October 16, 2018

1 Data

- Title: Condition monitoring of hydraulic systems
- Relevant Information:
  – The data set was experimentally obtained with a hydraulic test rig. This test rig consists of a primary working and a secondary cooling-filtration circuit which are connected via the oil tank [1], [2]. The system cyclically repeats constant load cycles (duration 60 seconds) and measures process values such as pressures, volume flows and temperatures while the condition of four hydraulic components (cooler, valve, pump and accumulator) is quantitatively varied.
  – Number of Instances: 2205
  – Number of Attributes: 43680 (8x60 (1 Hz) + 2x600 (10 Hz) + 7x6000 (100 Hz))
- Attribute Information: Attributes are sensor data (all numeric and continuous) from measurements taken at the same point in time, respectively, of a hydraulic test rig's working cycle. The sensors were read with different sampling rates, leading to different numbers of attributes per sensor despite they were all exposed to the same working cycle.
  1. Pressure sensors (PS1-6): 100 Hz, 6000 attributes per sensor (6 sensors)
  2. Motor power sensor (EPS1): 100 Hz, 6000 attributes per sensor (1 sensor)
  3. Volume flow sensors (FS1/2): 10 Hz, 600 attributes per sensor (2 sensors)
  4. Temperature sensors (TS1-4): 1 Hz, 60 attributes per sensor (4 sensors)
  5. Vibration sensor (VS1): 1 Hz, 60 attributes per sensor (1 sensor)
  6. Efficiency factor (SE): 1 Hz, 60 attributes per sensor (1 sensor)
  7. Virtual cooling efficiency sensor (CE): 1 Hz, 60 attributes per sensor (1 sensor)
  8. Virtual cooling power sensor (CP): 1 Hz, 60 attributes per sensor (1 sensor)
- Missing Attribute Values: None
- Class Distribution: there are 5 different class value vectors provided in profile.txt. All but number 5 (stable flag) describe degradation processes over time and, thus, their values do not represent distinct categories, but continuous values.
  1. Cooler condition:
     3: close to total failure (732 instances)
     20: reduced effifiency (732 instances)
     100: full efficiency (741 instances)

2. Valve condition :
   100: optimal switching behavior (1125 instances)
   90: small lag (360 instances)
   80: severe lag (360 instances)
   73: close to total failure (360 instances)
3. Internal pump leakage:
   0: no leakage (1221 instances)
   1: weak leakage (492 instances)
   2: severe leakage (492 instances)
4. Hydraulic accumulator / bar:
   130: optimal pressure (599 instances)
   115: slightly reduced pressure (399 instances)
   100: severely reduced pressure (399 instances)
   90: close to total failure (808 instances)
5. stable flag:
   0: conditions were stable (1449 instances)
   1: static conditions might not have been reached yet (756 instances)

2 Summary

In this analysis, I used internal pump leakage as the target variable, which is a three-level categorial variable. In this analysis, I used summary statistics, median, to represent all the attributes in each sensor. In other words, for each sensor, I computed median of all the attributes in that sensor. Thus, I reduced the dimensionality from 43680 to 17. The rational behind is that I picked six sensors, computed Spearman rank correlation for all attributes in each sensor, and found that most of the attributes in each sensor are highly correlated. For example, in TS, the correlations of almost all pairs of attributes are high than 0.99. So I use median to represent each sensor. I choose median over mean because median is more robust to outliers.

The description of the data file report that there is no missing data. I still check the data set to see if there is any missing value as a safety measure. There is no missing data in the file. I draw boxplots for each sensor and it showed that SE, PS1, PS2, and EPS1 seem to contain outliers. I used Z-score function to detect the outliers. If the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers. I found 20 rows from the dataset with outliers and had them removed. Now the data set is clean. I split up the data set to 70% training data and 30% test data. Then I performed the following classification algorithms on the 17 dimensional data. For each classification model, I report the classification score (the number of correct predictions made by the model over all kinds predictions made) and the confusion matrix. For RF and GBM, I also report the variable importance plots.

1. Multinomial Logistic Regression (MLR). I performed principal component analysis (PCA) on all the features to define the linear combinations of the attributes, for use in the multinomial logistic regression to predict the target variable. In other words, I used principal component regression (PCR) on the entire data set. In order to choose the number of components M, I computed 10-fold cross validation MSE obtained using PCR, as a function of M, on the

training data. Looking at the scree plot, it turned out that with $M = 6$, the multinomial logistic regression yielded one of the lowest MSE. Then I performed MLR on the transformed test data (each dimension now is a linear combination of different sensors). The MSE is 0.039 and the accuracy score is 0.962. The precision, recall, and F1 score are all 0.96.

2. Linear Discriminant Analysis (LDA). I trained LDA on the training data and predicted the test data. The accuracy score is 0.994. The precision, recall, and F1 score are all 0.99.

3. Random Forest (RF). I trained RF on the training data and tried to find the best set of tuning parameters using grid search and cross-validation. I predicted with RF using the best set of tuning parameter on the test data. The accuracy score is 0.992. The precision, recall, and F1 score are all 0.99. The variable importance plot shows that the sensor SE is the most important predictor for internal pump leakage, followed by FS1.

4. Support Vector Machine (SMV). I trained SMV on the training data and tried to find the best set of tuning parameters using grid search and cross-validation. I predicted with SMV using the best set of tuning parameter on the test data. The accuracy score is 0.982. The precision, recall, and F1 score are all 0.98.

5. k-Nearest Neighbors (KNN). I trained KNN on the training data and predicted the test data with different number of neighbors (k). I plotted both the training accuracy and the testing accuracy for different ks. It turned out that when $k = 1$, the test accuracy is the highest. When $k = 1$, it does not necessarily means that there is a overfitting problem. It is vey likely that there is a clear cut in the target variable. The accuracy score is 0.992. The precision, recall, and F1 score are all 0.99.

