Assignment3

October 17, 2018

1 ASSIGNMENT 3 - PCA and Logistic Regression

Matthew Dunne

1. Data Processing:

a) Import the data: Only keep numeric data (pandas has tools to do this!). Drop "PHONE" and "COUNTRY_SSA" as well.

```
In [1]: import numpy as np
        import os
        import pandas as pd
        data=pd.read_csv('ProviderInfo.csv')
        data=data.select_dtypes(include=['float64']).drop(['PHONE', 'COUNTY_SSA'], axis=1)
        data.head()
```

Out[1]:

	ZIP	BEDCERT	RESTOT	OVERALL_RATING	SURVEY_RATING	QUALITY_RATING	\
0	35653.0	57.0	51.5	5.0	5.0	5.0	
1	35150.0	85.0	74.2	3.0	3.0	5.0	
2	35768.0	50.0	NaN	1.0	2.0	2.0	
3	35206.0	92.0	79.8	2.0	2.0	4.0	
4	35111.0	103.0	98.1	3.0	3.0	4.0	

	STAFFING_RATING	RN_STAFFING_RATING	AIDHRD	VOCHRD	...	\
0	4.0	4.0	3.43572	1.16495	...	
1	1.0	1.0	NaN	NaN	...	
2	1.0	1.0	NaN	NaN	...	
3	3.0	3.0	2.32722	0.82104	...	
4	3.0	2.0	2.33617	0.92407	...	

	ADJ_AIDE	ADJ_LPN	ADJ_RN	ADJ_TOTAL	INCIDENT_CNT	CMPLNT_CNT	FINE_CNT	\
0	3.11741	1.24750	0.83853	5.13047	0.0	0.0	0.0	
1	NaN	NaN	NaN	NaN	0.0	0.0	1.0	
2	NaN	NaN	NaN	NaN	0.0	0.0	0.0	
3	2.40074	0.86962	0.56463	3.83026	0.0	1.0	0.0	
4	2.55126	1.08955	0.30360	3.95709	0.0	0.0	0.0	

	FINE_TOT	PAYDEN_CNT	TOT_PENLTY_CNT
0	0.0	0.0	0.0

```
1   15259.0         1.0             2.0
2       0.0         0.0             0.0
3       0.0         0.0             0.0
4       0.0         0.0             0.0

[5 rows x 28 columns]
```

b) This data is extra messy and has some NaN and NaT values. NaT values should be replaced by "np.nan." After this step, remove any rows that have an NaN value.

```
In [2]: data.replace(["NaN", 'NaT'], np.nan, inplace = True)
        clean_df= data.dropna(how='any', axis = 0)
        clean_df.head()
```

```
Out[2]:        ZIP  BEDCERT  RESTOT  OVERALL_RATING  SURVEY_RATING  QUALITY_RATING  \
        0  35653.0     57.0    51.5             5.0            5.0             5.0
        3  35206.0     92.0    79.8             2.0            2.0             4.0
        4  35111.0    103.0    98.1             3.0            3.0             4.0
        5  35611.0    149.0   119.7             5.0            3.0             5.0
        6  36025.0    124.0    96.0             5.0            4.0             5.0

           STAFFING_RATING  RN_STAFFING_RATING   AIDHRD   VOCHRD       ...        \
        0              4.0                 4.0  3.43572  1.16495       ...
        3              3.0                 3.0  2.32722  0.82104       ...
        4              3.0                 2.0  2.33617  0.92407       ...
        5              4.0                 3.0  2.57869  1.01443       ...
        6              3.0                 4.0  1.99985  0.62768       ...

           ADJ_AIDE  ADJ_LPN   ADJ_RN  ADJ_TOTAL  INCIDENT_CNT  CMPLNT_CNT  FINE_CNT  \
        0   3.11741  1.24750  0.83853    5.13047           0.0         0.0       0.0
        3   2.40074  0.86962  0.56463    3.83026           0.0         1.0       0.0
        4   2.55126  1.08955  0.30360    3.95709           0.0         0.0       0.0
        5   2.56783  1.04823  0.46444    4.07866           0.0         1.0       0.0
        6   2.12102  0.70311  0.75448    3.52979           1.0         1.0       0.0

           FINE_TOT  PAYDEN_CNT  TOT_PENLTY_CNT
        0       0.0         0.0             0.0
        3       0.0         0.0             0.0
        4       0.0         0.0             0.0
        5       0.0         0.0             0.0
        6       0.0         0.0             0.0

        [5 rows x 28 columns]
```

c) Split into train / test set using an 80/20 split.

```
In [3]: from sklearn.model_selection import train_test_split
        Y=clean_df['OVERALL_RATING']
        X=clean_df.drop(['OVERALL_RATING'], axis=1)
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=.20, random_state=
```

d) Scale all input features (NOT THE TARGET VARIABLE)

```
In [4]: from sklearn import preprocessing
        scaler=preprocessing.StandardScaler()
        X_train_scaled=scaler.fit_transform(X_train)
        X_test_scaled=scaler.transform(X_test)
```

