Stanly Gomes
Student ID: 801118166

HW3

October 25, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

[2]:
```
breast = load_breast_cancer()
```

[3]:
```
breast_data = breast.data
breast_data.shape
```

[3]:
```
(569, 30)
```

[4]:
```
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

[4]:
```
       0      1       2       3        4        5       6        7       8   \
0  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419
1  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812
2  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069
3  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597
4  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809

        9   …     20     21      22      23      24      25      26      27  \
0  0.07871  …  25.38  17.33  184.60  2019.0  0.1622  0.6656  0.7119  0.2654
1  0.05667  …  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860
2  0.05999  …  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430
3  0.09744  …  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575
4  0.05883  …  22.54  16.67  152.20  1575.0  0.1374  0.2050  0.4000  0.1625

       28       29
0  0.4601  0.11890
1  0.2750  0.08902
```

```
2  0.3613  0.08758
3  0.6638  0.17300
4  0.2364  0.07678

[5 rows x 30 columns]
```

[5]:
```
breast_labels = breast.target
breast_labels.shape
```

[5]:
```
(569,)
```

[6]:
```
labels = np.reshape(breast_labels,(569,1))
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

[6]:
```
(569, 31)
```

[7]:
```
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features
```

[7]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

[8]:
```
features_labels = np.append(features,'Type')
breast_dataset.columns = features_labels
breast_dataset.head()
```

[8]:
```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
```

```
4           0.13280          0.1980              0.10430          0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  Type
0          0.4601                  0.11890   0.0
1          0.2750                  0.08902   0.0
2          0.3613                  0.08758   0.0
3          0.6638                  0.17300   0.0
4          0.2364                  0.07678   0.0

[5 rows x 31 columns]
```

[9]:
```python
breast_dataset['Type'].replace(0, 'Benign',inplace=True)
breast_dataset['Type'].replace(1, 'Malignant',inplace=True)
```

[10]:
```python
breast_dataset.tail()
```

[10]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
```

```
566                0.05648  …       34.12          126.70      1124.0
567                0.07016  …       39.42          184.60      1821.0
568                0.05884  …       30.37           59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension       Type
564                0.2216          0.2060                  0.07115     Benign
565                0.1628          0.2572                  0.06637     Benign
566                0.1418          0.2218                  0.07820     Benign
567                0.2650          0.4087                  0.12400     Benign
568                0.0000          0.2871                  0.07039  Malignant

[5 rows x 31 columns]
```

[11]:
```python
# 80% Training split
from sklearn.preprocessing import StandardScaler
y = breast_dataset.loc[:,['Type']].values
x = breast_dataset.loc[:, features].values
x = StandardScaler().fit_transform(x)
X_train, X_test, Y_train, Y_test = train_test_split(x, y, train_size=0.8,␣
↪test_size=0.2)
y[:10]
```

[11]:
```
array([['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign'],
       ['Benign']], dtype=object)
```

[12]:
```python
#Problem 1
```

[13]:
```python
model = GaussianNB()
```

[14]:
```python
model.fit(X_train, Y_train.ravel())
print(model)
```

```
GaussianNB()
```

[15]:
```python
expected = Y_test
predicted = model.predict(X_test)
```

[16]:
```python
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
              precision    recall  f1-score   support

      Benign       0.88      0.88      0.88        33
   Malignant       0.95      0.95      0.95        81

    accuracy                           0.93       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.93      0.93      0.93       114

