Anomaly Detection

J.R. Powers-Luhn

***Abstract*—Principal component analysis is employed as a mechanism to detect outlier measurements from temperature and sightglass level readings in a Slurry Red Ceramic Melter. Two statistical metrics are employed to detect outliers: Hotelling's T^2 and the Q-statistic. The process of setting thresholds for these statistics is discussed and a brief exploration of the use of these statistics in diagnostics is offered.**

I. Introduction

Interrelated process failures fall into two categories. The failure can be within the model (meaning that the measured values are within the normal bounds in which the model was trained but do not follow the established relationships) or outside of the model (meaning that there is variation that cannot be explained by the model). A well designed fault detection system should be able to identify each of these types of faults.

Two statistics that can be used to detect these types of faults are Hotelling's T^2 (for detecting variation within a model) and Q-statistics (for detecting deviation from a model). These statistics measure deviation by using a principal component analysis transformation then calculating the deviation from the model. Simple thresholds can be set using known training data. Values that then exceed these thresholds can trigger alarms or investigation of the instruments that produced the measurements. Further, it is possible to calculate the relative contribution from each measurement in a data point to the T^2 and Q statistic, assisting in diagnostics.

II. Methodology

In order to measure a measurement's deviation within a model, Hotelling's T^2 (equation 1) statistic is used. A generalization of the student's T-test, this measures how far the measurement has departed from the center of the principal component space.

$$T_i^2 = t_i \lambda^- 1 t_i^T \tag{1}$$

If there is variation outside of the model the residuals will be high. This is captured by a second statistic: the Q statistic (equation 2), which measures deviation outside the model.

$$Q_i = x_i(\mathbf{I} - \mathbf{P}_k \mathbf{P}_k^T) x_i^T \tag{2}$$

The values of these statistics, calculated on a "known good" training set, can be used to set alarm thresholds for future data.

A PCA model was generated using a training data set of 450 samples of 21 values each. The number of principal components included in the model was determined by increasing the number of components and calculating the T^2 and Q statistics. After two PC's were included the T^2 values did not change and the Q statistics were reduced by an order of magnitude. Adding additional components did continue to lower as more components were added, but the rate decreased significantly–the next order-of-magnitude reduction threshold was not reached until 9 PC's were included. A threshold was set based on the 95% quantile of the data, giving values of 289 for T^2 and 16.3 for Q. Two test data sets were then projected into the same PC space and data points with T^2 and Q values above the selected thresholds were identified.

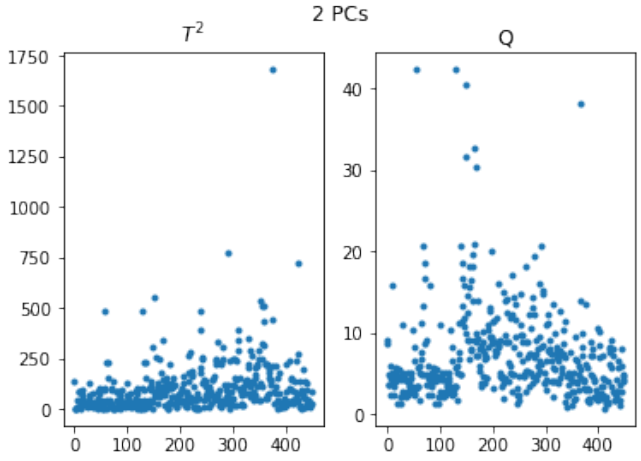


Fig. 1: Q and T^2 statistics for selected PCA model. After 2

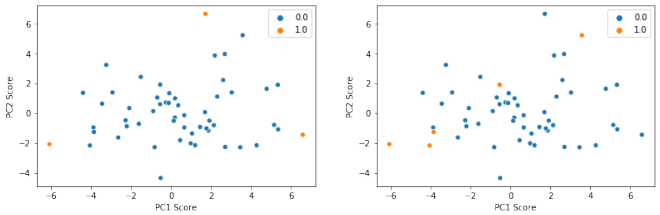
III. Results

T^2 and Q statistics for the training data are shown in figure 1. A model employing two principal components was selected, producing 95% quantile thresholds of 289 for T^2 and 16.3 for Q.

The same transformation and thresholds were applied to the test1 and test2 datasets. Scatterplots of the two principal component scores were plotted against each other. The test1 data (figure 2) essentially fit that model, with only a few values (identified in orange) exceeding the thresholds set.

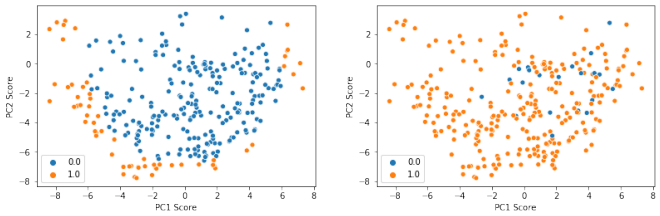
The contribution of each variable to the high T^2 score was calculated. In the case of the first outlier no unusual contribution was noted, meaning that it might be the result of normal statistical fluctuation. The second outlier had relatively large contributions from the 17 and 19th measurements. Finally the last outlier had a large contribution from the 5th measurement. These instruments should be examined for potential failure.