2. Model #1: Logistic Regression

a) Pick up from step d in Problem 1 (use the same data that has been scaled): Using LogisticRegression(), build a model to predict the "OVERALL_RATING". Note: The default in sklearn is "one-vs-rest" classification, where we calculate the probability of each class compared to the rest. This is fine for the homework!

```
In [5]: from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score
        logreg = LogisticRegression()
        logreg_model=logreg.fit(X_train_scaled, y_train)
        y_pred=logreg_model.predict(X_test_scaled)
```

b) For error evaluation, start by calculating the score (returns the mean accuracy).

```
In [6]: accuracy_score(y_pred, y_test)
```

```
Out[6]: 0.71359890109890112
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

```
In [7]: from sklearn.metrics import confusion_matrix, classification_report
        print('Confusion Matrix\n')
        print(confusion_matrix(y_pred, y_test))
        print('\nClassification Report\n')
        print(classification_report(y_pred, y_test))
```

```
Confusion Matrix

[[276  65   0   0   0]
 [ 66 434 173  96   0]
 [  0  71  66  10   0]
 [  0   6 201 455  22]
 [  0   0   0 124 847]]

Classification Report

             precision    recall  f1-score   support

        1.0       0.81      0.81      0.81       341
        2.0       0.75      0.56      0.65       769
        3.0       0.15      0.45      0.22       147
```

```
        4.0       0.66      0.67      0.66       684
        5.0       0.97      0.87      0.92       971

avg / total       0.78      0.71      0.74      2912
```

3. Model #2: PCA(n_components = 2) + Logistic Regression

a) Pick up from step d in Problem 1 (use the same data that has been scaled): We will now transform the X_train & X_test data using PCA with 2 components.

```
In [8]: from sklearn.decomposition import PCA
        pca2 = PCA(n_components = 2)
        X_train_pc2 = pca2.fit_transform(X_train_scaled)
        #do not to separate fit for PCA on test
        X_test_pc2=pca2.transform(X_test_scaled)
```

b) Then use the transformed data (X_train_pca) to fit a Logistic Regression model.

```
In [9]: logreg_model_pc2=logreg.fit(X_train_pc2, y_train)
        y_pred_pc2=logreg_model_pc2.predict(X_test_pc2)
```

c) Calculate the same error metrics as those from Model #1.

```
In [10]: print('Confusion Matrix For PCA=2 Logit Model\n')
         print(confusion_matrix(y_pred_pc2, y_test))
         print('\nClassification Report For PCA=2 Logit Model\n')
         print(classification_report(y_pred_pc2, y_test))
```

```
Confusion Matrix For PCA=2 Logit Model

[[132 108  23  28   9]
 [165 238 187 155 103]
 [  1   0   0   0   0]
 [ 10  28  16  22  28]
 [ 34 202 214 480 729]]

Classification Report For PCA=2 Logit Model

             precision    recall  f1-score   support

        1.0       0.39      0.44      0.41       300
        2.0       0.41      0.28      0.33       848
        3.0       0.00      0.00      0.00         1
        4.0       0.03      0.21      0.06       104
        5.0       0.84      0.44      0.58      1659

avg / total       0.64      0.38      0.47      2912
```

4. Model #3: PCA(n_components = 16) + Logistic Regression

a) Pick up from step d in Problem 1 (use the same data that has been scaled): We will now transform the X_train & X_test data using PCA with 16 components.

```
In [11]: pca16 = PCA(n_components = 16)
         X_train_pc16 = pca16.fit_transform(X_train_scaled)
         #do not to separate fit for PCA on test
         X_test_pc16=pca16.transform(X_test_scaled)
```

b) Then use the transformed data (X_train_pca) to fit a Logistic Regression model.

```
In [12]: logreg_model_pc16=logreg.fit(X_train_pc16, y_train)
         y_pred_pc16=logreg_model_pc16.predict(X_test_pc16)
```

c) Calculate the same error metrics as those from Model #1.

```
In [13]: print('Confusion Matrix For PCA=16 Logit Model\n')
         print(confusion_matrix(y_pred_pc16, y_test))
         print('\nClassification Report For PCA=16 Logit Model\n')
         print(classification_report(y_pred_pc16, y_test))
```

```
Confusion Matrix For PCA=16 Logit Model

[[278  66   0   0   0]
 [ 63 436 174  96   0]
 [  1  65  69   2   0]
 [  0   9 197 461  19]
 [  0   0   0 126 850]]

Classification Report For PCA=16 Logit Model

             precision    recall  f1-score   support

        1.0       0.81      0.81      0.81       344
        2.0       0.76      0.57      0.65       769
        3.0       0.16      0.50      0.24       137
        4.0       0.67      0.67      0.67       686
        5.0       0.98      0.87      0.92       976

avg / total       0.79      0.72      0.75      2912
```

5. Between Model #2 and Model #3, which performed the best?

Model #3 (with sixteen Principal Components) performed substantially better than Model #2 (with two Principal Components). Recall (percentage of 1's that were called 1's, 2's that were called 2's etc.) was 72% on average. Precision (percentage of what was called 1's were 1's, 2's were 2's, etc) was 79%. An improvement of 34% and 15% respectively from Model #2. Interestingly, Model #3 performed only slightly better than a standard logistic regression.