6. Gradient Boosted Machine (GBM). I trained GBM on the training data and tried to find the best set of tuning parameters using grid search and cross-validation. I predicted with GBM using the best set of tuning parameter on the test data. . The accuracy score is 0.992. The precision, recall, and F1 score are all 0.99. The variable importance plot shows that the sensor SE is the most important predictor for internal pump leakage, followed by FS1.

Overall, all six classification algorithms working very well in terms of predicting the test data. The accuracy scores of LDA, RF, KNN, and GBM are among the highest, larger than 0.99. MLR has the lowest accuracy score, 0.96. The variable importance plot shows that the sensors SE and FS1 are the top two most important predictors in predicting performance of the internal pump leakage, as suggested by both RF and GBM.

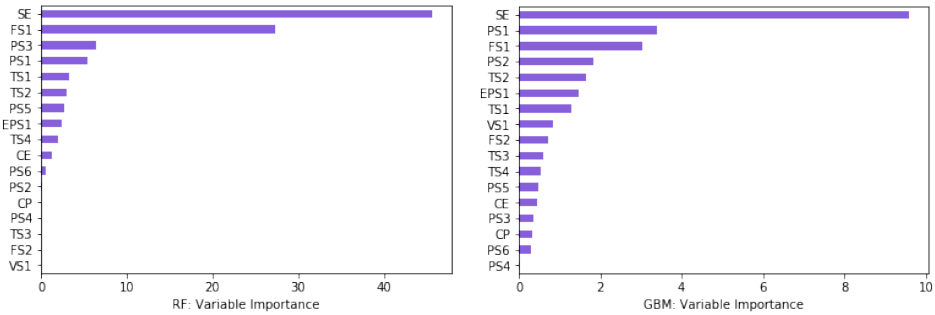


Figure 1. The variable importance plots from Random Forest and Gradient Boosted Machine

Table 1 shows accuracy score for each classification algorithm using Method 1 and Method 2. I did not perform multinomial logistic regression for Method 1. The reason is that I

use principal component regression with MLR to figure out the number of components. It would be inappropriate to use MLR on that results. As shown in the table, for Method 1, LDA, RF, KNN, and GBM perform similarly well and the accuracy score is higher than 0.97; for Method 2, RF, SVM, KNN, and GBM perform similarly well and the accuracy score is higher than 0.98.

Table 1: Accuracy score for each classification algorithm using Method 1 and Method 2

	Method 1	Method 2
Multinomial Logistic Regression	-	0.654
Linear Discriminant Analysis	0.989	0.761
Random Forest	0.982	0.986
Supoort Vector Machine	0.553	0.982
k-Nearest Neighbors	0.983	0.988
Gradient Boosted Machine	0.971	0.989

3 Analysis: Step by Step

In [44]:
```
#Importing all the libraries necessary for this project

import pandas as pd
import numpy as np
import zipfile
import csv
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, scale
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn import neighbors
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [4]:
```
# Load all attribute data from txts
teste_list = ['VS1', 'TS1', 'TS2', 'TS3', 'TS4','SE', 'PS1', 'PS2', 'PS3', 'PS4', '
              'PS6', 'FS1', 'FS2', 'EPS1', 'CP', 'CE']

teste_list = [s + '.txt' for s in teste_list]

new_df = {}
master_df = pd.DataFrame()
for key in teste_list:
    df = pd.read_table(key, header=None)
    master_df = pd.concat([master_df, df], axis=1)

# Load the target variable, only keep the internal pump leakage as target variable.
# Internal pump leakage has three levels, 0, 1, and 2.
target_df = pd.read_table('profile.txt', header=None)
target_df = target_df.drop([0,1,3,4], axis=1)
```

In [6]:
```
# Data Quality Check
print(master_df.head())
print(master_df.dtypes)
print(master_df.shape)
print(target_df.head())
print(target_df.dtypes)
print(target_df.shape)
```

```
      0      1      2      3      4      5      6      7      8      9   \
0  0.604  0.605  0.611  0.603  0.608  0.608  0.608  0.617  0.619  0.619
1  0.590  0.610  0.626  0.620  0.623  0.619  0.617  0.618  0.619  0.615
2  0.578  0.603  0.638  0.651  0.652  0.662  0.662  0.656  0.652  0.638
3  0.565  0.591  0.608  0.614  0.623  0.645  0.642  0.645  0.642  0.643
4  0.570  0.600  0.623  0.636  0.644  0.642  0.651  0.654  0.660  0.644

    ...        50      51      52      53      54      55      56      57  \
0   ...    31.554  30.953  30.639  30.561  30.368  30.224  29.790  29.261
1   ...    23.995  24.328  24.283  23.877  23.816  23.933  23.354  23.483
2   ...    21.711  21.564  21.564  21.526  21.753  21.749  21.802  21.582
3   ...    20.687  20.703  20.295  20.482  20.600  20.547  20.708  20.708
4   ...    19.887  19.919  19.696  19.634  19.747  20.005  19.919  19.736

       58      59
0  29.287  28.866
1  23.320  23.588
2  21.283  21.519
3  20.574  20.403
4  19.977  20.016

[5 rows x 43680 columns]
0     float64
1     float64
2     float64
3     float64
4     float64
5     float64
6     float64
7     float64
8     float64
9     float64
10    float64
11    float64
12    float64
13    float64
14    float64
15    float64
16    float64
17    float64
18    float64
19    float64
20    float64
21    float64
22    float64
23    float64
24    float64
25    float64
```

