In [188]:
```
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 sklearn.externals.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 onnx
=1.4.1 onnxruntime=0.3.0 skl2onnx=1.4.5

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

from sklearn import metrics
from sklearn import preprocessing

from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report

from feature_selector import FeatureSelector
from sklearn.ensemble import BaggingClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from mlxtend.feature_selection import ColumnSelector


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


from onnxmltools.convert.common.shape_calculator import calculate_linear_class
ifier_output_shapes
from skl2onnx.operator_converters.RandomForest import convert_sklearn_random_f
orest_classifier
```

CS 422 Project Report - Victoria Belotti

Abstract

One of my main findings from this project was that simple methods sometimes produced better results than anything complex. Specifically, I learned that a model does not necessarily have to be complex to perform well and that it is extremely easy to overfit a model. I also learned how to research, read, and understand the documentation of Python tools/packages, along with installing packages in some cases. Another research finding was understanding the importance of making not only accurate models, but also efficient models that are not too computationally expensive. Working on a large dataset for the first time revealed that brute force methods should be avoided.

If this project were to be continued in the future, other tests such as dividing the data into groups and training different pipelines on different groups could be used for comparison. Another possible future use is if this data was given a domain and then domain knowledge could also be applied to feature selection.

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 a package called FeatureSelector 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 random forest with max_depth = 2 and n_estimators = 100.

Reading in the data

In [3]:
```
df = pd.read_csv('data_public.csv.gz', compression='gzip', header=0, sep=',',
quotechar='"')
df.head()
```

Out[3]:

	A	B	C	D	E	F	G	
0	231.420023	-12.210984	217.624839	-15.611916	140.047185	76.904999	131.591871	198.16080
1	-38.019270	-14.195695	9.583547	22.293822	-25.578283	-18.373955	-0.094457	-33.71185
2	-39.197085	-20.418850	21.023083	19.790280	-25.902587	-19.189004	-2.953836	-25.29921
3	221.630408	-5.785352	216.725322	-9.900781	126.795177	85.122288	108.857593	197.64013
4	228.558412	-12.447710	204.637218	-13.277704	138.930529	91.101870	115.598954	209.30001

In [4]:
```
labels = ['A', 'B','C','D','E','F','G','H','I','J','K','L','M','N','O']
X = pd.DataFrame(data=df.drop('Class', axis=1),
                 columns=labels)
y = pd.DataFrame(data=df['Class'],
                 columns=['Class'])

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3)

training_data = pd.concat([X_train,
                       y_train],
                      axis=1)
training_data.head()
```

Out[4]:

	A	B	C	D	E	F	G	
956376	-33.514246	-12.986159	9.732139	19.439789	-31.232571	-27.490557	-6.361316	-28.
492858	252.052183	-12.026161	204.975766	-11.436195	128.708950	73.493618	117.484681	195
35547	226.876313	-12.545289	214.987767	-15.097891	132.724278	68.518947	102.471853	202.
125579	236.777686	-6.730203	223.721602	-13.934494	135.644324	82.508771	121.801578	202.
897948	-37.521529	-17.582101	17.983932	15.780881	-21.251064	-24.167114	0.169100	-29.

Data Processing and Analysis

Checking for Missing Values

First, I checked the data for missing values that might need to be imputed. I did this using the package FeatureSelector (https://github.com/WillKoehrsen/feature-selector (https://github.com/WillKoehrsen/feature-selector)).

In [79]:
```
fs_missing_test = FeatureSelector(data=training_data) #This tests all features and the class labels
fs_missing_test.identify_missing(missing_threshold=0)
```

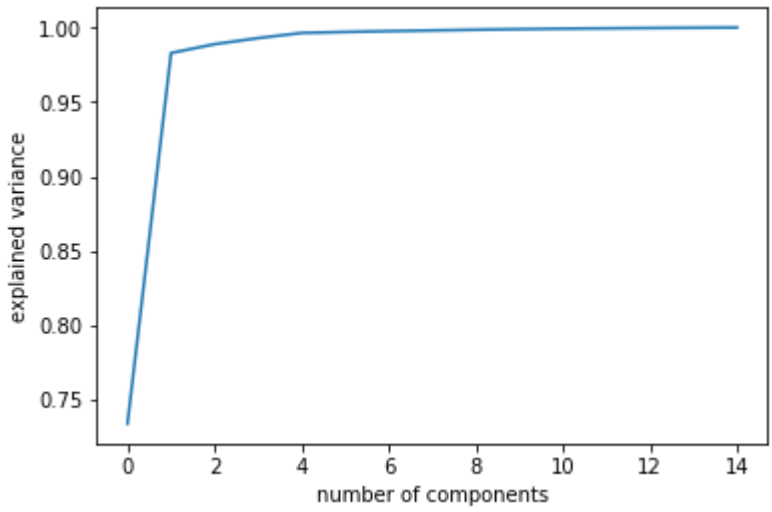
```
No labels provided. Feature importance based methods are not available.
0 features with greater than 0.00 missing values.
```

No missing data was found.

Principal Component Analysis

The next piece of information I wanted to discover was the optimal number of principal components to use throughout this project. To do this, I ran a PCA with n_components set to the total number of features and 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)).

In [206]:
```
sc = StandardScaler()
df_scaled = sc.fit_transform(df.drop('Class', axis=1))
df_scaled = pd.DataFrame(df_scaled, columns=labels)
df_scaled = pd.concat([df_scaled, df['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[206]:
```
array([7.33864677e-01, 2.49030223e-01, 5.94772600e-03, 4.07704155e-03,
       3.46584323e-03, 6.57804933e-04, 5.61838589e-04, 4.71665772e-04,
       4.61722639e-04, 3.82898239e-04, 3.11013079e-04, 2.56592626e-04,
       2.01142386e-04, 1.84748847e-04, 1.25062088e-04])
```

The elbow in the above scree plot occurs at n = 1, so I used 1 principal component in all tested pipelines.

Feature Removal

The next step of my method is to decide which features to remove. The first method I tried was brute force: writing a for-loop that iterated through each of the 15 features, removing each one, and running the pipeline to see which feature removals decreased the accuracy of the model. I hypothesized that a decrease in accuracy implies that the removed feature is one of the problematic ones that causes the model to be overfit. The initial pipeline and parameters used for this purpose were a slightly modified version of the one given in the example notebook. It consists of a data scaler, a PCA with one component, and a decision tree with max_depth of 3.

In [21]:
```
features = 'ABCDEFGHIJKLMNO'

for i in range(0, len(features)):

    pipeline = PMMLPipeline([
    ('mapper',
     DataFrameMapper([
         (X_train.columns.drop([features[i:i+1]]).values,
          [StandardScaler()])])),
    ('pca',
     PCA(n_components=1)),
    ('classifier',
     DecisionTreeClassifier(max_depth = 3))
    ])

    pipeline.fit(training_data.drop([features[i:i+1]], axis=1),
             training_data['Class'])

    results = pipeline.predict(X_test)
    actual = np.concatenate(y_test.values)
    print("Dropped feature:", features[i:i+1], ", Accuracy:", metrics.accuracy
_score(actual, results))
```

