In [195]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import tree
import graphviz
from sklearn_pandas import DataFrameMapper
from sklearn import decomposition
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn2pmml.decoration import ContinuousDomain
from sklearn2pmml.pipeline import PMMLPipeline
from sklearn2pmml import sklearn2pmml
from six import StringIO
from IPython.display import Image
from sklearn import tree
import sklearn.datasets as datasets
from sklearn.tree import export_graphviz



import pydotplus
import graphviz
import math



# Line to create Python enviroment
# conda create -n thing python=3.6 numpy=1.16.2 pandas=0.24.2 scipy=1.2.1 onn
from sklearn.datasets import load_iris
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from skl2onnx.common.data_types import FloatTensorType
from skl2onnx import convert_sklearn
import onnxruntime as rt
from onnx.tools.net_drawer import GetPydotGraph, GetOpNodeProducer

import graphviz

from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
```

CS422- Project Report- Elamathi Senthilkumar

Abstract

The main findings from this project was that simple methods sometimes produced better results than anything complex. I also learned how to work with the more complex datasets. And also, I learnt that the model doesnot need to be complex to performs well to fit the model well. I also learned that efficient models which are not computationally expensive is as important as building the accurate models. While working on the large dataset makes me understand that selecting features and other data analysis is necessary for finding the optimal model. I also learnt how to proceed with the general dataset for building the optimal model.If this project were to be continued in the future, by providing the domain and then domain knowledge for this dataset variables would be helpful in feature selection.

Overview

Problem statement

The ultimate 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 this project is to reduce the dataset. This is because there are 12 lakhs of datas in the given dataset which makes the computation of the models slower. Next, will select the features manually to keep and use in the model. The data is then scaled and principal component analysis (PCA) is performed on it and then eliminate the features with more outliers inorder to bulid a good model.The final classifier uses a random forest with max_depth = 2 and n_estimators = 100.

Reading in the data

In [2]:
```
data = pd.read_csv('data_public.csv.gz',nrows= 100,error_bad_lines=False, com
print(data.head())
```

```
           A          B           C          D           E          F  \
0  231.420023 -12.210984  217.624839 -15.611916  140.047185  76.904999
1  -38.019270 -14.195695    9.583547  22.293822  -25.578283 -18.373955
2  -39.197085 -20.418850   21.023083  19.790280  -25.902587 -19.189004
3  221.630408  -5.785352  216.725322  -9.900781  126.795177  85.122288
4  228.558412 -12.447710  204.637218 -13.277704  138.930529  91.101870

           G          H          I          J           K         L  \
0  131.591871  198.160805  82.873279  127.350084  224.592926 -5.992983
1   -0.094457  -33.711852  -8.356041   23.792402    4.199023  2.809159
2   -2.953836  -25.299219  -6.612401   26.285392    5.911292  6.191587
3  108.857593  197.640135  82.560019  157.105143  212.989231 -3.621070
4  115.598954  209.300011  89.961688  130.299732  201.795100 -1.573922

          M          N          O  Class
0 -14.689648  143.072058  153.439659      2
1 -59.330681  -11.685950    1.317104      3
2 -56.924996   -4.675187   -1.027830      2
3 -15.469156  135.265859  149.212489      2
4 -15.128603  148.368622  147.492663      3
```

Here, I took only 100 rows of data from the given dataset. This is because the given dataset consists of 1200000 datas which increases the computational time.

In [90]:
```
labels = ['A', 'B','C','D','E','F','G','H','I','J','K','L','M','N','O']
# X = pd.DataFrame(data=data.drop('Class', axis=1), columns=labels)
# y = pd.DataFrame(data=data['Class'],columns=['Class'])
X_train, X_test, y_train, y_test = train_test_split(data.drop(labels=['Class'
                                                    data['Class'],
                                                    test_size=0.3,
                                                   random_state = 0)
training_data = pd.concat([X_train,y_train],axis=1)
test_data = pd.concat([X_test, y_test],axis = 1)
# print(test_data.head())
print(training_data.head())
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
             A          B           C          D           E          F  \
60  239.304171  -8.058400  213.516898 -14.848938  130.549926  89.150737
80  -29.317217 -12.538028   22.548496  20.195156  -27.048223 -28.950758
90  -32.779495 -14.515699    8.628796  15.405380  -18.302331 -24.451934
68  -40.551506 -13.141822   11.965320  20.643612  -22.614439 -32.476179
51  240.311718  -7.817104  222.516289 -11.663915  123.402682  92.698089