A similar analysis was applied to the Q statistics for the five outliers in that parameter. Few measurements stood out except in the case of the first outlying point which showed significant deviation in the 9th measurement and smaller (though still significant) deviation in the 19th. Again, these instruments and



(a) T^2 statistic (b) Q-statistic

Fig. 2: test1 data principal components plotted against each other. Only three scattered points exceed the T^2 threshold, while five exceed the Q threshold (identified in orange). This is consistent with the training dataset.



(a) T^2 statistic (b) Q-statistic

Fig. 3: test2 data principal components plotted against each other. Many of the points exceed the T^2 threshold; nearly all of them exceed the Q threshold.

portions of the process should be examined more closely to search for a fault.

The test2 data, on the other hand, deviated significantly from the model (figure 3). The Q-statistic value for nearly every one of the points exceeded the threshold and a large number of the points also exceeded their threshold. Extracting diagnostic information from these measurements would be difficult; the entire system should be examined.

IV. Conclusions

A PCA method for detecting anomalies in data was examined employing two statistical measures–Hotelling's T^2 and the Q-statistic. Thresholds were set for these statistics and were applied to two sets of data containing anomalies. In the first case, it was possible to extract information as to the source of anomalous readings, potentially assisting the diagnostic process. In the second case it was quickly apparent that the model no longer described the data and that the entire model should be re-examined from scratch.