```
Dropped feature: A , Accuracy: 0.9997333333333334
Dropped feature: B , Accuracy: 0.9664055555555555
Dropped feature: C , Accuracy: 0.9978638888888889
Dropped feature: D , Accuracy: 0.9996527777777777
Dropped feature: E , Accuracy: 0.9999888888888889
Dropped feature: F , Accuracy: 0.9271888888888888
Dropped feature: G , Accuracy: 0.9885694444444444
Dropped feature: H , Accuracy: 0.9999944444444444
Dropped feature: I , Accuracy: 1.0
Dropped feature: J , Accuracy: 0.9020638888888889
Dropped feature: K , Accuracy: 1.0
Dropped feature: L , Accuracy: 0.9988416666666666
Dropped feature: M , Accuracy: 1.0
Dropped feature: N , Accuracy: 1.0
Dropped feature: O , Accuracy: 1.0
```

This brute force method showed that when Features F and J were removed, the model saw a greater decrease (from around 1.0 to around .90-.92) in accuracy than the other features, which initially led me to believe that F and J were the variables that caused the model to overfit. I speculated that removing them would fix the overfitting problem; however, that was not the case. Removing F and J still returned accuracy near 1.0, as seen below:

In [54]:
```
to_drop = ['F','J']


pipeline0 = PMMLPipeline([
    ('mapper',
     DataFrameMapper([
         (X_train.columns.drop(to_drop).values,
          [StandardScaler()])])),
    ('pca',
     PCA(n_components=1)),
    ('classifier',
     DecisionTreeClassifier(max_depth = 2))
    ])

pipeline0.fit(training_data.drop(to_drop, axis=1),
         training_data['Class'])

results = pipeline0.predict(X_test.drop(to_drop, axis=1))
actual = np.concatenate(y_test.values)
print('Accuracy:', metrics.accuracy_score(actual, results))
```

```
Accuracy: 0.9957555555555555
```

The accuracy when both F and J were dropped did in fact decrease the overall model accuracy as expected, but not nearly to the degree as it did when either F or J was dropped individually. This caused me to wonder about the relationship between F and J. To investigate this, I ran a correlation metric on all of the features.

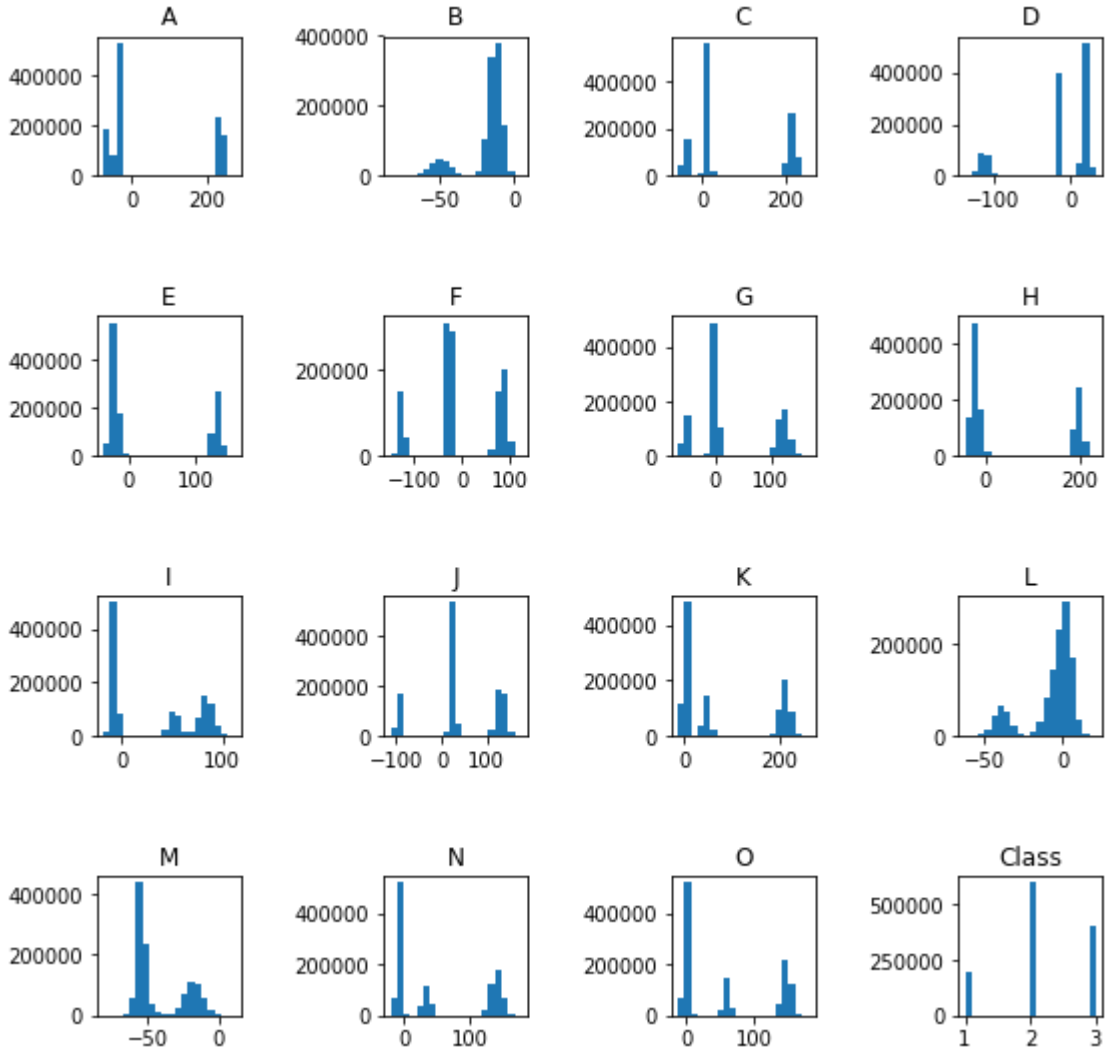
In [15]:
```
df.corr('pearson')
```

Out[15]:

	A	B	C	D	E	F	G	H	
A	1.000000	0.455949	0.991999	0.071330	0.990703	0.905353	0.972223	0.988807	0.818
B	0.455949	1.000000	0.541742	0.865856	0.352946	0.760708	0.620607	0.339549	-0.098
C	0.991999	0.541742	1.000000	0.176224	0.971805	0.943482	0.988351	0.968342	0.753
D	0.071330	0.865856	0.176224	1.000000	-0.047459	0.477183	0.279248	-0.062451	-0.502
E	0.990703	0.352946	0.971805	-0.047459	1.000000	0.849129	0.939705	0.997116	0.879
F	0.905353	0.760708	0.943482	0.477183	0.849129	1.000000	0.969055	0.841227	0.508
G	0.972223	0.620607	0.988351	0.279248	0.939705	0.969055	1.000000	0.934714	0.678
H	0.988807	0.339549	0.968342	-0.062451	0.997116	0.841227	0.934714	1.000000	0.886
I	0.818399	-0.098558	0.753474	-0.502643	0.879142	0.508345	0.678043	0.886017	1.000
J	0.870016	0.803246	0.915784	0.544357	0.805749	0.989868	0.949429	0.796856	0.439
K	0.968827	0.246429	0.937868	-0.163679	0.989217	0.781534	0.894114	0.990875	0.926
L	0.139619	0.854635	0.238723	0.949485	0.026319	0.518117	0.335039	0.012005	-0.418
M	0.958931	0.345030	0.941040	-0.042057	0.964769	0.823551	0.910385	0.964627	0.848
N	0.953081	0.194578	0.916578	-0.217856	0.979925	0.745156	0.867546	0.982403	0.943
O	0.920322	0.098805	0.873800	-0.316241	0.958885	0.675416	0.815281	0.962873	0.970
Class	0.891631	0.785198	0.933739	0.512742	0.831156	0.995675	0.963443	0.822728	0.476

Displaying this correlation showed that F and J were the most highly correlated with the class labels themselves, with close to 100% correlation. This led me to explore the correlation and similarity between the other features. The next step I tried in the pursuit of feature extraction was plotting each feature as a histogram to visually inspect and compare similarities.

In [28]:
```
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)
```



Using visual inspection (looking at which histogram distrubtions looked similar), I determined there to be close relationships among the following features:

- F, J (the problem ones)
- C, G
- A, E, H
- B, L
- L
- D
- I
- K, N
- M
- O

From this, I then tested a model based on choosing one of each from the groups I found. As with F and J, I found that if I removed an entire group of similarly-distrubuted features, the model would go back to being overfit. The following example uses the first feature on each line above, dropping the others. Additionally, at this point in the project I switched to using random forests in my pipeline, however, at this stage they are untuned.

In [69]:
```
to_keep1 = ['F','C','A','B','L','D','I','K','M','O']