             G           H          I           J           K          L  \
60  116.322496  206.761109  78.783836  131.202371  214.020206 -10.375638
80   -1.217660  -26.013388  -4.165968   22.452930    3.593841   4.188496
90    1.254471  -22.656895  -4.654272   26.320910    2.372705   2.605553
68   -0.396090  -24.577213  -8.366304   24.953665    8.380710   7.043656
51  131.641115  193.290140  72.766889  126.875078  211.699336   3.039842

            M           N           O  Class
60 -24.253837  135.814220  150.660207      3
80 -55.205484   -9.623017   -6.404541      2
90 -55.464372   -6.113055    0.800140      3
68 -58.483872  -10.802592    0.137812      3
51 -11.040107  146.367650  147.064597      2
(70, 15)
(70,)
(30, 15)
(30,)
```

Data Processing and Analysis

In [3]:
```
print(data['Class'].value_counts())
```

```
2    51
3    38
1    11
Name: Class, dtype: int64
```

In [4]:
```
data['Class'].unique()
```

Out[4]:
```
array([2, 3, 1], dtype=int64)
```

In [89]:
```
data.shape
```

Out[89]:
```
(100, 16)
```

Check for missing values

Here, I checked for the missing values which might need to be imputed.

In [6]:
```
data.isna().sum()
```

Out[6]:
```
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
```

No missing data is found here.

In [7]:
```
data.columns
```

Out[7]:
```
Index(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N',
       'O', 'Class'],
      dtype='object')
```

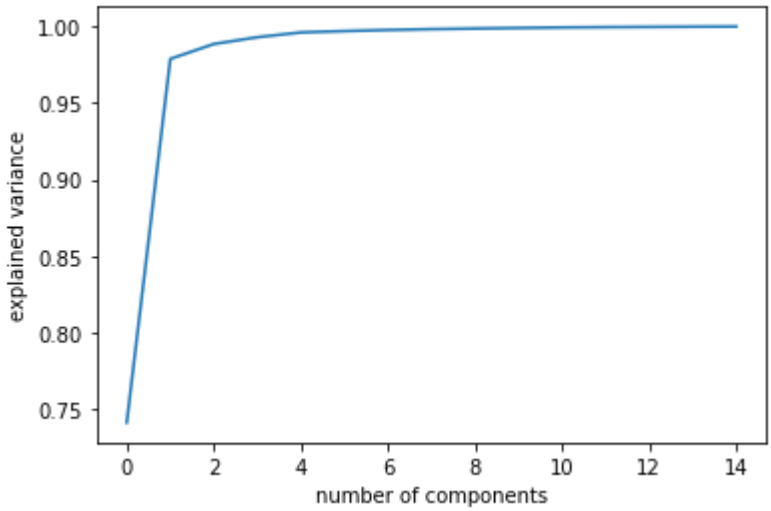
In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A       100 non-null    float64
 1   B       100 non-null    float64
 2   C       100 non-null    float64
 3   D       100 non-null    float64
 4   E       100 non-null    float64
 5   F       100 non-null    float64
 6   G       100 non-null    float64
 7   H       100 non-null    float64
 8   I       100 non-null    float64
 9   J       100 non-null    float64
 10  K       100 non-null    float64
 11  L       100 non-null    float64
 12  M       100 non-null    float64
 13  N       100 non-null    float64
 14  O       100 non-null    float64
 15  Class   100 non-null    int64  
dtypes: float64(15), int64(1)
memory usage: 12.6 KB
```

Principle Component Analysis

Next, I found the optimal number of principal components. For this I ran a PCA with number of components set to the total number of features and then generated a scree plot (the code to generate this plot was found at [https://districtdatalabs.silvrback.com/principal-component-analysis-with-python (https://districtdatalabs.silvrback.com/principal-component-analysis-with-python)](https://districtdatalabs.silvrback.com/principal-component-analysis-with-python) .

In [92]:
```
sc = StandardScaler()
df_scaled = sc.fit_transform(data.drop('Class', axis=1))
df_scaled = pd.DataFrame(df_scaled, columns=labels)
df_scaled = pd.concat([df_scaled, data['Class']], axis=1)
pca_test = decomposition.PCA(n_components=15)
pca_test.fit(df_scaled.drop('Class', axis=1))
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('explained variance')
plt.show()
pca_test.explained_variance_ratio_
```



Out[92]:
```
array([7.41287199e-01, 2.37577974e-01, 9.73279491e-03, 4.42601851e-03,
       3.13996540e-03, 8.67875405e-04, 7.21518397e-04, 5.50155134e-04,
       3.93352006e-04, 3.63461292e-04, 3.44300069e-04, 2.14475706e-04,
       1.45457288e-04, 1.35602649e-04, 9.98508070e-05])
```

Here, the elbow of the above scree plot occurs at n = 1. So, I took PCA(n_components=1) in all tested pipelines.