[[29  4]
 [ 4 77]]
```

[17]:
```python
# Compare with Logistic Regression
classifier = LogisticRegression(penalty='none',random_state=0)
```

[18]:
```python
classifier.fit(X_train, Y_train.ravel())
```

[18]:
```
LogisticRegression(penalty='none', random_state=0)
```

[19]:
```python
Y_pred = classifier.predict(X_test)
```

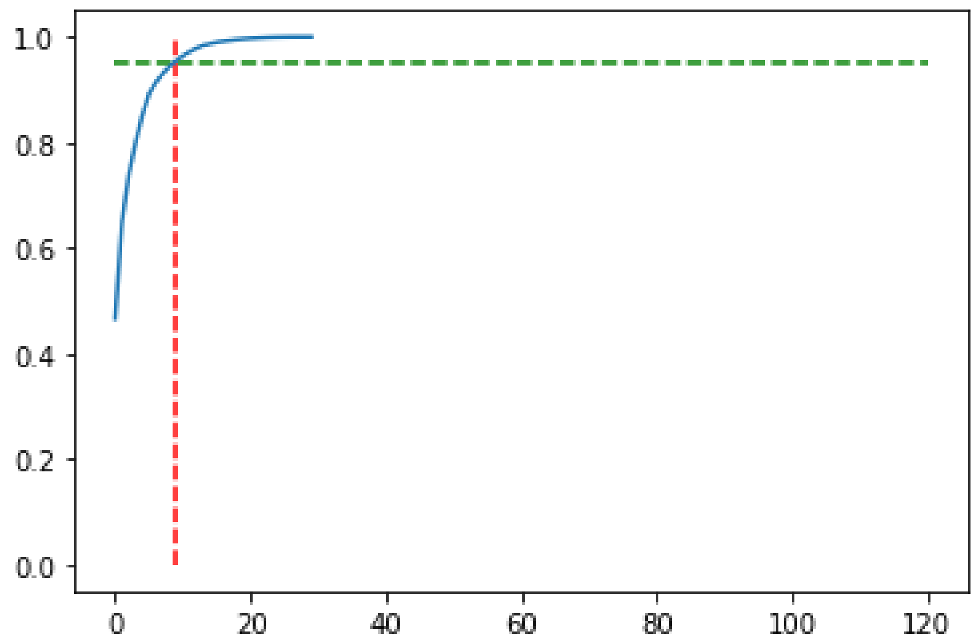
[20]:
```python
print(metrics.classification_report(expected, Y_pred))
print(metrics.confusion_matrix(expected, Y_pred))
```

```
              precision    recall  f1-score   support

      Benign       0.91      0.94      0.93        33
   Malignant       0.97      0.96      0.97        81

    accuracy                           0.96       114
   macro avg       0.94      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