pipeline1 = PMMLPipeline([
    ('mapper',
     DataFrameMapper([
         (X_train[to_keep1].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)
print('Accuracy:',metrics.accuracy_score(actual, results))
```

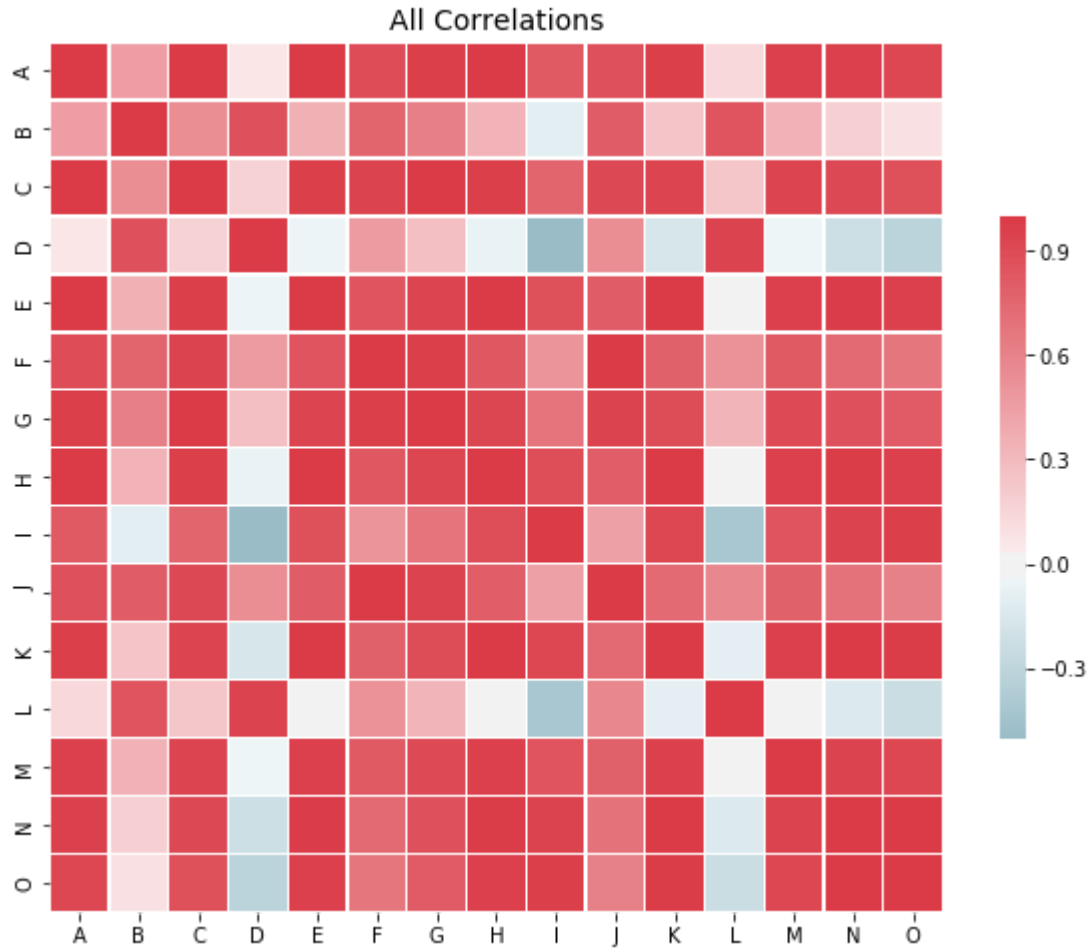
```
Accuracy: 0.96775
```

This produced the lowest accuracy value I had seen so far (i.e. not 1.0 or extremely close to 1.0) from removing multiple features . From here, I realized that removing redundant features was a good methodology. Using the FeatureSelector package, I systemtically determined the collinearity of each feature, based on its correlation with all the other features.

In [208]:
```
fs = FeatureSelector(data = training_data.drop('Class',axis=1), labels =traini
ng_data['Class'] )
fs.identify_collinear(correlation_threshold = 0.98)
```