Feature Extraction

First, I tried executing executing this with the whole dataset. But the accuracy obtained for all the features are almost similar which is nearly 0.498. Also, the computational time is much higher. Therefore, here we are taking only few rows of datas for processing. By using the brute-force method, I found that its for loop was iterated through each of the 15 features. By removing each of these features and run the model to check which feature removal decreases the accuracy of the model. This decrease implies that the removed feature makes the model to be overfit. Here, for the initial pipeline I took a data scaler, a PCA with one component, and a decision tree with max_depth of 3. https://github.com/jpmml/sklearn2pmml/blob/master/sklearn2pmml/pipeline/__init__.py (https://github.com/jpmml/sklearn2pmml/blob/master/sklearn2pmml/pipeline/__init__.py)

In [99]:
```
from sklearn import metrics
features = 'ABCDEFGHIJKLMNO'
for i in range(0, len(features)):
    pipeline = PMMLPipeline([('mapper',DataFrameMapper([(X_train.columns.drop
                              ('pca',PCA(n_components=1)),('classifier',Decisi
    pipeline.fit(training_data.drop([features[i:i+1]], axis=1),training_data[
    results = pipeline.predict(X_test)
    actual = np.concatenate(y_test.values.reshape(30,1))
    print("Dropped feature:", features[i:i+1], ", Accuracy:", metrics.accurac
```

```
Dropped feature: A , Accuracy: 0.43333333333333335
Dropped feature: B , Accuracy: 0.4
Dropped feature: C , Accuracy: 0.4666666666666667
Dropped feature: D , Accuracy: 0.5333333333333333
Dropped feature: E , Accuracy: 0.6333333333333333
Dropped feature: F , Accuracy: 0.43333333333333335
Dropped feature: G , Accuracy: 0.6666666666666666
Dropped feature: H , Accuracy: 0.4666666666666667
Dropped feature: I , Accuracy: 0.6
Dropped feature: J , Accuracy: 0.4666666666666667
Dropped feature: K , Accuracy: 0.43333333333333335
Dropped feature: L , Accuracy: 0.5
Dropped feature: M , Accuracy: 0.5666666666666667
Dropped feature: N , Accuracy: 0.43333333333333335
Dropped feature: O , Accuracy: 0.43333333333333335
```

There is no need for us to keep the all correlated columns in our model. So, I build a correlation matrix, plots and histograms to visualize the features that are highly correlated with one another.

In [100]:
```
data.corr('pearson')
```

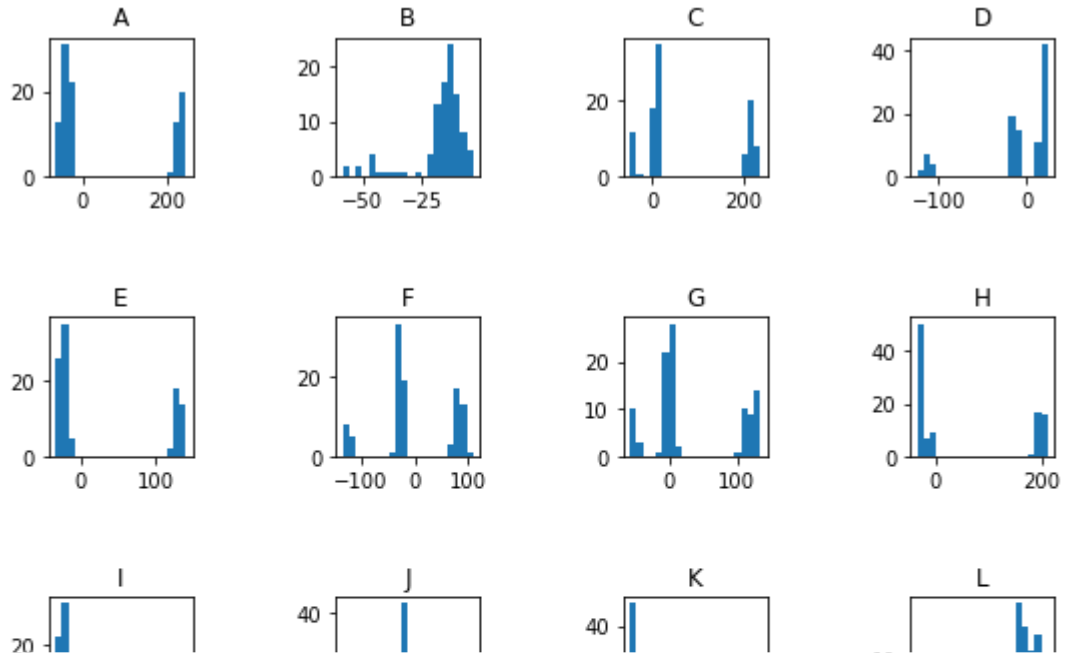
Out[100]:

	A	B	C	D	E	F	G	H	
A	1.000000	0.443324	0.992221	-0.009377	0.992565	0.908459	0.973585	0.990521	
B	0.443324	1.000000	0.524271	0.806912	0.358606	0.739405	0.595758	0.344047	-
C	0.992221	0.524271	1.000000	0.091479	0.976877	0.942882	0.988951	0.972685	
D	-0.009377	0.806912	0.091479	1.000000	-0.111103	0.396549	0.192673	-0.129988	-
E	0.992565	0.358606	0.976877	-0.111103	1.000000	0.862047	0.948286	0.997669	
F	0.908459	0.739405	0.942882	0.396549	0.862047	1.000000	0.967251	0.852897	
G	0.973585	0.595758	0.988951	0.192673	0.948286	0.967251	1.000000	0.941937	
H	0.990521	0.344047	0.972685	-0.129988	0.997669	0.852897	0.941937	1.000000	
I	0.847650	-0.042059	0.790781	-0.524268	0.895852	0.561716	0.719230	0.904135	
J	0.876538	0.777072	0.918455	0.459292	0.823763	0.987700	0.948368	0.814217	
K	0.975110	0.269864	0.949360	-0.212315	0.990479	0.803094	0.910336	0.993260	
L	0.063695	0.779927	0.157737	0.935343	-0.029966	0.440751	0.254101	-0.053717	-
M	0.963167	0.344323	0.947965	-0.103628	0.969020	0.839970	0.918592	0.970311	
N	0.959332	0.200390	0.926223	-0.276881	0.981918	0.764119	0.880805	0.984633	
O	0.935314	0.139517	0.895761	-0.351313	0.965898	0.711772	0.842241	0.969694	
Class	-0.069540	0.064982	-0.069030	0.053286	-0.071882	-0.029789	-0.071642	-0.073901	-

Type *Markdown* and LaTeX: α^2

In [96]:
```
fig1 = plt.figure()
for i in range(1,16):
 fig1.add_subplot(4,4,i)
 plt.hist(data[features[i-1:i]], bins=20)
 plt.title(features[i-1:i])

fig1.add_subplot(4,4,16)
plt.hist(data['Class'], bins=20)
plt.title('Class')
fig1.subplots_adjust(hspace=1, wspace=1)
fig1.set_figheight(9)
fig1.set_figwidth(9)
```



From the above histogram distributions, I determined that there might be some close relationship among the following features.

- A,C,E,K,H
- B,L
- D
- J,N,O
- F
- I,G
- M

Next, I tested these features by taking one feature from the groups to test the accuracy.From here I understood that removing the redundant feature was a good methodology which improves the overall accuracy.

In [107]:
```
to_keep1 = ['A','B','D','I','J','F','M']
pipeline1 = PMMLPipeline([
 ('mapper',
 DataFrameMapper([
 (X_train[to_keep2].columns,
 [StandardScaler()])])),
 ('pca',
 PCA(n_components=1)),
 ('classifier',
 RandomForestClassifier(max_depth=2,n_estimators=10))
])
pipeline1.fit(training_data,#.drop('Class',axis=1),
 training_data['Class'])
results = pipeline1.predict(X_test)
actual = np.concatenate(y_test.values.reshape(30,1))
print('Accuracy:',metrics.accuracy_score(actual, results))
```

```
Accuracy: 0.6
```