```
26    float64
27    float64
28    float64
29    float64
       ...
30    float64
31    float64
32    float64
33    float64
34    float64
35    float64
36    float64
37    float64
38    float64
39    float64
40    float64
41    float64
42    float64
43    float64
44    float64
45    float64
46    float64
47    float64
48    float64
49    float64
50    float64
51    float64
52    float64
53    float64
54    float64
55    float64
56    float64
57    float64
58    float64
59    float64
Length: 43680, dtype: object
(2205, 43680)
   2
0  0
1  0
2  0
3  0
4  0
2    int64
dtype: object
(2205, 1)
```

In [7]:
```
# Check if there is any missing data.
# In fact, the data set website said there is no missing value in this data set. So
def draw_missing_data_table(df):
    total = df.isnull().sum().sort_values(ascending=False)
    percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    return missing_data

draw_missing_data_table(master_df)
```

Out[7]:

	Total	Percent
59	0	0.0
2197	0	0.0
2205	0	0.0
2204	0	0.0
2203	0	0.0
2202	0	0.0
2201	0	0.0
2200	0	0.0
2199	0	0.0
2198	0	0.0
2196	0	0.0
2186	0	0.0
2195	0	0.0
2194	0	0.0
2193	0	0.0
2192	0	0.0
2191	0	0.0
2190	0	0.0
2189	0	0.0
2188	0	0.0
2206	0	0.0
2207	0	0.0
2208	0	0.0
2209	0	0.0
2226	0	0.0
2225	0	0.0
2224	0	0.0
2223	0	0.0
2222	0	0.0
2221	0	0.0
...	...	...
4740	0	0.0
4739	0	0.0
4738	0	0.0
4737	0	0.0
4736	0	0.0
4735	0	0.0

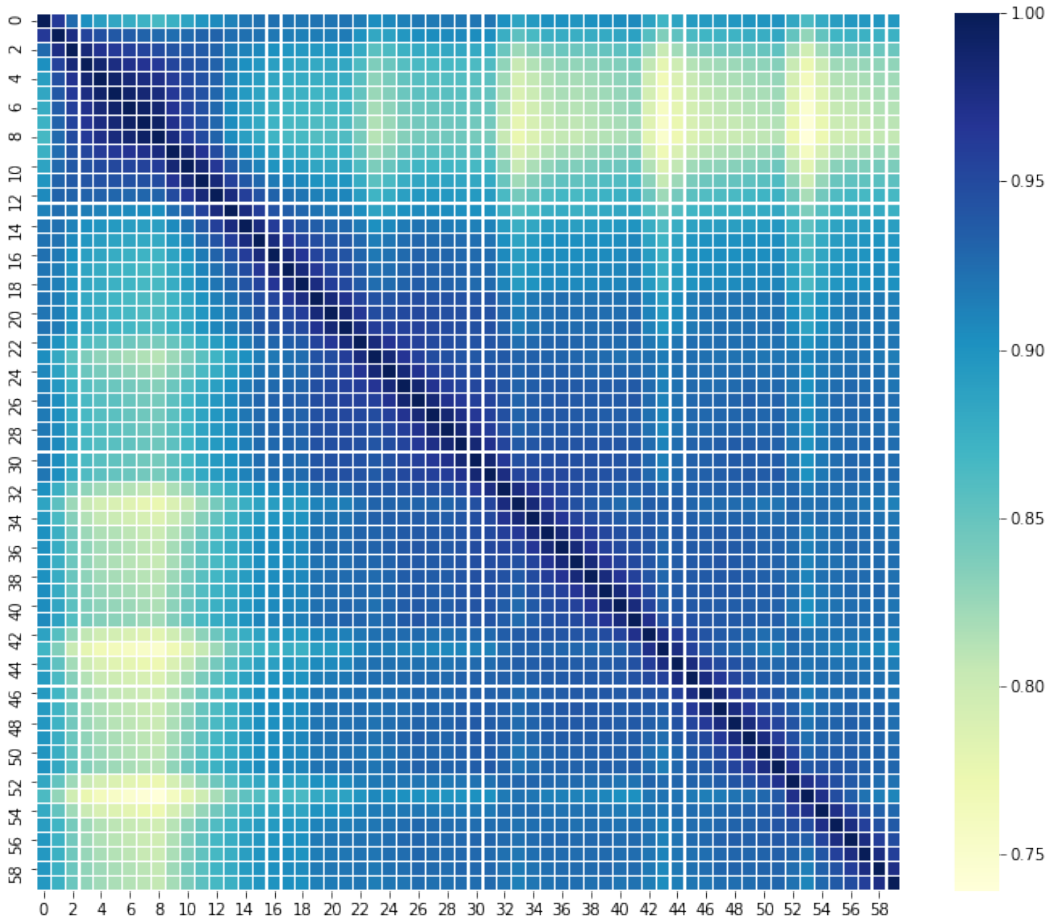
```
4734      0      0.0
4733      0      0.0
4750      0      0.0
4752      0      0.0
4771      0      0.0
4753      0      0.0
4770      0      0.0
4769      0      0.0
4768      0      0.0
4767      0      0.0
4766      0      0.0
4765      0      0.0
4764      0      0.0
4763      0      0.0
4762      0      0.0
4761      0      0.0
4760      0      0.0
4759      0      0.0
4758      0      0.0
4757      0      0.0
4756      0      0.0
4755      0      0.0
4754      0      0.0
0         0      0.0