```
8 features with a correlation magnitude greater than 0.98.
```

In [77]:
```
fs.plot_collinear( plot_all = True)
```



In [78]:
```
collinear_features = fs.ops['collinear']
fs.record_collinear
```

Out[78]:

	drop_feature	corr_feature	corr_value
0	C	A	0.992005
1	E	A	0.990709
2	G	C	0.988350
3	H	A	0.988820
4	H	E	0.997119
5	J	F	0.989854
6	K	E	0.989245
7	K	H	0.990891
8	N	H	0.982414
9	N	K	0.992167
10	O	K	0.983012
11	O	N	0.988941

These collinearity metric confirmed (F and J are highly correlated to each other), added to (one example is A being highly correlated with E and C), and corrected (B and L weren't that correlated after all) my earlier visual inspection comparisons drawn between the features. These metrics indicated that the following groupings of similarity exist among the features:

- F, J (and the class labels)
- C, G
- A, E, H, K, N, O
- B
- L
- D
- I
- M

Removing the hyper-correlated features F and J, and then removing the features that FeatureSelector said were also highly correlated with one another, yielded the following results:

In [90]:
```
to_keep2 = ['A','B','D','I','L','M']


pipeline2 = PMMLPipeline([
    ('mapper',
     DataFrameMapper([
         (X_train[to_keep2].columns,
          [StandardScaler()])])),
    ('pca',
     PCA(n_components=1)),
    ('classifier',
     RandomForestClassifier(max_depth=2,n_estimators=10))
])

pipeline2.fit(training_data,#.drop('Class',axis=1),
              training_data['Class'])
results = pipeline2.predict(X_test)
actual = np.concatenate(y_test.values)
print('Accuracy:',metrics.accuracy_score(actual, results))
```

```
Accuracy: 0.9323694444444445
```

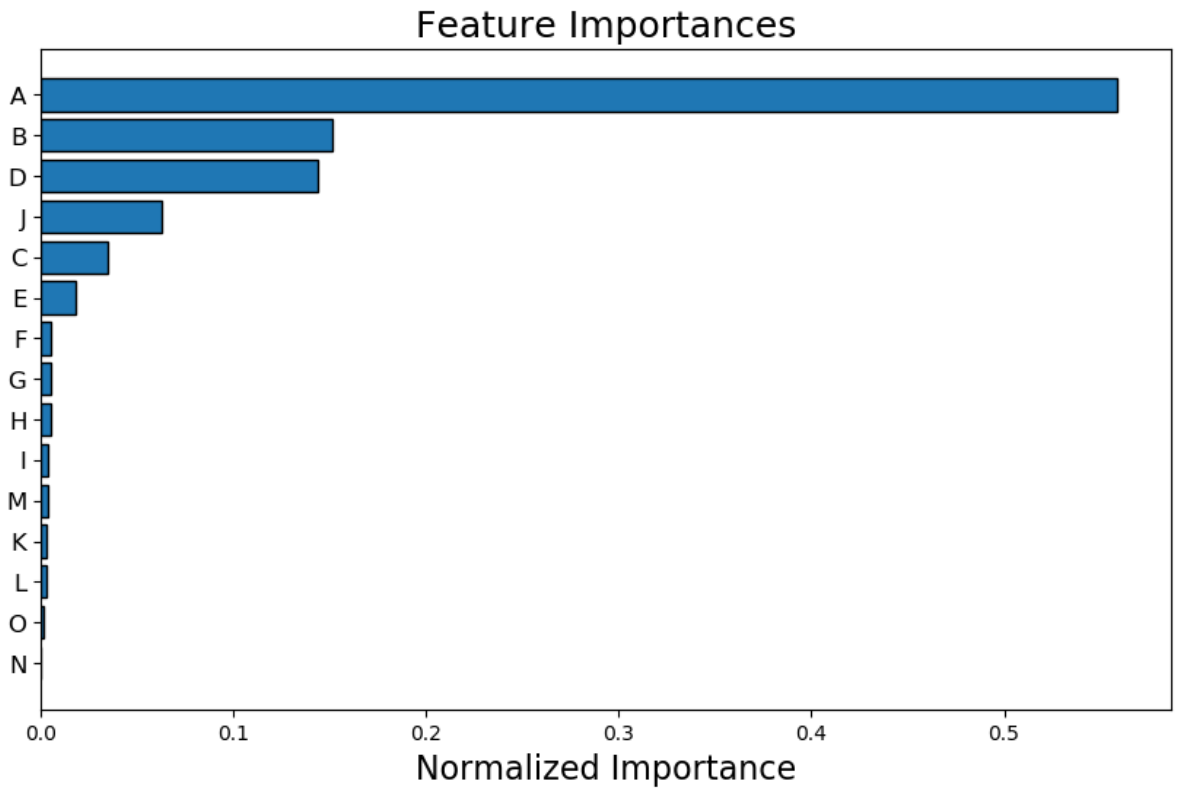
As can be seen, the accuracy has been further reduced to around .93 (down from close to 1.0), which means removing the highly correlated features in theory makes the model more generalizable. The next feature reduction method I used was also from the FeatureSelector package: Zero Importance Features. This package uses LightGBM and repeated tests to determine and rank the contributed important of each feature. I set eval_metric to 'multi_error' because it was one of the few multiclass metrics availible and it was less computationally expensive than log_loss. It utilizes early stopping in an effort to reduce overfitting.

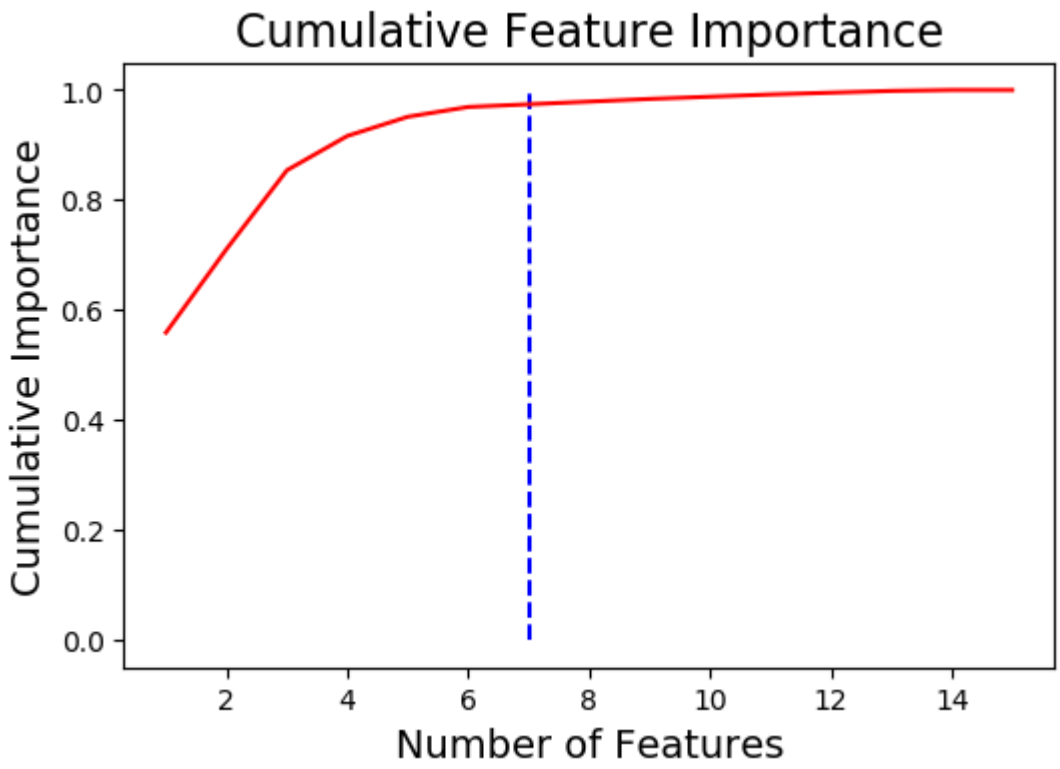
In [211]:
```
fs.identify_zero_importance(task = 'classification',
                            eval_metric = 'multi_error',
                            n_iterations = 10,
                             early_stopping = True)
```