V. Appendix

Python code used to perform calculations and generate graphics.

```
#!/usr/bin/env python
# coding: utf-8

# In[166]:


import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from utilities import train_test_val_split, rmse, load_matlab_data
from scipy.io import loadmat
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA


# In[167]:


matlab_data = loadmat("data/hwk8data.mat")
print(matlab_data.keys())


# In[168]:


data = matlab_data['data']
test1 = matlab_data['test1']
test2 = matlab_data['test2']


# In[169]:


print(data.shape)


# Briefly describe PCA and how it is used for anomaly detection. Describe the use of Q- and T2-

# In[170]:


def t2(data, pca):
    """Calculate the T^2 score for a data sample
    """
    l = np.diag(pca.explained_variance_)
    t = pca.transform(data)
    return np.array([tt @ l @ tt.T for tt in t])


# In[171]:


def q(data, pca):
```

```
    """Calculate Q-score for a data sample
    """
    # Get dimensionality of the data
    i = data.shape[1]

    pp = pca.components_.T @ pca.components_

    return np.array([x @ (np.eye(i) - pp) @ x.T for x in data])


# Develop a PCA model of the normal operating data contained in data.

# In[172]:


scale = StandardScaler().fit(data)


# In[173]:


data_s = scale.transform(data)


# In[174]:


pca = PCA().fit(data_s)


# In[175]:


test1_s = scale.transform(test1)
test2_s = scale.transform(test2)


# Determine on your own how many PCs should be included in the model.

# Calculate the number of components required to explain 95% of the variance

# In[176]:


np.where(np.cumsum(pca.explained_variance_ratio_) > 0.95)


# In[177]:


plt.plot(range(1, pca.explained_variance_.shape[0]+1), np.cumsum(pca.explained_variance_ratio_
plt.axhline(y=0.95, color='r', linestyle=':', alpha=0.3)
plt.axvline(x=7, color='r', linestyle=':', alpha=0.3)
plt.xlabel("Number of principal components")
plt.ylabel("Fraction of variance explained")


# In[178]:
```

```
plt.plot(pca.explained_variance_)
plt.axvline(x=4, color='r', linestyle=':', alpha=0.3)
plt.axvline(x=7, color='r', linestyle=':', alpha=0.3)


# In[179]:


np.sum(pca.explained_variance_ratio_[:5])


# Calculate the norm of the residuals as a percentage of the norm of the scaled training data

# In[180]:


e = []
for i in range(1, data.shape[1]):
    pc = PCA(n_components=i).fit(data_s)
    p = pc.inverse_transform(pc.transform(data_s))
    res = la.norm(data_s - p) / la.norm(data_s)
    e.append(np.mean(res))


# In[181]:


plt.plot(range(1, data.shape[1]), e)
plt.axhline(y=0.1, color='r', linestyle=':', alpha=0.3)
plt.axvline(x=13, color='r', linestyle=':', alpha=0.3)
plt.axvline(x=4, color='r', linestyle=':', alpha=0.3)
plt.axvline(x=7, color='r', linestyle=':', alpha=0.3)


# No clear elbow in this plot. 8 principal components gives ~25% variance from the model.

# Determine which components carry more than $\epsilon$ variance

# In[182]:


epsilon = 0.005


# In[183]:


np.where(pca.explained_variance_ratio_ > epsilon)


# Look at $T^2$ and Q values to select threshold values

# In[184]:


def plot_q_and_t2(data_s, n_components, t2_thresh=None, q_thresh=None):
```

```
    pc = PCA(n_components=n_components)
    pc.fit(data_s)
    qvals = q(data_s, pc)
    t2vals = t2(data_s, pc)
    f, axs= plt.subplots(1, 2)
    axs[0].plot(t2vals, '.')
    axs[0].set_title(r"$T^2$")
    if t2_thresh is not None:
        axs[0].axhline(y=t2_thresh, color='r',
                       alpha=0.3, linestyle=':')
    axs[1].plot(qvals, '.')
    axs[1].set_title(r"Q")
    if q_thresh is not None:
        axs[1].axhline(y=q_thresh, color='r',
                       alpha=0.3, linestyle=':')
    plt.suptitle(f"{n_components} PCs")
    plt.show()


# In[185]:


sns.heatmap(np.corrcoef(data_s, rowvar=False), square=True)


# From the correlation heat map it appears that the dimensionality of these data are at least

# In[186]:


for _ in range(1, data_s.shape[1]):
    plot_q_and_t2(data_s, _)


# $T^2$ settles down after 2 PC's. $Q$ (as expected) continues to lower as number of PC's incr

# In[187]:


pca = PCA(n_components=2)
pca.fit(data_s)


# Now set the $T^2$ and $Q$ thresholds at the 95% quantile

# In[188]:


t2_data = t2(data_s, pca)


# In[189]:


t2_thresh = np.quantile(t2_data, 0.95)


# In[190]:
```

```
q_data = q(data_s, pca)
```

```
# In[191]:
```

```
q_thresh = np.quantile(q_data, 0.95)
```

```
# In[192]:
```

```
t2_thresh
```

```
# In[193]:
```

```
q_thresh
```

```
# In[216]:
```

```
plot_q_and_t2(data_s, 2, t2_thresh=np.quantile(t2_data, 0.95),
              q_thresh=np.quantile(q_data, 0.95))
```

```
# Project the data in 'test1' and 'test2' into the PC space that you found in step 2. Evaluate
```

```
# In[195]:
```

```
t2_1 = t2(test1_s, pca)
```

```
# In[196]:
```

```
q_1 = q(test1_s, pca)
```

```
# In[197]:
```

```
t2_2 = t2(test2_s, pca)
```

```
# In[198]:
```

```
q_2 = q(test2_s, pca)
```

```
# In[199]:
```

```
hue_t1 = np.zeros_like(t2_1)
hue_t1[np.argwhere(t2_1 > t2_thresh)] = 1


# In[200]:


hue_t2 = np.zeros_like(t2_2)
hue_t2[np.argwhere(t2_2 > t2_thresh)] = 1


# In[201]:


hue_q1 = np.zeros_like(q_1)
hue_q1[np.argwhere(q_1 > q_thresh)] = 1


# In[202]:


hue_q2 = np.zeros_like(q_2)
hue_q2[np.argwhere(q_2 > q_thresh)] = 1


# In[218]:


sns.scatterplot(pca.transform(test1_s)[:,0],
                pca.transform(test1_s)[:,1],
                hue=hue_t1)
plt.xlabel("PC1 Score")
plt.ylabel("PC2 Score")


# In[220]:


sns.scatterplot(pca.transform(test1_s)[:,0],
                pca.transform(test1_s)[:,1],
                hue=hue_q1)
plt.xlabel("PC1 Score")
plt.ylabel("PC2 Score")


# In[221]:


sns.scatterplot(pca.transform(test2_s)[:,0],
                pca.transform(test2_s)[:,1],
                hue=hue_t2)
plt.xlabel("PC1 Score")
plt.ylabel("PC2 Score")


# In[222]:
```

```
sns.scatterplot(pca.transform(test2_s)[:,0],
                pca.transform(test2_s)[:,1],
                hue=hue_q2)
plt.xlabel("PC1 Score")
plt.ylabel("PC2 Score")


# In[208]:


def new_plot_q_and_t2(data_s, pc=pca, t2_thresh=None, q_thresh=None):
    qvals = q(data_s, pc)
    t2vals = t2(data_s, pc)
    f, axs= plt.subplots(1, 2)
    axs[0].plot(t2vals, '.')
    axs[0].set_title(r"$T^2$")
    if t2_thresh is not None:
        axs[0].axhline(y=t2_thresh, color='r',
                       alpha=0.3, linestyle=':')
    axs[1].plot(qvals, '.')
    axs[1].set_title(r"Q")
    if q_thresh is not None:
        axs[1].axhline(y=q_thresh, color='r',
                       alpha=0.3, linestyle=':')
    plt.show()


# In[209]:


new_plot_q_and_t2(test1_s, t2_thresh=t2_thresh,
              q_thresh=q_thresh)


# In[210]:


new_plot_q_and_t2(test2_s, t2_thresh=t2_thresh,
              q_thresh=q_thresh)


# From these scores it appears that the data in `test1` is consistent with the training data,

# In[211]:


plt.plot(q_2)
plt.axhline(y=q_thresh, color='r', linestyle=':', alpha=0.3)


# In[212]:


data.shape
```