstract

The basic idea I learned during this project is that while dealing with any of the projects/work related to data we should be more focused on the data given to us rather than just getting outputs and better accuracies. If I talk about this particular project at first I started it treating like a normal datamining problem and started building a model using different machine learning algorithm which just led me to poor accuracy score and several problems like model overfitting. And then I tried different approach and started researching on the dataset provided and started understanding the hidden nature of data. During this project I learned that rather than blindly following the orthodox way of solving data mining problems we should better understand the problem on hand and look a different wayout on solving that. I learned that there are lot of stuff available on internet, so we just need to understand the CONCEPT out of it and implement according to what our dataset needs. Another things was models should not just be accurate but rather it should be efficient. Also I learned about some new things like Pipelines, ONNX, and different python liberaries which I haven't even heard before this.

Well I tried my very best with whatever research I was able to do in given time frame. But if the project where to be continued we can try making multiple pipelines and then merge them together to get a better and efficent performance. Also we can resample data with different ratios.

Overview

Problem statement: The objective of this project is to build a model that generalizes well out of sample

Relevant literature: See References

Proposed methodology: The first step of my model using Correlation to manually select a few features to keep and use in the model. The data is then scaled and principal component analysis (PCA) is performed on it. The final classifier uses a SMOTE with LogisticRegression.

Reading Data

In [22]:
```
df = pd.read_csv("C:\\Users\\windows10\\Desktop\\IIT\\Sem 1\\Data Mining\\Project\\data
df = shuffle(df)

X = df.drop('Class',axis = 1)
Y = df['Class']

class_1 = df[df['Class'] ==1]
class_2 = df[df['Class'] ==2]
class_3 = df[df['Class'] ==3]
Tot = len(class_1)+len(class_2)+len(class_3)

print("Class 1:%d , Percentage : %.3f%%"%((len(class_1),((len(class_1))/Tot)*100)))
print("Class 2:%d , Percentage : %.3f%%"%((len(class_2),((len(class_2))/Tot)*100)))
print("Class 3:%d , Percentage : %.3f%%"%((len(class_3),((len(class_3))/Tot)*100)))
```

```
Class 1:199992 , Percentage : 16.666%
Class 2:599228 , Percentage : 49.936%
Class 3:400780 , Percentage : 33.398%
```

Here I have imported the dataset and used Shuffle() to shuffle the data because there is a possibility that a particular class is there only at the end of dataset and while training data that part won't be there in training set and our model would be biased and not effective so shuffle() will randomly reaarange data. Also I have looked at the percentage of Classes present in dataset and from this I concluded that the given dataset is Imbalanced dataset, as we can see almost 50% of the dataset belongs to class 2 and just 16% to class 1 and so I approached the problem keeping this in mind.

Data Processing and Analysis

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
A        0
B        0
C        0
D        0
E        0
F        0
G        0
H        0
I        0
J        0
K        0
L        0
M        0
N        0
O        0
Class    0
dtype: int64
```

I then checked for any null values in the dataset but there are no null values in our dataset.

Principal Component Analysis

In [24]:
```
sc = StandardScaler()
```

```
df_scaled = sc.fit_transform(df.drop('Class', axis=1))
df_scaled = pd.DataFrame(df_scaled)

pca_test = PCA(n_components=15)
pca_test.fit_transform(df_scaled)