[43680 rows x 2 columns]
```

In [16]:
```
# I picked VS and LS to see the correlation of all attibutes from a type of sensor
VS = pd.read_table('VS1.txt', header=None)

list1 = ['TS1', 'TS2', 'TS3', 'TS4']
list1 = [s + '.txt' for s in list1]
new_df = {}
TS = pd.DataFrame()
for key in list1:
    df = pd.read_table(key, header=None)
    TS = pd.concat([TS, df], axis=1)
```

In [20]:
```
# Spearman rank correlation for all the features in VS shown by heatmap
# It seems like most of the features in a type of sensor are highly correlated
print("The dimension of VS is : ", VS.shape)
corrmat1 = VS.corr(method='spearman')
f, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(corrmat1, ax=ax, cmap="YlGnBu", linewidths=0.1)
```

```
The dimension of VS is :  (2205, 60)
```

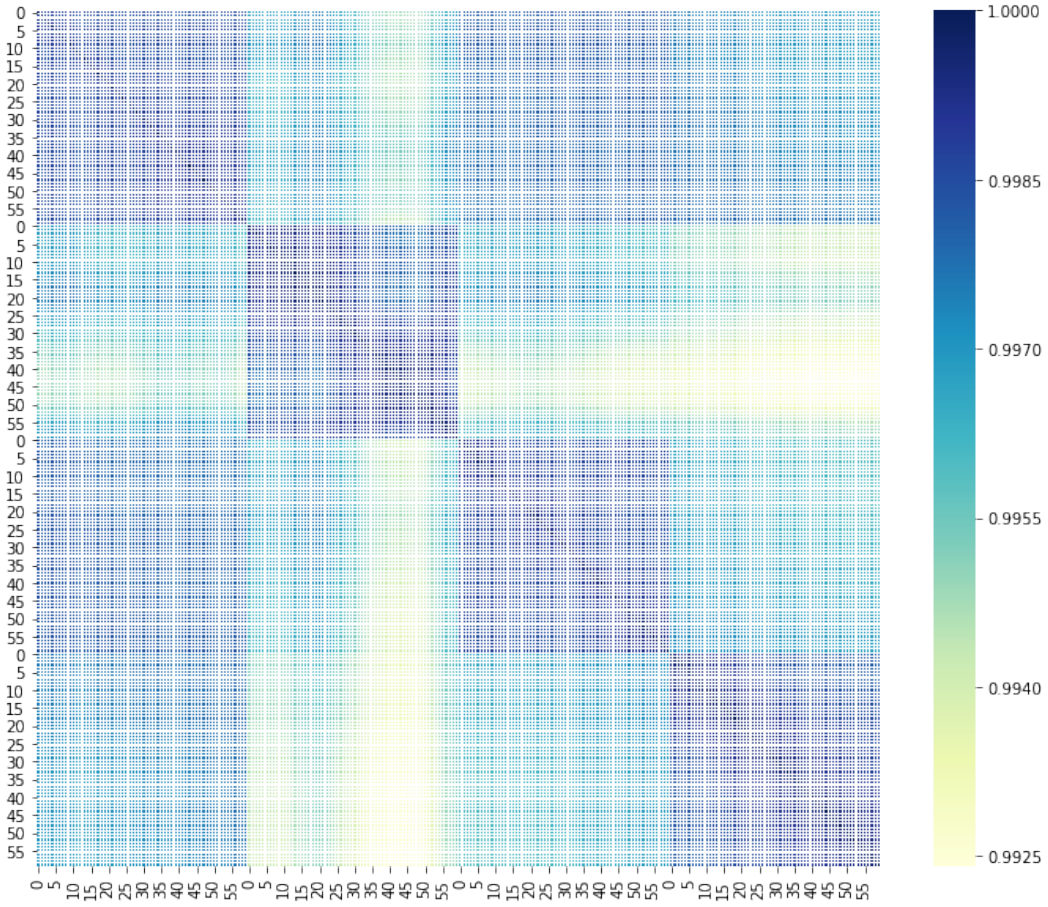
Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1ef5a358>
```



In [18]:
```
# Spearman rank correlation for all the features in TS shown by heatmap
# It seems like almost all of the features in TS are highly correlated
# Even the lightest color represent correlation > 0.99

print("The dimension of TS is : ", TS.shape)
corrmat = TS.corr(method='spearman')
f, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(corrmat, ax=ax, cmap="YlGnBu", linewidths=0.1)
```

```
The dimension of TS is :  (2205, 240)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a189630b8>
```



In [37]:
```
# Perform principal component regression (PCR) on the data
# Combine principal component analysis with multinomial logistic regression
# 10-folder cross-validation applied to get the best number of components with the

# Scale the data to be between -1 and 1
scaler = StandardScaler()
X = scaler.fit_transform(master_df)
pca = PCA()
X_reduced = pca.fit_transform(scale(master_df))


# 10-fold CV, with shuffle
n = len(X_reduced)
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)
regr = LogisticRegression(solver='newton-cg', multi_class='multinomial')
mse = []
# Calculate MSE with only the intercept (no principal components in regression)
score = -1*cross_val_score(regr, np.ones((n,1)), target_df.values.ravel(), cv=kf_1
```

```
        mse.append(score)
        print(score)
        # Calculate MSE using CV for the 19 principle components, adding one component at
        for i in np.arange(1, 20):
            score = -1*cross_val_score(regr, X_reduced[:,:i],target_df.values.ravel(), cv=
            mse.append(score)
            print(score)
```

```
1.1155368161250514
1.1155368161250514
0.9245680789798436
```

```
/anaconda3/lib/python3.6/site-packages/scipy/optimize/linesearch.py:313: LineSearchWarning:
  warn('The line search algorithm did not converge', LineSearchWarning)
/anaconda3/lib/python3.6/site-packages/sklearn/utils/optimize.py:195: UserWarning: Line Sea
  warnings.warn('Line Search failed')
```