[[31  2]
 [ 3 78]]
```

[21]:
```python
# Problem 2

#Decide the number of PCA components
from sklearn.decomposition import PCA
pca = PCA(random_state=88)
pca.fit(X_train)
```

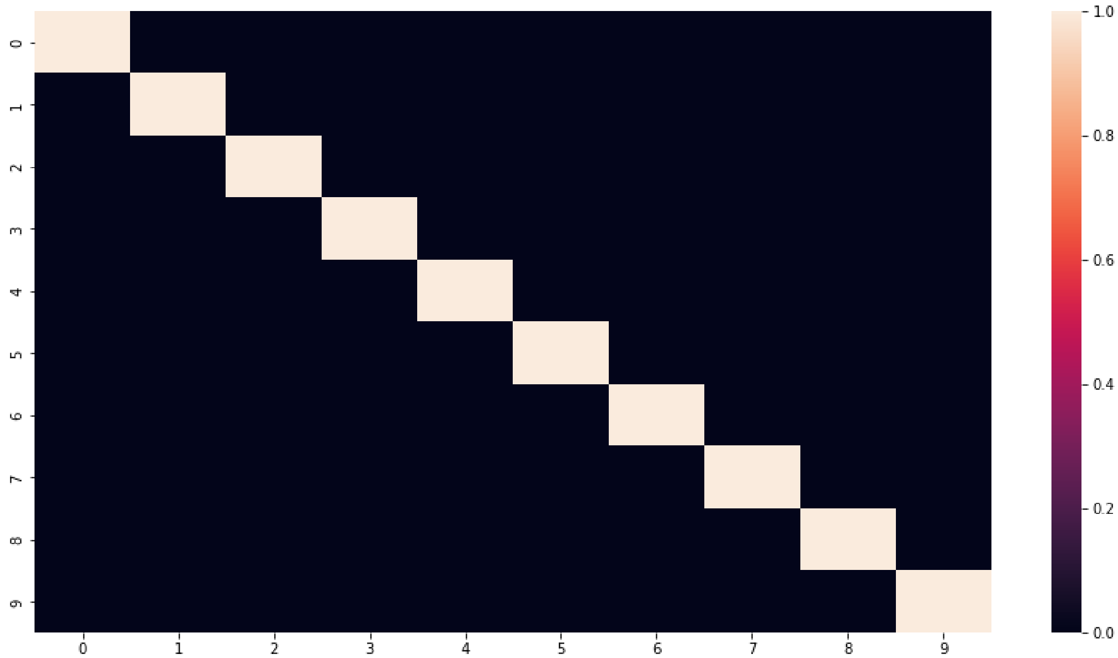
```
explained_variance = np.cumsum(pca.explained_variance_ratio_)
plt.vlines(x=9, ymax=1, ymin=0, colors='r', linestyle="--")
plt.hlines(y=0.95, xmax=120, xmin=0, colors="g", linestyles="--")
plt.plot(explained_variance)
#principalComponents = pca.fit_transform(x)
#principalDf = pd.DataFrame(data=principalComponents
#                           , columns=['principal component 1', 'principal␣
↪component 2'])
```

[21]: [<matplotlib.lines.Line2D at 0x2096357b580>]



[22]:
```
# Train PCA model
pca_final = PCA(0.95)
df_train_pca = pca_final.fit_transform(X_train)
```

[23]:
```
# Correlations between components
corr_mat = np.corrcoef(df_train_pca.transpose())
plt.figure(figsize=[15,8])
sns.heatmap(corr_mat)
plt.show()
```



[24]:
```
# Apply PCA model to test data
df_test_pca = pca_final.transform(X_test)
# Train Logistic Regression
LR = LogisticRegression()
LR_model = LR.fit(df_train_pca, Y_train.ravel())
```

[25]:
```
# Model Evaluation
Y_pred_pca = LR_model.predict(df_test_pca)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_pred_pca)
cnf_matrix
```

[25]:
```
array([[31,  2],
       [ 2, 79]], dtype=int64)
```

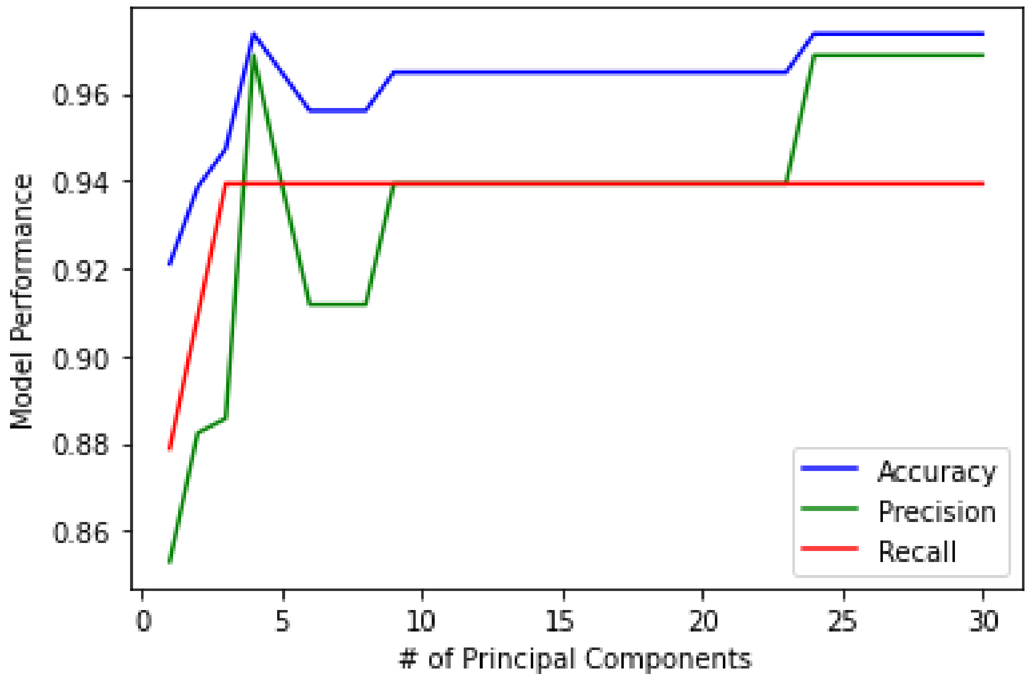
[26]:
```
# Evaluate accuracy, precision, and recall
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred_pca))
print("Precision:",metrics.precision_score(Y_test, Y_pred_pca,␣
↪pos_label='Benign'))
print("Recall:",metrics.recall_score(Y_test, Y_pred_pca, pos_label='Benign'))
Accuracy = []
Precision = []
Recall = []
k = range(1,len(features)+1)
```