```
Training Gradient Boosting Model

Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588442
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588132
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588887
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588934
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588482
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588408
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588424
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.58863
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.58932
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[8]     valid_0's multi_error: 0        valid_0's multi_logloss: 0.588816

1 features with zero importance after one-hot encoding.
```

Below, the features are plotted according to their importance, as measured by identify_zero_importance (the only feature determined to have zero importance was Feature O). It can be seen that, in a similar fashion to PCA, 97 percent of the importance can be captured within seven features.

In [215]:
```
zero_importance_features = fs.ops['zero_importance']
fs.plot_feature_importances(threshold = 0.97, plot_n = 15)
```






```
7 features required for 0.97 of cumulative importance
```

In spite of Feature J's high correlation with the class labels, it was determined to only be the fourth most important feature. That, in combination with the fact that I speculated J was one of the intentional problematic features, I decided to test combinations of features that were found to be more important than J (A, B, D). This was done by iteratively testing all possible subsets of A, B, and D.

In [136]:
```
to_test =  [['A'],['B'],['D'],['A','B'],['A','D'],['B','D'],['A','B','D']] #Al
l possible subsets of A, B, D

for to_keep3 in to_test:
    pipeline3 = PMMLPipeline([
        ('mapper',
         DataFrameMapper([
             (X_train[to_keep3].columns,
              [StandardScaler()])])),
        ('pca',
         PCA(n_components=1)),
        ('classifier',
         RandomForestClassifier(max_depth=2,n_estimators=10))
    ])

    pipeline3.fit(training_data,#.drop('Class',axis=1),
                  training_data['Class'])
    results = pipeline3.predict(X_test)
    actual = np.concatenate(y_test.values)
    print('Feature(s) tested:',to_keep3, 'Accuracy:', metrics.accuracy_score(a
ctual, results))
```

```
Feature(s) tested: ['A'] Accuracy: 0.9999444444444444
Feature(s) tested: ['B'] Accuracy: 0.7252972222222223
Feature(s) tested: ['D'] Accuracy: 1.0
Feature(s) tested: ['A', 'B'] Accuracy: 0.9999277777777777
Feature(s) tested: ['A', 'D'] Accuracy: 1.0
Feature(s) tested: ['B', 'D'] Accuracy: 0.8571083333333334
Feature(s) tested: ['A', 'B', 'D'] Accuracy: 0.9154055555555556
```

These features attained a high accuracy, but not an absolutely accuracy that might imply they are overfit. At this point in my research, I began using ColumnSelector for my pipelines (http://rasbt.github.io/mlxtend/user_guide/feature_selection/ColumnSelector/ (http://rasbt.github.io/mlxtend/user_guide/feature_selection/ColumnSelector/)). Using the combinations of features that did not overfit the model, [B], [B, D], and [A, B, D], I tested each of those subsets using 12-fold cross validation (found from https://chrisalbon.com/machine_learning/model_evaluation/cross_validation_pipeline/ (https://chrisalbon.com/machine_learning/model_evaluation/cross_validation_pipeline/)) to see if the results seen above held for the entire dataset, which they did.

In [226]:
```
cv_pipeline1 = PMMLPipeline([
        ('column_selector', ColumnSelector(cols=['B'])),
        ('scaler',StandardScaler()),
        ('pca',
         PCA(n_components=1)),
        ('classifier',
         RandomForestClassifier(max_depth=3,n_estimators=10))
    ])

cv_pipeline1.fit(training_data,#.drop('Class',axis=1),
                 training_data['Class'])
results = cv_pipeline1.predict(X_test)
actual = np.concatenate(y_test.values)
print(metrics.classification_report(actual, results))
print('Accuracy:',metrics.accuracy_score(actual, results))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     60208
           2       0.75      0.69      0.72    179971
           3       0.58      0.65      0.61    119821

   micro avg       0.73      0.73      0.73    360000
   macro avg       0.78      0.78      0.78    360000
weighted avg       0.73      0.73      0.73    360000

Accuracy: 0.7285222222222222
```

In [169]:
```
cross_validate(cv_pipeline1, df.drop('Class',axis=1), df['Class'], cv=12)
```