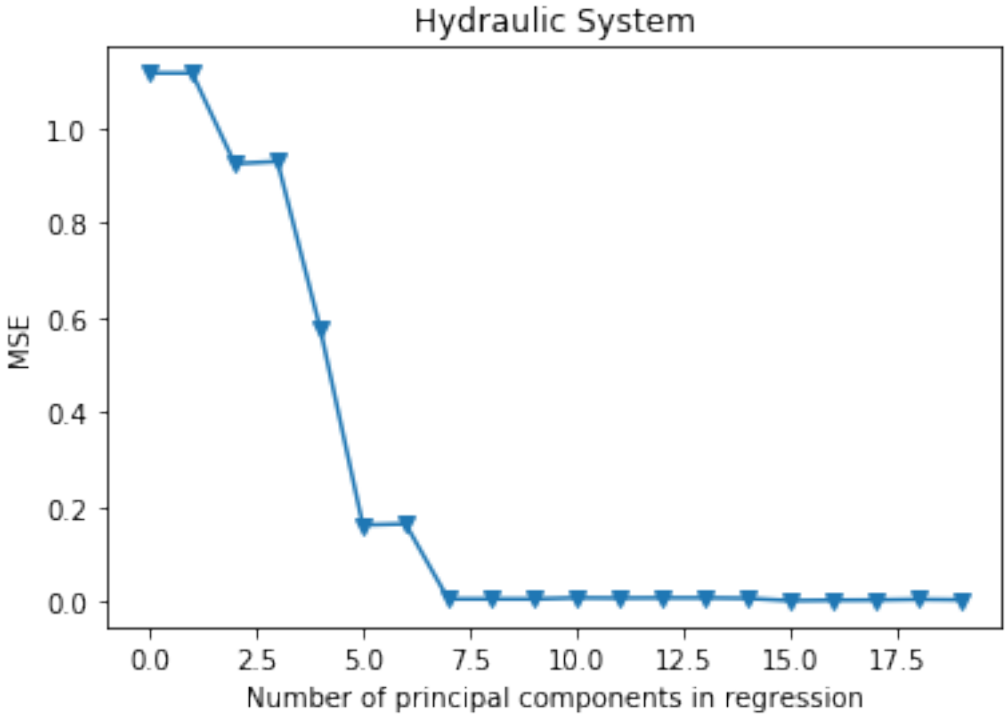
```
0.9291032496914851
0.577538050185109
0.16278897573015216
0.16505553270259155
0.006803784450843274
0.006803784450843274
0.006803784450843274
0.009066227889757302
0.008615795968737145
0.009070341423282603
0.009070341423282603
0.008163307280954341
0.0031797614150555326
0.0045413410119292475
0.004539284245166598
0.005892636774989716
0.0054401480872069115
```

```
In [38]: score_15comp = -1*cross_val_score(regr, X_reduced[:,:15], target_df.values.ravel()
         print(score_15comp)
         #With 15 components, the logisitc regression has the lowest MSE for 10-fold cross-
```

```
0.0031797614150555326
```

```
In [18]: plt.plot(mse, '-v')
         plt.xlabel('Number of principal components in regression')
         plt.ylabel('MSE')
         plt.title('Hydraulic System')
```

```
plt.xlim(xmin=-1);
# The plot below indicates that the lowest training MSE is reached when doing regr
```



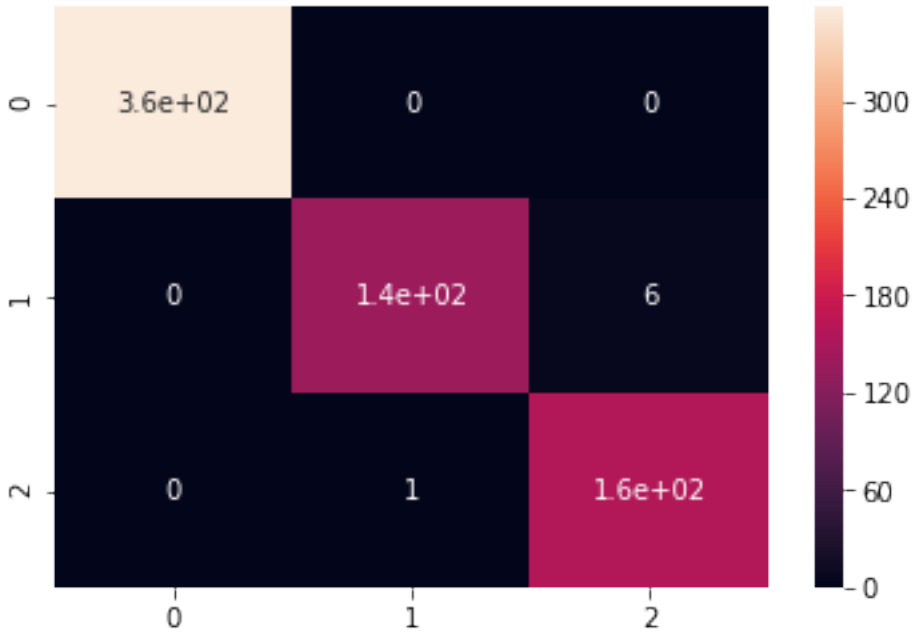
In [42]:
```
# Transform attribute data with PCA loadings and fit classification models on 15 p
xnew = X_reduced[:,:15]
y = target_df.values.ravel()

# Splitting up data into 70% train data and 30% test data
x_train, x_test, y_train, y_test = train_test_split(xnew, y, test_size=0.3, random
```