pca_test.explained_variance_ratio_

df_scaled
```

Out[24]:

	0	1	2	3	4	5	6	7	
0	-0.691417	0.080524	-0.573349	0.708062	-0.775994	-0.170327	-0.440053	-0.749047	-0.95491
1	-0.660889	-0.062193	-0.555306	0.804625	-0.756470	-0.280093	-0.500947	-0.641367	-0.90331
2	-0.676110	0.437521	-0.508816	0.720759	-0.762416	-0.300628	-0.535385	-0.751394	-1.07865
3	-0.865114	-2.100622	-1.088193	-2.386775	-0.611016	-1.601475	-1.214715	-0.536836	0.50862
4	-0.652078	0.682769	-0.600312	0.687750	-0.758895	-0.313001	-0.489122	-0.758205	-0.95825
...	...	...	...	...	...	...	...	...	.
1199995	-0.668339	0.303151	-0.544646	0.768045	-0.724872	-0.300526	-0.404543	-0.734180	-0.93869
1199996	-0.854229	-1.957951	-1.034573	-1.965295	-0.589254	-1.708085	-1.256635	-0.547817	0.44036
1199997	1.357778	0.319297	1.257549	-0.028138	1.391524	1.307277	1.209148	1.395215	1.28988
1199998	-0.647929	0.130119	-0.533364	0.689072	-0.768578	-0.381756	-0.495110	-0.791983	-1.03142
1199999	-0.851585	-1.714079	-0.981684	-1.995224	-0.698924	-1.666350	-1.273332	-0.459111	0.34298

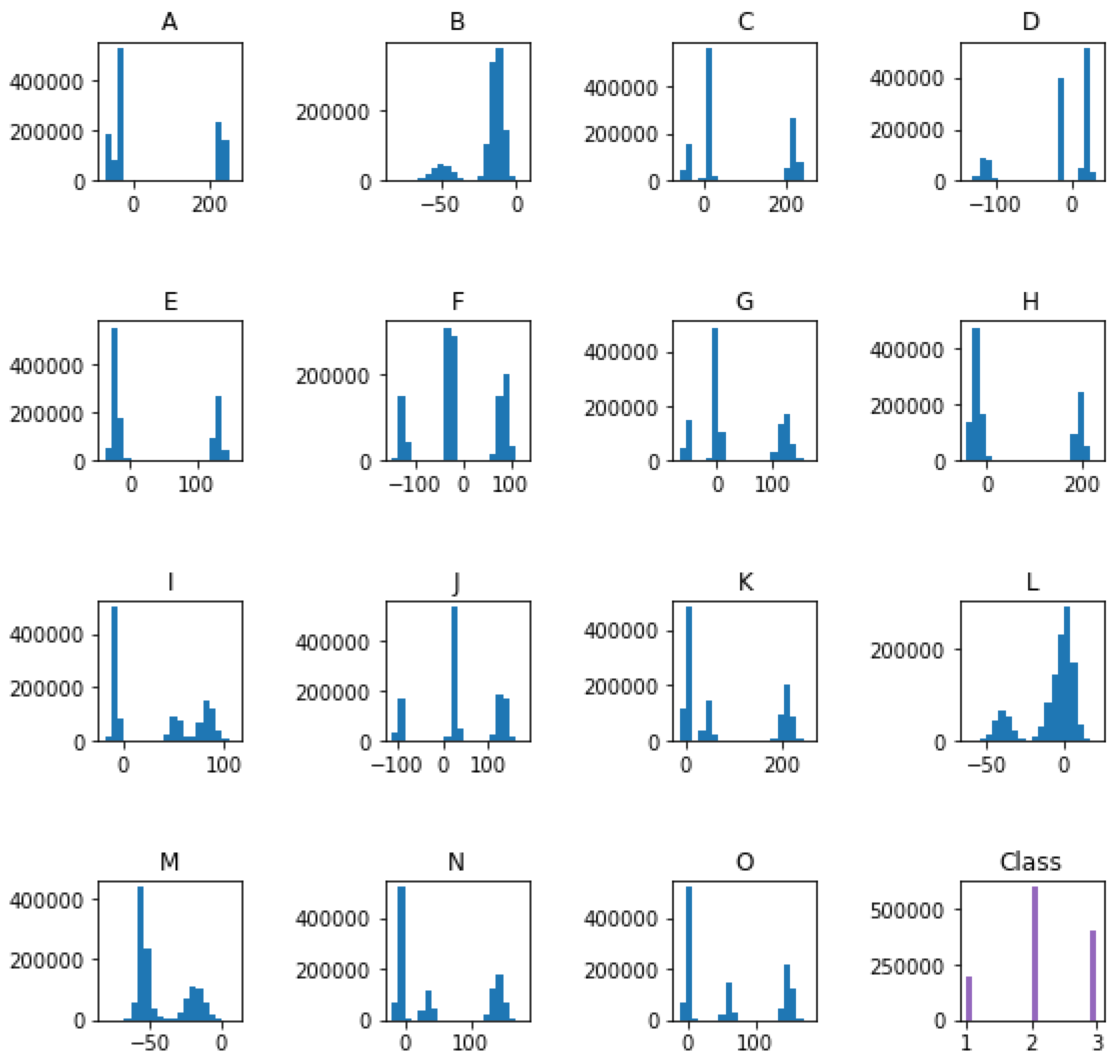
1200000 rows × 15 columns

I then Scaled the data and used Principal Component Analysis to discover optimal number of Principal Component is need to use througout the project. And from the results I decided to go with 2 principal components as using PC1 and PC2 we get 97% of information.

In [25]:

```
features = 'ABCDEFGHIJKLMNO'
fig1 = plt.figure()
for i in range(1,16):
    fig1.add_subplot(4,4,i)
    plt.hist(df[features[i-1:i]], bins=20)
    plt.title(features[i-1:i])
    fig1.add_subplot(4,4,16)
    plt.hist(df['Class'], bins=20)
    plt.title('Class')
    fig1.subplots_adjust(hspace=1, wspace=1)
    fig1.set_figheight(9)
    fig1.set_figwidth(9)
```

```
<ipython-input-25-e0874754092a>:7: MatplotlibDeprecationWarning: Adding an axes using th
e same arguments as a previous axes currently reuses the earlier instance.  In a future
version, a new instance will always be created and returned.  Meanwhile, this warning ca
n be suppressed, and the future behavior ensured, by passing a unique label to each axes
instance.
  fig1.add_subplot(4,4,16)
```



Using visual inspection, I determined there to be close relationships among the following features:

A,E,H,I

B

C,G

D

F,J

K,N

M

O

So I took one feature from each group because only one feature would be needed from that group as they show similar trend. And if we choose all the feature it might have biased results.

```
In [26]: features_1 = ['E','H','I','G','J','N','Class']
         X = df.drop(features_1, axis = 1)
         Y = df['Class']
```

```
In [27]: X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state
```

```
In [28]: dummy = DummyClassifier(strategy = 'most_frequent')
         pipeline1 = Pipeline(steps=[
             ('StandardScaler', StandardScaler()),
             ('PCA',PCA(n_components = 2)),
             ('Classifier',dummy)])
         pipeline1.fit(X_train,y_train)
         predictions = pipeline1.predict(X_test)

         print(classification_report(y_test, predictions))
```

```
C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           1       0.00      0.00      0.00     59879
           2       0.50      1.00      0.66    179321
           3       0.00      0.00      0.00    120800

    accuracy                           0.50    360000
   macro avg       0.17      0.33      0.22    360000
weighted avg       0.25      0.50      0.33    360000

C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

DummyClassifier: This classifier is useful as a simple baseline to compare with other (real) classifiers.

I used DummyClassifier() in order to see how the accuracy will be and we can see even without training the data we are getting the almost 50% accuracy. And so I researched more about Imbalanced dataset and got the insights that imbalanced dataset's model should not be relied on accuracy because intuitions for classification accuracy developed on balanced class distributions will be applied and will be wrong.

```
In [29]: rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

         pipeline2 = Pipeline(steps=[
             ('StandardScaler', StandardScaler()),
             ('PCA',PCA(n_components = 2)),
             ('classifier',rfc)])
         pipeline2.fit(X_train,y_train)
         predictions = pipeline2.predict(X_test)

         print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support
```

```
           1       0.17      0.12      0.14     59879
           2       0.50      0.62      0.55    179321
           3       0.34      0.26      0.30    120800

    accuracy                           0.42    360000
   macro avg       0.33      0.33      0.33    360000
weighted avg       0.39      0.42      0.40    360000
```

Then I compare this to RandomForestClassifier, an actual trained classifier and our accuracy decreased as compared to Dummy Classifier above.And we can see the accuracy decreased and so we can't rely just on accuracy. In imbalanced data we need to look at f1 score,recall and precisiom value to better the model.

In [30]:
```
lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)
pipeline3 = Pipeline(steps=[
    ('StandardScaler', StandardScaler()),
    ('PCA',PCA(n_components = 2)),
    ('regressor',lr)])
pipeline3.fit(X_train,y_train)
predictions = pipeline3.predict(X_test)

print(classification_report(y_test, predictions))
```

```
C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           1       0.00      0.00      0.00     59879
           2       0.50      1.00      0.66    179321
           3       0.00      0.00      0.00    120800

    accuracy                           0.50    360000
   macro avg       0.17      0.33      0.22    360000
weighted avg       0.25      0.50      0.33    360000

C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\windows10\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_classi
fication.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Then I tried with a different model to look if we can improve results and tried LogisticRegression and got better results compared to RandomForestClassifier.

In [31]:
```
X = pd.concat([X_train, y_train], axis=1)
class_1 = X[X['Class'] ==1]
class_2 = X[X['Class'] ==2]
class_3 = X[X['Class'] ==3]