```
c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\sklearn\utils\deprecati
on.py:125: FutureWarning: You are accessing a training score ('train_score'),
which will not be available by default any more in 0.21. If you need training
scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[169]:
```
{'fit_time': array([6.77587509, 7.08604789, 7.40712309, 6.64372158, 6.5677216
1,
        6.80394077, 6.61427355, 7.28850555, 6.64721918, 7.00725627,
        6.68412209, 6.7120471 ]),
 'score_time': array([0.07081079, 0.07779002, 0.06248474, 0.06466341, 0.07813
239,
        0.07579589, 0.09075975, 0.08676815, 0.07679439, 0.07480025,
        0.06981301, 0.07579565]),
 'test_score': array([0.72805272, 0.72485275, 0.72608274, 0.72295   , 0.72301
,
        0.72614   , 0.72583   , 0.72455   , 0.7257    , 0.72508   ,
        0.72482725, 0.72543451]),
 'train_score': array([0.72623793, 0.72535975, 0.72803975, 0.72314273, 0.7232
5364,
        0.72659727, 0.72441818, 0.72327364, 0.72644909, 0.72538909,
        0.72599752, 0.72672686])}
```

In [223]:
```
cv_pipeline2 = PMMLPipeline([
        ('column_selector', ColumnSelector(cols=['B','D'])),
        ('scaler',StandardScaler()),
        ('pca',
         PCA(n_components=1)),
        ('classifier',
         RandomForestClassifier(max_depth=3,n_estimators=10))
    ])

cv_pipeline2.fit(training_data,
                 training_data['Class'])
results = cv_pipeline2.predict(X_test)
actual = np.concatenate(y_test.values)
print(metrics.classification_report(actual, results))
print('Accuracy:',metrics.accuracy_score(actual, results))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     60208
           2       0.86      0.85      0.86    179971
           3       0.78      0.80      0.79    119821

   micro avg       0.86      0.86      0.86    360000
   macro avg       0.88      0.88      0.88    360000
weighted avg       0.86      0.86      0.86    360000

Accuracy: 0.8571361111111111
```

In [170]:
```
cross_validate(cv_pipeline2, df.drop('Class',axis=1), df['Class'], cv=12)
```

```
c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\sklearn\utils\deprecati
on.py:125: FutureWarning: You are accessing a training score ('train_score'),
which will not be available by default any more in 0.21. If you need training
scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[170]:
```
{'fit_time': array([6.73897552, 6.71603703, 6.74894977, 6.76789808, 6.7848522
7,
        6.93544865, 6.79582238, 7.29049921, 7.17480946, 6.9553957 ,
        7.05014157, 6.68508792]),
 'score_time': array([0.0757966 , 0.07482958, 0.07679343, 0.08078313, 0.07779
145,
        0.07779074, 0.08577061, 0.07882094, 0.07380247, 0.08676887,
        0.07280588, 0.0767951 ]),
 'test_score': array([0.85599144, 0.85761142, 0.85815142, 0.85654   , 0.85537
,
        0.85672   , 0.85435   , 0.85695   , 0.85753   , 0.85572   ,
        0.85856859, 0.85710714]),
 'train_score': array([0.85621351, 0.85678987, 0.85663987, 0.85685364, 0.8571
4818,
        0.85679273, 0.85710273, 0.85680364, 0.85702909, 0.85695818,
        0.85655195, 0.85687571])}
```

In [233]:
```
cv_pipeline3 = PMMLPipeline([
        ('column_selector', ColumnSelector(cols=['A','B','D'])),
        ('scaler',StandardScaler()),
        ('pca',
         PCA(n_components=1)),
        ('classifier',
         RandomForestClassifier(max_depth=3,n_estimators=10))
    ])

cv_pipeline3.fit(training_data,
                 training_data['Class'])
results = cv_pipeline3.predict(X_test)
actual = np.concatenate(y_test.values)
print(metrics.classification_report(actual, results))
print('Accuracy:',metrics.accuracy_score(actual, results))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     60208
           2       0.91      0.92      0.92    179971
           3       0.88      0.87      0.87    119821

   micro avg       0.92      0.92      0.92    360000
   macro avg       0.93      0.93      0.93    360000
weighted avg       0.92      0.92      0.92    360000

Accuracy: 0.9154027777777778
```

In [173]:
```
cross_validate(cv_pipeline3, df.drop('Class',axis=1), df['Class'], cv=12)
```

```
c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\sklearn\utils\deprecati
on.py:125: FutureWarning: You are accessing a training score ('train_score'),
which will not be available by default any more in 0.21. If you need training
scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[173]:
```
{'fit_time': array([8.25890684, 7.88790417, 7.73132133, 7.45206666, 7.5704281
3,
        7.37686753, 7.96828222, 7.77320838, 7.54882002, 8.22503257,
        7.62414551, 7.36248112]),
 'score_time': array([0.07879066, 0.09873533, 0.07579684, 0.07583427, 0.07810
64 ,
        0.08055806, 0.07881761, 0.07779288, 0.07585073, 0.08506203,
        0.09355164, 0.06386113]),
 'test_score': array([0.91667083, 0.91645084, 0.91432086, 0.91538   , 0.91641
,
        0.9162    , 0.91632   , 0.91717   , 0.91576   , 0.91617   ,
        0.91578916, 0.91623832]),
 'train_score': array([0.91598447, 0.91608538, 0.91632174, 0.91613182, 0.9160
2727,
        0.91607364, 0.91596364, 0.91592909, 0.91616818, 0.91606364,
        0.91615644, 0.91612743])}
```

As can be seen, [B] held around an accuracy of .72, [B, D] around .85, and [A, B, D] around .91. I decided to choose to use all three of [A, B, D] because that set seemed to have the best performance without overfitting.

Random Forest Tuning

With the parameters selected, the final step of my model is tuning and regularizing the RandomForestClassifier. Throughout my above research, I had used max_dept = 3 and n_estimators = 10 because they were generic parmeters that were stable and were also not too computationally expensive. Here, I wanted to specifically find and tune the optimal RandomForest parameters. This was done using GridSearchCV (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

Scaling and PCA are performed before creating the random forests because that is how it will be in the final pipeline.

In [73]:
```
labels = ['A', 'B','C','D','E','F','G','H','I','J','K','L','M','N','O']
X_scaled = pd.DataFrame(data=df_scaled.drop('Class', axis=1),
                        columns=labels)
y_scaled = pd.DataFrame(data=df_scaled['Class'],
                        columns=['Class'])

X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_spli
t(X_scaled,
                                                    y_scaled,
                                                    test_size=0.3)

training_data_scaled = pd.concat([X_train_scaled,
                        y_train_scaled],
                       axis=1)

training_data_scaled.head()

pca = decomposition.PCA(n_components=1)
```

In [74]:
```
to_use = ['A','B','D']

prin_comps = pca2.fit_transform(training_data_scaled[to_use])
```

Here, GridSearchCV is used to test for depth.