In [45]:
```
# LDA applied to train data
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)

# LDA applied to the test data
predictions_lda = lda.predict(x_test)
# Print Accuracy Score for LDA
print("Accuracy Score is:")
print(accuracy_score(y_test, predictions_lda))
print()
# Classification Report of Prediction
print("Classification Report:")
print(classification_report(y_test, predictions_lda))
```

```
        # Confusion Matrix for predictions made
        conf2 = confusion_matrix(y_test,predictions_lda)
        conf2
        # Plot Confusion Matrix
        label = ["0","1","2"]
        sns.heatmap(conf2, annot=True, xticklabels=label, yticklabels=label)
```

```
Accuracy Score is:
0.9894259818731118

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       358
          1       0.99      0.96      0.98       145
          2       0.96      0.99      0.98       159

avg / total       0.99      0.99      0.99       662
```

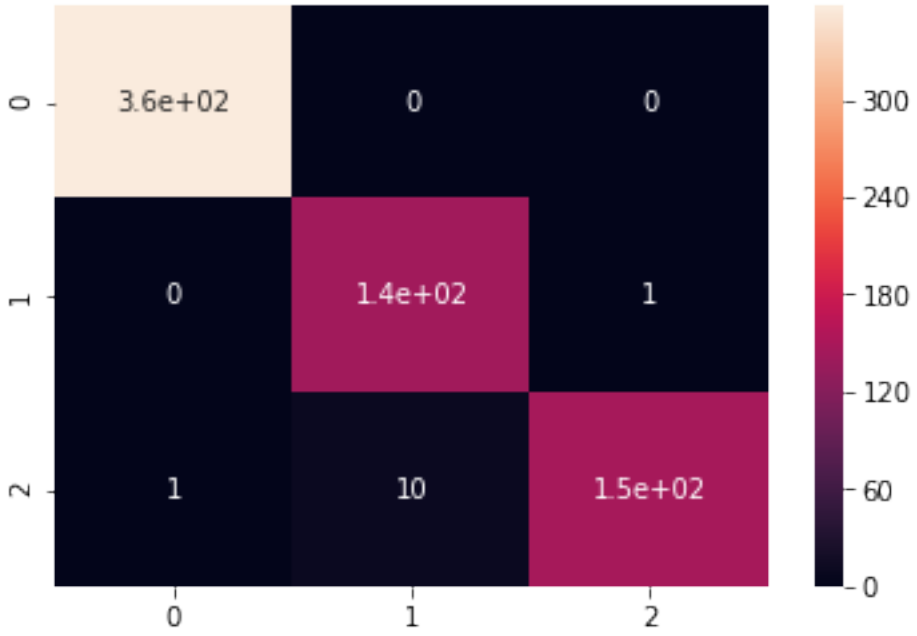
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0ca47358>



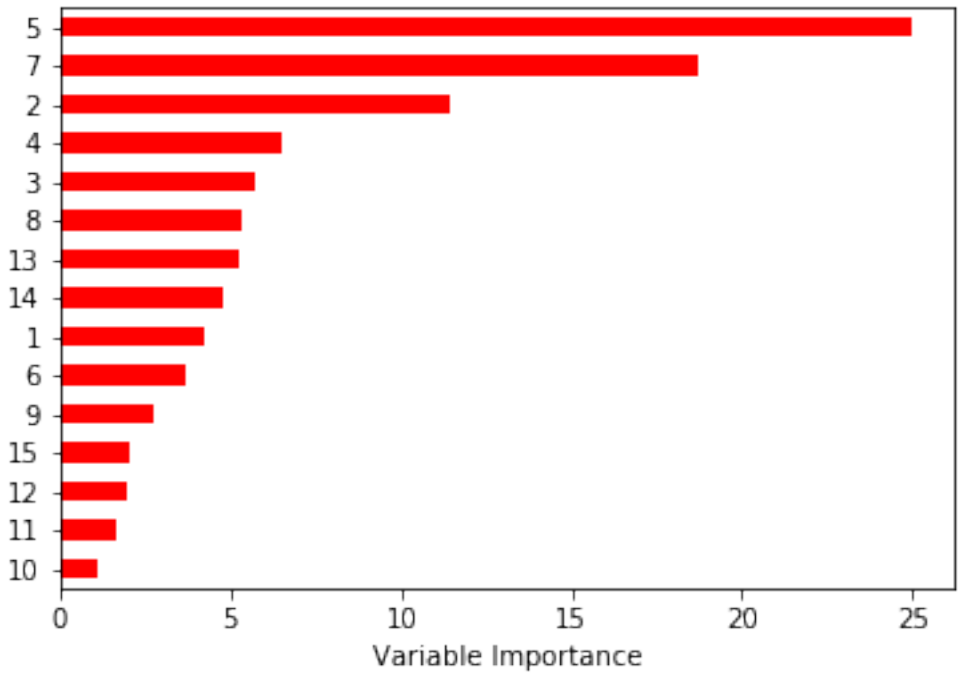
In [46]: *# Random Forest Classification Algorithm applied to train data*

```
RF = RandomForestClassifier()
RF.fit(x_train, y_train)
RF.score(x_train, y_train)
# Predict Random Forest Algorithm on Test Data
predictions_RF = RF.predict(x_test)
# Print Accuracy Score for Random Forest Algorithm
print("Accuracy Score is: ")
print(accuracy_score(y_test, predictions_RF))
print()
# Classification Report of Prediction
print("Classification Report: ")
print(classification_report(y_test, predictions_RF))
# Confusion Matrix for predictions made
conf = confusion_matrix(y_test, predictions_RF)
conf
# Plot Confusion Matrix
label = ["0", "1","2"]
sns.heatmap(conf, annot = True, xticklabels = label, yticklabels = label)
plt.show()
```

```
Accuracy Score is:
0.9818731117824774

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       358
          1       0.94      0.99      0.96       145
          2       0.99      0.93      0.96       159

avg / total       0.98      0.98      0.98       662
```