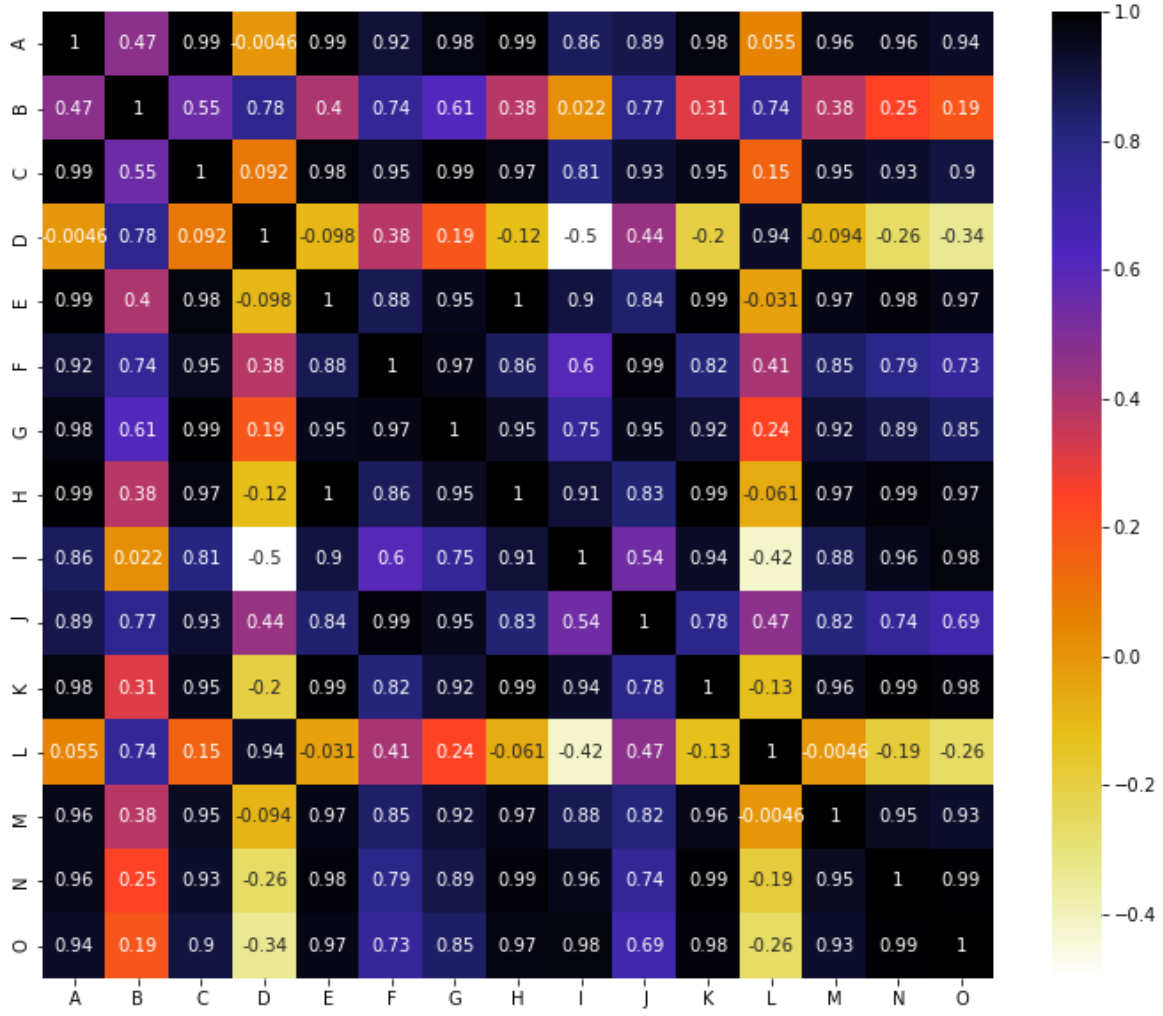
Correlation Plot

Now, let us check further by taking the features that have correlation magnitude greater than 0.95.

https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e (https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e)

In [97]:

```
import seaborn as sns
plt.figure(figsize=(12,10))
corr = X_train.corr()
sns.heatmap(corr, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```



With the help of this correlation function, we can identify the features which have high correlation magnitude. I refered this code from the given link.

https://gist.github.com/rachidelfermi/655e781b60e83b56c9560d9c872852ca (https://gist.github.com/rachidelfermi/655e781b60e83b56c9560d9c872852ca)

In [14]:
```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other fea