class_2_downsampled = resample(class_2,replace=True, n_samples=len(class_3), random_sta
class_1_upsampled = resample(class_1,replace=True, n_samples=len(class_3), random_state
sampled = pd.concat([class_1_upsampled,class_2_downsampled, class_3])

sampled.Class.value_counts()
```

Out[31]:
```
1    279980
2    279980
3    279980
Name: Class, dtype: int64
```

Then I tried another method called Sampling. So here I sampled the data equally (equal number of data in each class) using Up-sampling and downsampling.

In [34]:
```
X_train = sampled.drop('Class', axis = 1)
y_train = sampled.Class

lrc_sampled = LogisticRegression(solver='liblinear').fit(X_train, y_train)
pipeline2 = Pipeline(steps=[
    ('StandardScaler', StandardScaler()),
    ('PCA',PCA(n_components = 2)),
    ('regressor',lrc_sampled)])
pipeline2.fit(X_train,y_train)
predictions = pipeline2.predict(X_test)

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           1       0.17      0.17      0.17     59879
           2       0.50      0.67      0.57    179321
           3       0.34      0.16      0.22    120800

    accuracy                           0.42    360000
   macro avg       0.34      0.34      0.32    360000
weighted avg       0.39      0.42      0.39    360000
```

I then trained the model and again performed LogisticRegression to see if I get better results and I did get better precision and F1 score. But still to further confirm model I used SMOTE for imbalanced dataset because in sampling we might loose data as it adds as well as remove data in the process we might loose the information.

Final Model: Training and Validation

In [36]:
```
features_1 = ['E','H','I','G','J','N','Class']
X = df.drop(features_1, axis=1)
Y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=2

sm = SMOTE(random_state=27)
X_train, y_train = sm.fit_resample(X_train, y_train)

smote_final = LogisticRegression(solver='liblinear').fit(X_train, y_train)
final_pipeline = Pipeline(steps=[
    ('StandardScaler', StandardScaler()),
    ('PCA',PCA(n_components = 2)),
    ('regressor',smote_final)])
final_pipeline.fit(X_train,y_train)
predictions = final_pipeline.predict(X_test)

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           1       0.17      0.17      0.17     59879
           2       0.50      0.67      0.57    179321
           3       0.34      0.16      0.22    120800

    accuracy                           0.42    360000
   macro avg       0.33      0.33      0.32    360000
weighted avg       0.39      0.42      0.39    360000
```

I got similar output for SMOTE as well but I would rather go with SMOTE because SMOTE creates synthetic observations based upon the existing minority observations so we might won't loose much data as compared to Sampling in which the data is replicated which might make model biased.

In [37]:
```
from sklearn.model_selection import cross_validate
cross_validate(final_pipeline, df.drop('Class',axis=1), df['Class'], cv=6)
```

Out[37]:
```
{'fit_time': array([7.38550091, 7.17494488, 6.63760519, 6.62628031, 6.53290009,
        6.40003181]),
 'score_time': array([0.08758402, 0.09373999, 0.09793758, 0.07246709, 0.07295299,
        0.07811809]),
 'test_score': array([0.49936 , 0.49936 , 0.499355, 0.499355, 0.499355, 0.499355])}
```

In [38]:
```
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input', FloatTensorType([None, 2]))]
onx = convert_sklearn(final_pipeline, initial_types=initial_type)
with open("smote_logistic_regression.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

Conclusion

My end results shows that I have much better model than I started and the main reason for not looking at accuracy and focusing on recall and f1 score is because of the given dataset(imbalanced) which has higher number of particular class and most of the machine learning algorithms works better when we have equally distributed classes. And Low recall indicates a high number of false negatives and higher recall indicates higher true positives. Positive result is that my model has somewhat better recall and f1 score when tested and caveat is that while generating synthetic examples, SMOTE does not take into consideration neighboring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise.

Refrences

- https://towardsdatascience.com/classification-of-unbalanced-datasets-8576e9e366af
- https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18
- https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python
- https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reduction-techniques-python/
- https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/

- https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
- http://onnx.ai/sklearn-onnx/api_summary.html#skl2onnx.update_registered_converter
- https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html
- https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html