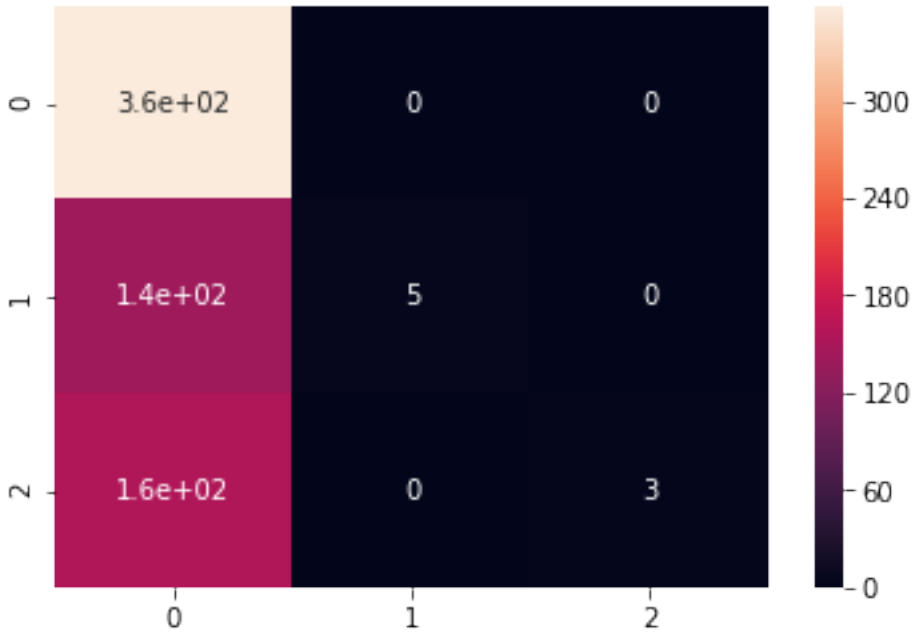

```
In [47]: Importance = pd.DataFrame({'Importance':RF.feature_importances_*100}, index = np.a
         Importance.sort_values('Importance', axis=0, ascending=True).plot(kind='barh', col
         plt.xlabel('Variable Importance')
         plt.gca().legend_ = None
```



```
In [86]: # Trying to see what type of sensor has the largest loading on Components 5, 7, an
         pca = PCA(n_components = 15)
         tt = pca.fit_transform(scale(master_df))

In [84]: loadings = pca.components_
         comp5 = loadings[5]
         index_max = np.argmax(comp5)
         print(index_max)
         comp7 = loadings[7]
         index_max = np.argmax(comp7)
         print(index_max)
         comp2 = loadings[2]
         index_max = np.argmax(comp2)
         print(index_max)
         # It shows FS has attributes with the largest loading on all three componets

13222
6820
12254


In [48]: # Need to train a Support Vector Machine classifier
         # find the parameters by cross-validation
         from sklearn.model_selection import GridSearchCV
         tuned_parameters = [{'C': [0.01, 0.1, 1, 10, 100],
                              'gamma': [0.5, 1,2,3,4]}]
         clf = GridSearchCV(SVC(kernel='rbf'), tuned_parameters, cv=10, scoring='accuracy',
         clf.fit(x_train, y_train)
         #clf.cv_results_

Out[48]: GridSearchCV(cv=10, error_score='raise',
                estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False),
                fit_params=None, iid=True, n_jobs=1,
                param_grid=[{'C': [0.01, 0.1, 1, 10, 100], 'gamma': [0.5, 1, 2, 3, 4]}],
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring='accuracy', verbose=0)

In [49]: # The best parameter set
         print(clf.best_params_)

{'C': 10, 'gamma': 0.5}
```

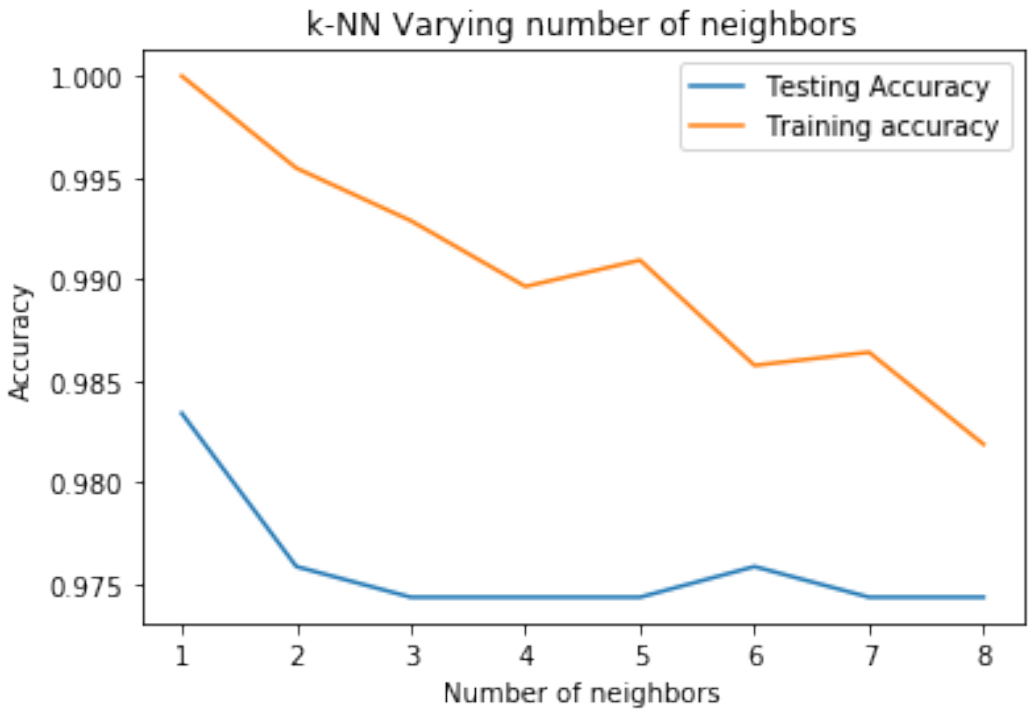
```
In [50]: # Print Accuracy Score for SVM Algorithm
         print("Accuracy Score is: ")
         print(clf.best_estimator_.score(x_test, y_test))
         print()
         # Classification Report of Prediction
         print("Classification Report: ")
         print(classification_report(y_test, clf.best_estimator_.predict(x_test)))
         # Confusion Matrix for predictions made
         conf = confusion_matrix(y_test, clf.best_estimator_.predict(x_test))
         conf
         # Plot Confusion Matrix
         label = ["0", "1","2"]
         sns.heatmap(conf, annot = True, xticklabels = label, yticklabels = label)
         plt.show()
```

```
Accuracy Score is:
0.552870090634441

Classification Report:
              precision    recall  f1-score   support

           0       0.55      1.00      0.71       358
           1       1.00      0.03      0.07       145
           2       1.00      0.02      0.04       159

avg / total       0.76      0.55      0.41       662
```