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested i
                colname = corr_matrix.columns[i]  # getting the name of colum
                col_corr.add(colname)
    return col_corr
```

In [15]:
```
corr_features = correlation(X_train, 0.95)
len(set(corr_features))
```

Out[15]: 9

In [16]:
```
corr_features
```

Out[16]: {'C', 'E', 'G', 'H', 'J', 'K', 'M', 'N', 'O'}

Now, I dropped these features from the dataset.

In [17]:
```
X_train.drop(corr_features,axis=1)
X_test.drop(corr_features,axis=1)
```

Out[17]:

	A	B	D	F	I	L
26	-31.831440	-16.037890	16.041759	-20.897764	-5.793556	3.788012
86	-31.594503	-14.607033	10.240161	-25.031143	-9.194471	-0.904170
2	-39.197085	-20.418850	19.790280	-19.189004	-6.612401	6.191587
55	224.304563	-10.047709	-12.310613	96.210583	85.212948	0.965348
75	-37.899442	-17.262739	21.526282	-30.937837	-6.392880	2.256472
93	-41.612929	-14.034674	20.445236	-28.321117	-4.724764	-3.535887
16	-30.023792	-8.107719	19.765770	-20.265524	-7.577699	2.464305
73	-31.214595	-16.850719	22.812030	-28.439917	-10.113632	1.876784
54	-36.118938	-14.321411	16.897426	-33.050007	-6.862152	5.856772
95	-30.799124	-8.534127	22.064023	-25.767853	-7.092668	-0.870427
53	246.407178	-9.634963	-15.829488	91.373245	90.988458	2.987500
92	241.526176	-16.947143	-13.764490	75.836688	73.583625	-6.770924
78	-36.098476	-11.680902	17.143544	-30.012980	-2.546599	2.595115
13	-32.610177	-9.981722	13.533336	-25.211891	-9.242013	5.004292
7	-28.620633	-16.324678	19.866385	-22.328572	-8.616671	4.953251
30	-33.426572	-17.053078	16.076831	-28.255024	-11.067195	0.868626
22	-63.798054	-47.508030	-108.156312	-121.994302	48.467472	-38.325831
24	214.501120	-12.386835	-11.482754	84.174828	79.514846	-9.476301
33	-61.020666	-47.286879	-114.643763	-133.597008	69.351557	-44.272380
8	-41.092898	-11.525839	18.670988	-25.918632	-13.371210	-1.041023
43	224.489072	-7.808840	-14.547985	79.584513	94.125792	-11.514014
62	-54.449029	-44.891024	-123.941370	-122.517815	54.352370	-27.607384
3	221.630408	-5.785352	-9.900781	85.122288	82.560019	-3.621070
71	-42.360407	-17.259601	18.507104	-20.698841	-4.648664	8.410465
45	-39.681024	-13.209121	17.586571	-26.202514	-9.300830	7.720388
48	222.190685	-15.306152	-12.990590	91.225028	88.132967	6.068442
6	-35.819795	-16.688245	17.570011	-20.625764	-10.699215	-2.365574
99	-32.366423	-15.652219	22.510080	-32.167443	-9.511788	4.172722
82	-36.698735	-13.690396	16.953206	-26.061282	-7.352873	7.401070
76	-60.974865	-58.692843	-115.455430	-132.341544	53.799941	-43.696538

In [117]:
```
print(X_train.drop(corr_features,axis=1).shape)
print(y_train.shape)
```

```
(70, 6)
(70,)
```

Feature Removals and models

After removing the highly correlated features from the features from the model, the accuracy for RandomForestClassifier reduces further which shows that the model becomes more generalized.

In [57]:
```
pipe_3 = make_pipeline(StandardScaler(),
                       PCA(n_components=1),
                       RandomForestClassifier())

pipe_3.fit(X_train.drop(corr_features,axis=1), training_data['Class'])
y_pred = pipe_3.predict(X_test.drop(corr_features,axis=1))
print('Test Accuracy: %.3f' % pipe_3.score(X_test.drop(corr_features,axis=1),
```

```
Test Accuracy: 0.400
```

In [58]:
```
pipe_1 = make_pipeline(StandardScaler(),
                       PCA(n_components=1),
                       DecisionTreeClassifier(max_depth = 3))

pipe_1.fit(X_train.drop(corr_features,axis=1), training_data['Class'])
y_pred = pipe_1.predict(X_test.drop(corr_features,axis=1))
print('Test Accuracy: %.3f' % pipe_1.score(X_test.drop(corr_features,axis=1),
```

```
Test Accuracy: 0.567
```

In [21]:
```
pipe_2 = make_pipeline(StandardScaler(),
                       PCA(n_components=1),
                       LogisticRegression(random_state=0))

pipe_2.fit(X_train.drop(corr_features,axis=1), training_data['Class'])
y_pred = pipe_2.predict(X_test.drop(corr_features,axis=1))
print('Test Accuracy: %.3f' % pipe_2.score(X_test.drop(corr_features,axis=1),
```

