Non Deep Learning Pump Model

This model uses a Support Vector Machine to classify the pump data in order to predict normal vs recovering/broken timestamps. The hidden code block following this markdown cell is where the data is read in from the object store and put into a DataFrame. The data in the DataFrame comes from the Feature Creation Deliverable Notebook. The second hidden cell is for the PCA data that comes from the second part of the Feature Creation Notebook. Both of the SVMs use GridSearchCV to tune the correct kernel and gamma settings. Predictions for both iterations are made using the best parameters from the grid search and the F1-score and Matthews Correlation Coefficient is calculated.

In [1]:

```
# The code was removed by Watson Studio for sharing.
```

Out[1]:

	Id	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	sensor_06	sensor_07	sensor_08	sensor_09	sensor_10	s
0	0	2018-04-01 00:00:00	0.231450	-0.151675	0.639386	1.057675	0.303443	0.177097	-0.042091	0.132586	0.181964	0.122858	-0.350860	
1	1	2018-04-01 00:01:00	0.231450	-0.151675	0.639386	1.057675	0.303443	0.177097	-0.042091	0.132586	0.181964	0.122858	-0.350860	
2	2	2018-04-01 00:02:00	0.180129	-0.072613	0.639386	1.093565	0.334786	0.008647	-0.082656	0.089329	0.207112	0.101892	-0.297906	
3	3	2018-04-01 00:03:00	0.219228	-0.151675	0.627550	1.093564	0.260045	0.207693	-0.086035	0.185835	0.246628	0.136839	-0.239029	
4	4	2018-04-01 00:04:00	0.182573	-0.138499	0.639386	1.093564	0.317909	0.184568	-0.069133	0.169195	0.246628	0.136839	-0.163810	

In [2]:

```
# The code was removed by Watson Studio for sharing.
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
0	-0.010524	0.776836	-0.522604	-0.782975	-1.943597	1.535913	-1.132301	-0.175648	-1.079528	-0.922840	-0.261418	-0.087582	0.276406	-0.1
1	-0.010524	0.776836	-0.522604	-0.782975	-1.943597	1.535913	-1.132301	-0.175648	-1.079528	-0.922840	-0.261418	-0.087582	0.276406	-0.1
2	-0.151425	0.782444	-0.493743	-0.843039	-2.105161	1.502969	-1.128054	-0.031836	-0.955613	-0.787217	-0.285881	-0.095448	0.120880	-0.1
3	-0.151886	0.816479	-0.541572	-0.971684	-2.012970	1.586361	-1.169678	-0.009723	-0.820260	-0.656993	-0.279351	-0.077121	0.195787	-0.2
4	-0.106250	0.929110	-0.407707	-1.025532	-1.989539	1.511101	-1.171104	-0.146129	-1.093221	-0.809584	-0.145487	-0.011419	0.105777	-0.1

In [3]:

```
#This is were all of the imports used in the final model are initiated
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
```

Split the Data

In order to prevent overfitting the data is split into train and test sets.

In [4]:

```
#The sensor data is split from the rest of the DataFrame as the features in the SVM Model
key_data = data.loc[:,'sensor_00': 'machine_status']
X = key_data.loc[:,'sensor_00': 'sensor_51']
y = key_data.machine_status

X_pca = data_pca.loc[:,'0':'24']
y_pca = data_pca.machine_status
```

In [5]:

```
#Train_test_split used to check the accuracy of the data and to check for any overfitting.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y_pca, test_size = 0.2)
```

In [6]:

```
#The hypertuning parameters are laid out here
kern = ['linear', 'poly', 'rbf', 'sigmoid']
gamma = ['scale', 'auto']

param = {'kernel': kern, 'gamma': gamma}
```

First Hyperparameter Tuning

The first tuning is trained using the 51 sensors from the water pump. The best parameters are printed out and the final scores are calculated.

In [7]:

```
#The GridSearch for hyperparameters is done here. The SVC instance is instantiated put into the GridSearchCV fi
svc = SVC()
clf = GridSearchCV(svc, param, verbose = 1)
clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed: 40.2min finished
```

Out[7]:

```
GridSearchCV(estimator=SVC(),
             param_grid={'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             verbose=1)
```

In [8]:

```
#Best hypertuning parameters
clf.best_params_
```

Out[8]:

```
{'gamma': 'scale', 'kernel': 'poly'}
```

In [9]:

```
#After the GridSearch is run the best model is predicted using the test data
y_pred = clf.predict(X_test)
```

In [10]:

```
#The best model is tested using f1_score
f1 = f1_score(y_test, y_pred)
phi = matthews_corrcoef(y_test, y_pred)
```

Second Hyperparameter Tuning

The second hyperparameter tuning is done on the PCA sensor data from the feature creation notebook. The best parameters are printed out and the final scores are calculated.

In [11]:

```
#The GridSearch for hyperparameters is done here. The SVC instance is instantiated put into the GridSearchCV fi
svc2 = SVC()
clf2 = GridSearchCV(svc2, param, verbose = 1)
clf2.fit(X_train_pca, y_train_pca)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed: 40.3min finished
```

Out[11]:

```
GridSearchCV(estimator=SVC(),
             param_grid={'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             verbose=1)
```

In [12]:

```
#Best hypertuning parameters
clf2.best_params_
```

Out[12]:

```
{'gamma': 'auto', 'kernel': 'poly'}
```

In [13]:

```
#After the GridSearch is run the best model is predicted using the test data
y_pred_pca = clf2.predict(X_test_pca)
```

In [14]:

```
#The best model is tested using f1_score and matthews correlation coefficient
f1_pca = f1_score(y_test_pca, y_pred_pca)
phi_pca = matthews_corrcoef(y_test_pca, y_pred_pca)
```

Final Score for Each Feature Creation Iteration

In [15]:

```
#Printouts of the score for the two types with standard data
print('Original Iteration')
print('F1 Score: {}'.format(f1))
print('Matthews Corr Coeficient: {}'.format(phi))

#Printouts of the score for the two types with PCA data
print('PCA Iteration')
print('F1 Score: {}'.format(f1_pca))
print('Matthews Corr Coeficient: {}'.format(phi_pca))
```

```
Original Iteration
F1 Score: 0.9999027639351434
Matthews Corr Coeficient: 0.9985364444899135
PCA Iteration
F1 Score: 0.9999269326318866
Matthews Corr Coeficient: 0.9989295017009503
```

Conclusion

The goal of this notebook was to test if a non deep learning model could accurately predict whether the pump is running normally or is in a broken or recovering state. The F1-score and the Matthews Correlation Coefficient shows that the goal was met. With both scores being above 0.99, the model seems to predict the pump accurately. The reason both scores are used in this case is that the F1-score is a well known accuracy measurement and the Matthews Correlation Coefficient also takes the true negatives into account. Taking true negatives into account is very important in this case as we want to be very accurate in predicting the broken state. The second iteration using PCA did not provide any benefit in this case. With further PCA reduction the model may have been able to train faster and be more accurate, but the current PCA was slower and about the same accuracy. To conclude, the model was accurate and did not take long to train, therefore our goal was met.