In [62]:
```
# Need to train k-Nearest Neighbor
# First I would like to observe the accuracies for different values of k.
# Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(x_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(x_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(x_test, y_test)
```

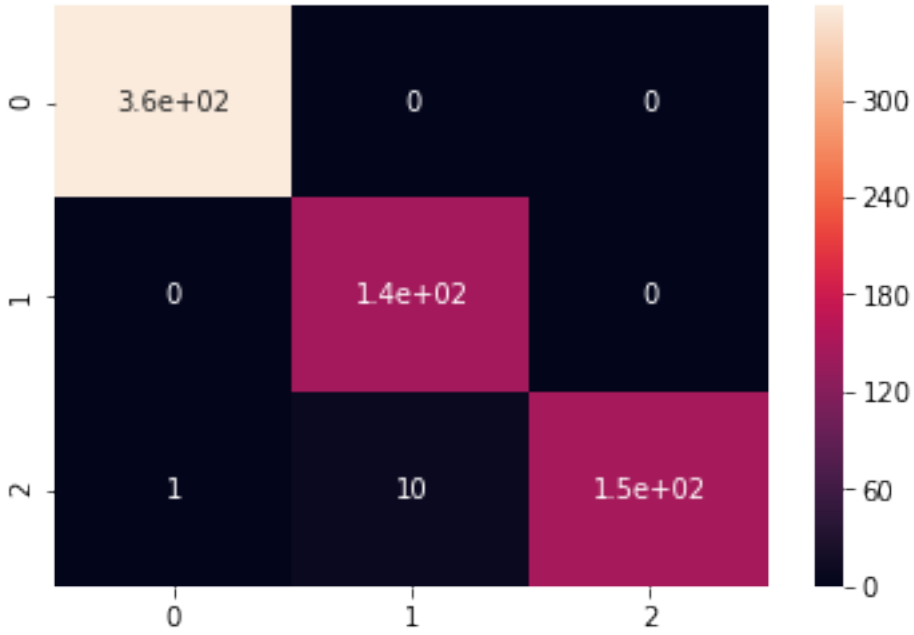
In [63]:
```
#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
# As we can see from the plot, it appears like the test accuracy is highest when u
# but using 2 neighbors or less seems to result in a complex model that over fits
```



In [69]:
```
# KNN applied to training data and predict on test data with 6 neighbors

knn = KNeighborsClassifier(n_neighbors = 6)
knn.fit(x_train, y_train)
pred = knn.predict(x_test)
print("Accuracy Score is: ")
print(knn.score(x_test, y_test))
print()
print("Classification Report: ")
print(classification_report(y_test, pred))
conf = confusion_matrix(y_test, pred)
conf
# Plot Confusion Matrix
label = ["0", "1","2"]
sns.heatmap(conf, annot = True, xticklabels = label, yticklabels = label)
plt.show()
```

```
Accuracy Score is:
0.9758308157099698

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       358
```

```
          1       0.93      0.97      0.95       145
          2       0.98      0.92      0.95       159

avg / total       0.98      0.98      0.98       662
```



In [53]:
```
# Gradient Boosting Classification Algorithm applied to train data

GB = GradientBoostingClassifier(n_estimators=500, learning_rate=0.01, random_state
GB.fit(x_train, y_train)
GB.score(x_train, y_train)
# Predict Random Forest Algorithm on Test Data
predictions_GB = GB.predict(x_test)

predictions_GB
# Print Accuracy Score for Random Forest Algorithm
print("Accuracy Score is: ")
print(accuracy_score(y_test, predictions_GB))
print()
# Classification Report of Prediction
print("Classification Report: ")
print(classification_report(y_test, predictions_GB))
# Confusion Matrix for predictions made
conf = confusion_matrix(y_test, predictions_GB)
```

```
conf
# Plot Confusion Matrix
label = ["0", "1","2"]
sns.heatmap(conf, annot = True, xticklabels = label, yticklabels = label)
plt.show()
```

```
Accuracy Score is:
0.9712990936555891

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       358
          1       0.92      0.95      0.94       145
          2       0.95      0.93      0.94       159

avg / total       0.97      0.97      0.97       662
```