```
Test Accuracy: 0.600
```

In [49]:
```
## LEts make the list of pipelines
pipelines = [pipe_1,pipe_2,pipe_3]
print(pipelines)
```

```
[Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('pca', PCA(n_components=1)),
                 ('decisiontreeclassifier',
                  DecisionTreeClassifier(max_depth=3))]), Pipeline(steps=
[('standardscaler', StandardScaler()),
                 ('pca', PCA(n_components=1)),
                 ('logisticregression', LogisticRegression(random_state=
0))]), Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('pca', PCA(n_components=1)),
                 ('randomforestclassifier', RandomForestClassifier())])]
```

With the available features after removal, I compared the accuracy of these features in different models such as RandomForestClassifier, Decision tree and LogisticRegression. I obtained 0.4,0.567 and 0.6 accuracies for RandomForestClassifier, Decision tree and LogisticRegression respectively. Thus, LogisticRegression performs well in this case.

Training and cross-validating the final model

The final pipeline is represented as follows.

In [180]:
```
final_pipe = make_pipeline(StandardScaler(),
                           PCA(n_components=1),
                           RandomForestClassifier(random_state=0))

final_pipe.fit(X_train.drop(corr_features,axis=1), training_data['Class'])
y_pred = final_pipe.predict(X_test)
```

In [69]:
```
final_pipeline=[final_pipe]
print(final_pipeline)
```

```
[Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('pca', PCA(n_components=1)),
                 ('randomforestclassifier',
                  RandomForestClassifier(random_state=0))])]
```

In [187]:
```
pipe_2==[pipe_2]
print(pipe_2)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('pca', PCA(n_components=1)),
                 ('logisticregression', LogisticRegression(random_state=
0))])
```

In [181]:
```
cross_validate(final_pipe, data.drop('Class',axis=1), data['Class'], cv=6)
```

Out[181]:
```
{'fit_time': array([0.12365794, 0.10959029, 0.11055279, 0.09471488, 0.09373
903,
        0.09373975]),
 'score_time': array([0.        , 0.        , 0.        , 0.01562405, 0.015
62357,
        0.01562357]),
 'test_score': array([0.41176471, 0.47058824, 0.47058824, 0.35294118, 0.312
5    ,
        0.375     ])}
```

In [184]:
```
cross_validate(pipe_2, data.drop('Class',axis=1), data['Class'], cv=6)
```

Out[184]:
```
{'fit_time': array([0.00995445, 0.00699854, 0.01099992, 0.00899935, 0.00827
36 ,
        0.00699973]),
 'score_time': array([0.00200033, 0.00207901, 0.00399923, 0.00200009, 0.002
00009,
        0.00099993]),
 'test_score': array([0.52941176, 0.52941176, 0.52941176, 0.47058824, 0.5
,
        0.5       ])}
```

To evaluate the model, I performed K-fold cross validation on it one last time. The accuracy held around .4 in almost all trials.

After this, the pipeline was trained on the entire dataset for deployment.

In [169]:
```
final_pipe.fit(data_train, data_train['Class'])
```

Out[169]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('pca', PCA(n_components=1)),
                ('randomforestclassifier',
                 RandomForestClassifier(random_state=0))])
```

In [188]:
```
pipe_2.fit(data_train, data_train['Class'])
```

Out[188]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('pca', PCA(n_components=1)),
                ('logisticregression', LogisticRegression(random_state=
0))])
```

Deploying the onnx model

The final step is deploying the model. I refered this in the Individual Project - Pipeline - ONNX - Example given in the Blackboard.

In [192]:
```
# input_types = dict([(x, FloatTensorType([1, 1])) for x in labels])
to_keep = ['A','B','D','I','J','F','M']
input_types = dict([(x, FloatTensorType([None,1])) for x in to_keep])
print(input_types)
try:
    model_onnx = convert_sklearn(final_pipe,
                                 'pipeline_final_onnx',
                                 initial_types=list(input_types.items()))
except Exception as e:
    print(e)

with open("final_pipeline_Elamathi.onnx", "wb") as f:
    f.write(model_onnx.SerializeToString())
```