In [66]:
```
random_classifier = RandomForestClassifier(n_estimators=10)
parameters = {'max_depth': [1,2,3,4]}
random_grid = GridSearchCV(random_classifier, parameters, cv = 4)

random_grid.fit(prin_comps, training_data_scaled['Class'])
```

Out[66]:
```
GridSearchCV(cv=4, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, cr
iterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [1, 2, 3, 4]}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring=None, verbose=0)
```

In [72]:
```
random_grid.cv_results_
```

Out[72]:
```
{'mean_fit_time': array([ 3.69464415,  6.09152806,  8.39974636, 10.3620792
6]),
 'std_fit_time': array([0.01630318, 0.02657662, 0.42725819, 0.26584812]),
 'mean_score_time': array([0.29041517, 0.31343567, 0.33756429, 0.36770993]),
 'std_score_time': array([0.00618637, 0.00477264, 0.00808566, 0.01877138]),
 'param_max_depth': masked_array(data=[1, 2, 3, 4],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1},
  {'max_depth': 2},
  {'max_depth': 3},
  {'max_depth': 4}],
 'split0_test_score': array([0.74887382, 0.91576747, 0.9156389 , 0.9157389
]),
 'split1_test_score': array([0.74901905, 0.91610952, 0.91614762, 0.9159809
5]),
 'split2_test_score': array([0.74891785, 0.91585674, 0.91585198, 0.9158234
1]),
 'split3_test_score': array([0.7485988 , 0.91549007, 0.91559007, 0.9154900
7]),
 'mean_test_score': array([0.74885238, 0.91580595, 0.91580714, 0.91575833]),
 'std_test_score': array([0.00015558, 0.00022136, 0.00021986, 0.00017758]),
 'rank_test_score': array([4, 2, 1, 3]),
 'split0_train_score': array([0.74897698, 0.91586957, 0.915841  , 0.9158822
7]),
 'split1_train_score': array([0.74885238, 0.91575714, 0.91574127, 0.9157746
]),
 'split2_train_score': array([0.74894167, 0.9158414 , 0.91583505, 0.9158414
]),
 'split3_train_score': array([0.74906706, 0.9159541 , 0.91592553, 0.9159604
5]),
 'mean_train_score': array([0.74895952, 0.91585556, 0.91583571, 0.91586468]),
 'std_train_score': array([7.69224618e-05, 7.03441518e-05, 6.52209093e-05, 6.
73371950e-05])}
```

These results lead me to set max_depth = 2 because that is when the accuracy levels out around .91. Even thought increased depths provide marginally increased results, the increase is so small that the increased depth is more likely to overfit than provide any marginal benefit.

Next, I tested for n_estimators. I could have done that all in one step in my original GridSearchCV, but I wanted to see how each parameter tested indepdendent of each other (i.e. with the other parameter held as a constant).

In [76]:
```
random_classifier2 = RandomForestClassifier(max_depth=2)
parameters = {'n_estimators': [1,5,10,20,50,100,1000]}
random_grid2 = GridSearchCV(random_classifier2, parameters, cv = 4)

random_grid2.fit(prin_comps, training_data_scaled['Class'])
```

Out[76]:
```
GridSearchCV(cv=4, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, cr
iterion='gini',
            max_depth=2, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [1, 5, 10, 20, 50, 100, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [78]:
```
random_grid2.cv_results_
```

Out[78]:

```
{'mean_fit_time': array([  0.84288883,   3.16738009,   6.10882866,  12.25902063,
         32.36601311,  63.80614704, 622.02916372]),
 'std_fit_time': array([ 0.09418476,  0.04252745,  0.04659849,  0.32248454,  1.10252549,
         0.64457712, 10.31595633]),
 'mean_score_time': array([ 0.07811517,  0.17668945,  0.31599194,  0.60473961,  1.46850598,
         3.06798303, 28.5679425 ]),
 'std_score_time': array([0.03316344, 0.01553016, 0.01214375, 0.00612832, 0.06037476,
        0.11344803, 0.97550052]),
 'param_n_estimators': masked_array(data=[1, 5, 10, 20, 50, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 1},
  {'n_estimators': 5},
  {'n_estimators': 10},
  {'n_estimators': 20},
  {'n_estimators': 50},
  {'n_estimators': 100},
  {'n_estimators': 1000}],
 'split0_test_score': array([0.91576707, 0.91549088, 0.91576231, 0.91549564, 0.91576231,
        0.91574326, 0.9157385 ]),
 'split1_test_score': array([0.91574762, 0.91572857, 0.91575714, 0.9157619 , 0.9157619 ,
        0.91572857, 0.91574762]),
 'split2_test_score': array([0.9144381 , 0.9144381 , 0.91445238, 0.91445238, 0.91444286,
        0.91445238, 0.91445238]),
 'split3_test_score': array([0.91455674, 0.91451388, 0.91453769, 0.91451388, 0.91451388,
        0.91451388, 0.91450435]),
 'mean_test_score': array([0.91512738, 0.91504286, 0.91512738, 0.91505595, 0.91512024,
        0.91510952, 0.91511071]),
 'std_test_score': array([0.0006314 , 0.00057369, 0.00063307, 0.00058091, 0.00064236,
        0.00062679, 0.00063262]),
 'rank_test_score': array([1, 7, 1, 6, 3, 5, 4]),
 'split0_train_score': array([0.91493955, 0.9148951 , 0.91491256, 0.91489352, 0.91491256,
        0.91494907, 0.91494272]),
 'split1_train_score': array([0.91495238, 0.91494127, 0.91494286, 0.91494286, 0.91494127,
        0.91494127, 0.91494127]),
 'split2_train_score': array([0.91538254, 0.91538254, 0.91538413, 0.91538254, 0.91538095,
        0.91538413, 0.91538413]),
 'split3_train_score': array([0.91534299, 0.91534458, 0.91532236, 0.91534458, 0.91534458,
        0.91534458, 0.91534299]),
 'mean_train_score': array([0.91515436, 0.91514087, 0.91514048, 0.91514087, 0.91514484,
        0.91515476, 0.91515278]),
```

```
 'std_train_score': array([0.00020892, 0.00022369, 0.00021415, 0.00022377, 0.
00021854,
        0.00021008, 0.00021128])}
```

While the results of this grid search say that the best parameter is n_estimators = 1, if I were to set n_estimators = 1, that would just be a decision tree. Additionally, the results for all tested n_estimators parameters resulted in very consistent accuracies, with every one being approximately .91. Because of this, I ended up choosing n_estimators = 100 because a paper (https://www.researchgate.net/publication/230766603_How_Many_Trees_in_a_Random_Forest (https://www.researchgate.net/publication/230766603_How_Many_Trees_in_a_Random_Forest)) says the ideal amount of trees is 64 to 128 and that past around 128, doubling or increasing the amount of trees only increases computational cost, not peformance results.

Final Model: Training and Validation

This is the final pipeline I ended up constructing.

In [199]:
```
final_pipeline = Pipeline([
        ('column_selector', ColumnSelector(cols=['A','B','D'])),
        ('scaler',StandardScaler()),
        ('pca',
         PCA(n_components=1)),
        ('classifier',
         RandomForestClassifier(max_depth=2,n_estimators=100))
    ])
```

To evaluate the model, I performed K-fold cross validation on it one last time. The accuracy held around .91 in all trials.

In [96]:
```
cross_validate(final_pipeline, df.drop('Class',axis=1), df['Class'], cv=6)
```

```
c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\sklearn\utils\deprecati
on.py:125: FutureWarning: You are accessing a training score ('train_score'),
which will not be available by default any more in 0.21. If you need training
scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[96]:
```
{'fit_time': array([58.47530723, 59.58823943, 57.87677455, 56.5878427 , 56.27
88415 ,
        55.39300299]),
 'score_time': array([1.34944582, 1.39280057, 1.37487626, 1.34691811, 1.34171
867,
        1.32859993]),
 'test_score': array([0.91656542, 0.91489543, 0.91632042, 0.916605  , 0.91587
458,
        0.91607916]),
 'train_score': array([0.91596992, 0.91640092, 0.91600792, 0.915784  , 0.9161
4108,
        0.91620617])}
```

Next, the pipeline was trained on the entire dataset for deployment.

In [200]:
```
final_pipeline.fit(df, df['Class'])
```

Out[200]:
```
Pipeline(memory=None,
     steps=[('column_selector', ColumnSelector(cols=['A', 'B', 'D'], drop_axi
s=False)), ('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tru
e)), ('pca', PCA(copy=True, iterated_power='auto', n_components=1, random_sta
te=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('classifier', Rando...obs=Non
e,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))])
```

The final step is deploying the model. I was not able to convert my pipeline to an ONNX file, as seen below. From research, it seems that I would have to register ColumnSelector as an ONNX converter (http://onnx.ai/sklearn-onnx/api_summary.html#skl2onnx.update_registered_converter (http://onnx.ai/sklearn-onnx/api_summary.html#skl2onnx.update_registered_converter)).

ONNX Attempt 1

In [194]:
```
# input_types = dict([(x, FloatTensorType([1, 1])) for x in labels])

to_keep = ['A','B','D']
input_types = dict([(x, FloatTensorType([1,15])) for x in to_keep])

print(input_types)
try:
    model_onnx = convert_sklearn(final_pipeline,
                         'final_pipeline_Victoria_Belotti_onnx',
                                initial_types=list(input_types.items()))
except Exception as e:
    print(e)

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

```
{'A': FloatTensorType(shape=[1, 15]), 'B': FloatTensorType(shape=[1, 15]),
'D': FloatTensorType(shape=[1, 15])}
'RandomForestClassifier' object has no attribute 'classes_'

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-194-3ef086b46d49> in <module>
     13
     14 with open("final_pipeline_Victoria_Belotti.onnx", "wb") as f:
---> 15     f.write(model_onnx.SerializeToString())
     16

NameError: name 'model_onnx' is not defined
```

ONNX Attempt 2

In [191]:
```
labels
```

Out[191]:
```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

In [198]:
```
input_types = dict([(x, FloatTensorType([1,15])) for x in labels])
model_onnx = convert_sklearn(final_pipeline, initial_types=list(input_types.items()))
with open("onnx_test.onnx", "wb") as f:
    f.write(model_onnx.SerializeToString())
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-198-c3c1a97a3057> in <module>
      1 input_types = dict([(x, FloatTensorType([1,15])) for x in labels])
----> 2 model_onnx = convert_sklearn(final_pipeline, initial_types=list(input_types.items()))
      3 with open("onnx_test.onnx", "wb") as f:
      4     f.write(model_onnx.SerializeToString())

c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\skl2onnx\convert.py in convert_sklearn(model, name, initial_types, doc_string, target_opset, custom_conversion_functions, custom_shape_calculators, custom_parsers, options)
    118
    119     # Infer variable shapes
--> 120     topology.compile()
    121
    122     # Convert our Topology object into ONNX. The outcome is an ONNX model.

c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\skl2onnx\common\_topology.py in compile(self)
    822         self._resolve_duplicates()
    823         self._fix_shapes()
--> 824         self._infer_all_types()
    825         self._check_structure()
    826

c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\skl2onnx\common\_topology.py in _infer_all_types(self)
    675                 self.custom_shape_calculators[operator.type](operator)
    676             else:
--> 677                 operator.infer_types()
    678
    679     def _resolve_duplicates(self):

c:\users\tbelo\anaconda3\envs\thing\lib\site-packages\skl2onnx\common\_topology.py in infer_types(self)
    124         if self.type is None:
    125             raise RuntimeError("Unable to find a shape calculator for type "
--> 126                                "'{}'.".format(type(self.raw_operator)))
    127         try:
    128             shape_calc = _registration.get_shape_calculator(self.type)

RuntimeError: Unable to find a shape calculator for type '<class 'mlxtend.feature_selection.column_selector.ColumnSelector'>'.
```

Conclusion

While cross validation has shown that my model most likely does not overfit or hyperfit the data, I believe more testing could be done to narrow down which feature subset of A, B, and D is truly the most optimal. A positive result is that my model seems to peform well when cross validated, but a caveat is that I discarded majority of the information and data given, which means some crucial piece of information or contributed variance could have been lost in one of the discarded features.

References

Links I drew specific code or packages from are in-line above.

Other links:

- https://machinelearningmastery.com/k-fold-cross-validation/# (https://machinelearningmastery.com/k-fold-cross-validation/#)
- https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn#step-7 (https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn#step-7)
- https://stats.stackexchange.com/questions/111968/random-forest-how-to-handle-overfitting (https://stats.stackexchange.com/questions/111968/random-forest-how-to-handle-overfitting)
- https://chrisalbon.com/machine_learning/model_evaluation/cross_validation_pipeline/ (https://chrisalbon.com/machine_learning/model_evaluation/cross_validation_pipeline/)
- https://www.datascience.com/blog/machine-learning-generalization (https://www.datascience.com/blog/machine-learning-generalization)