```
{'A': FloatTensorType(shape=[None, 1]), 'B': FloatTensorType(shape=[None,
1]), 'D': FloatTensorType(shape=[None, 1]), 'I': FloatTensorType(shape=[Non
e, 1]), 'J': FloatTensorType(shape=[None, 1]), 'F': FloatTensorType(shape=
[None, 1]), 'M': FloatTensorType(shape=[None, 1])}
```

In [193]:
```
input_types = dict([(x, FloatTensorType([1,15])) for x in X_train.columns.val
model_onnx = convert_sklearn(final_pipe, initial_types=list(input_types.items
with open("onnx_test_Elamathi.onnx", "wb") as f:
 f.write(model_onnx.SerializeToString())
```

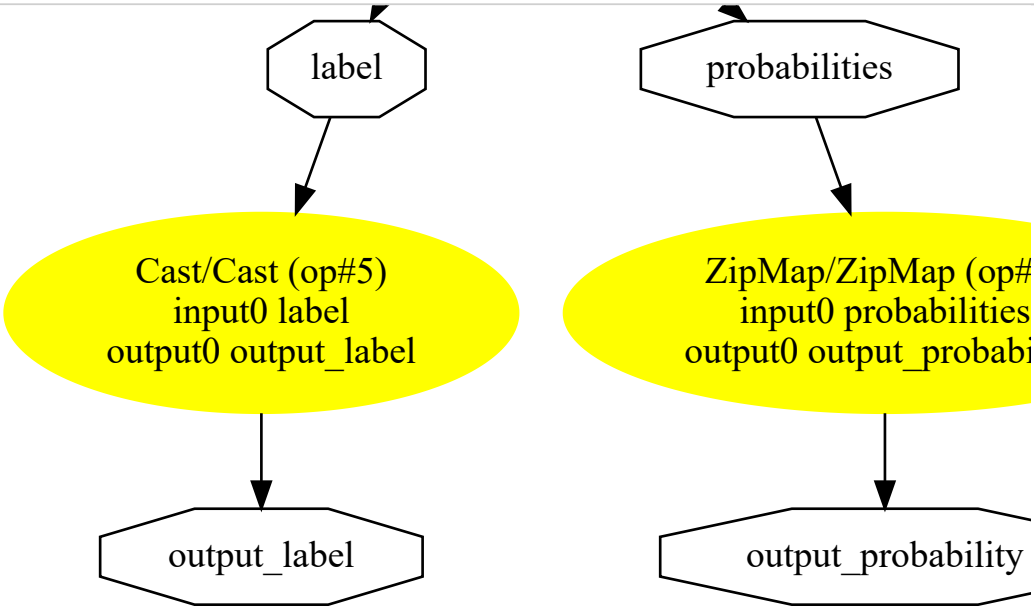
In [171]:
```
inputs_onnx = {k: np.array(v).astype(np.float32)[:, np.newaxis] for k, v in X

session_onnx = rt.InferenceSession("iris_pipeline.onnx")
predict_onnx = session_onnx.run(None, inputs_onnx)
print("\n","predict", predict_onnx[0])
```

```
predict [2 2 2 ... 2 2 2]
```

In [194]:
```
pydot_graph = GetPydotGraph(model_onnx.graph,
                            name=model_onnx.graph.name,
                            rankdir="TB",
                            node_producer=GetOpNodeProducer("docstring",
                                                            color="yellow",
                                                            fillcolor="yellow
                                                            style="filled"))

graphviz.Source(pydot_graph)
```



Conclusion

Finally, I compared two models and understood that even the simple logistic regression method provides the overall accuracy of 0.6. But, while doing the cross validation, I got the accuracy range of 0.4 for all the features for RandomForestClassification, which shows that this model mostlikely does not overfit or hyperfit the data. I also believes that more testing and better feature selection will makes this model provides more optimal solution. A **positive** result is that my model seems to peform well when cross validated, but a **caveat** is that while doing all these steps, I discarded majority of the information and the data. This means that some crucial piece of information or contributed variance might be lost in one of the discarded features.

References

The codes are refered from the links given above the codes. The other links refered are shown as follows.

http://restanalytics.com/2020-12-07-Using-Scikit-Learn-Pipelines-and-Converting-Them-To-PMML/ (http://restanalytics.com/2020-12-07-Using-Scikit-Learn-Pipelines-and-Converting-Them-To-PMML/)

https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e (https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e)

https://github.com/jpmml/sklearn2pmml/tree/master/sklearn2pmml/pipeline (https://github.com/jpmml/sklearn2pmml/tree/master/sklearn2pmml/pipeline)

https://towardsdatascience.com/is-random-forest-better-than-logistic-regression-a-comparison-7a0f068963e4 (https://towardsdatascience.com/is-random-forest-better-than-logistic-regression-a-comparison-7a0f068963e4)

https://queirozf.com/entries/scikit-learn-pipeline-examples (https://queirozf.com/entries/scikit-learn-pipeline-examples)