```
Accuracy: 0.9649122807017544
Precision: 0.9393939393939394
```

```
Recall: 0.9393939393939394
```

[27]:
```
# Plot Model Performance vs # of Principal Components
for i in k:
    pca_final = PCA(n_components=i)
    df_train_pca = pca_final.fit_transform(X_train)
    df_test_pca = pca_final.transform(X_test)
    LR = LogisticRegression()
    LR_model = LR.fit(df_train_pca, Y_train.ravel())
    Y_pred_pca = LR_model.predict(df_test_pca)
    #print("K = ", i)
    #print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred_pca))
    #print("Precision:",metrics.precision_score(Y_test, Y_pred_pca,␣
 ↪pos_label='Benign'))
    #print("Recall:",metrics.recall_score(Y_test, Y_pred_pca,␣
 ↪pos_label='Benign'))
    Accuracy.append(metrics.accuracy_score(Y_test, Y_pred_pca))
    Precision.append(metrics.precision_score(Y_test, Y_pred_pca,␣
 ↪pos_label='Benign'))
    Recall.append(metrics.recall_score(Y_test, Y_pred_pca, pos_label='Benign'))
```

[28]:
```
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision, 'g', label="Precision")
plt.plot(k, Recall, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.legend()
plt.show()
```



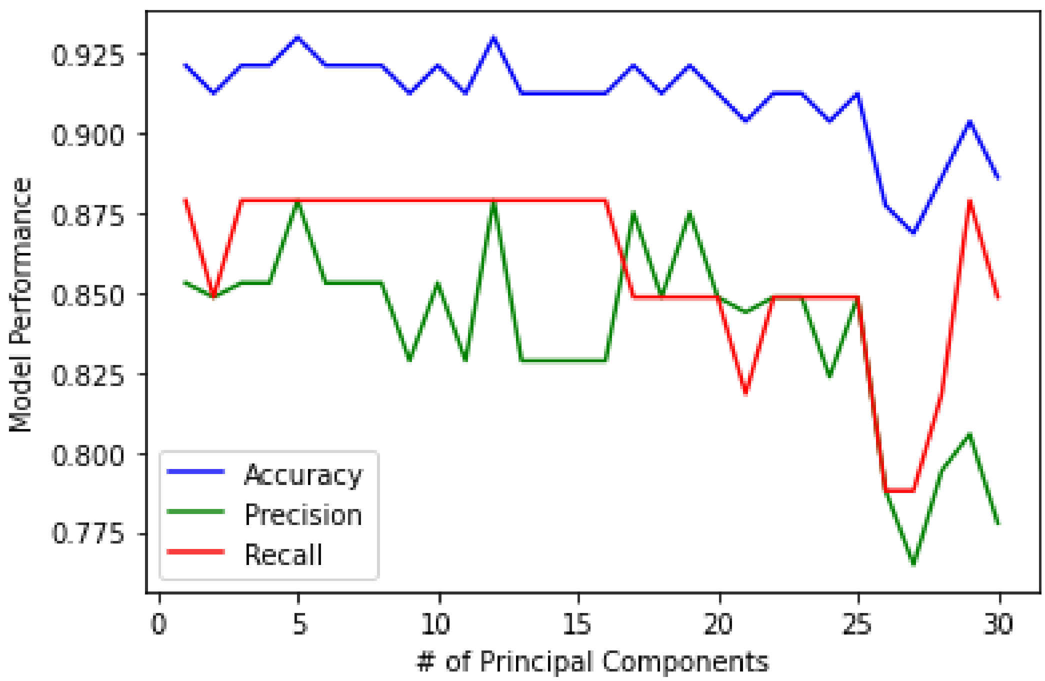
[29]:
```
# Problem 3
```

[30]:
```
model = GaussianNB()
model.fit(df_train_pca, Y_train.ravel())
k = range(1,len(features)+1)
Accuracy_NB = []
Precision_NB = []
Recall_NB = []
```

[31]:
```
for i in k:
    pca_final = PCA(n_components=i)
    df_train_pca = pca_final.fit_transform(X_train)
    df_test_pca = pca_final.transform(X_test)
    model = GaussianNB()
    model.fit(df_train_pca, Y_train.ravel())
    Y_pred_pca = model.predict(df_test_pca)
    #DEBUG print("K = ", i)
    #DEBUG print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred_pca))
    #DEBUG print("Precision:",metrics.precision_score(Y_test, Y_pred_pca,␣
 ↪pos_label='Benign'))
    #DEBUG print("Recall:",metrics.recall_score(Y_test, Y_pred_pca,␣
 ↪pos_label='Benign'))
    Accuracy_NB.append(metrics.accuracy_score(Y_test, Y_pred_pca))
```

```
    Precision_NB.append(metrics.precision_score(Y_test, Y_pred_pca,␣
↪pos_label='Benign'))
    Recall_NB.append(metrics.recall_score(Y_test, Y_pred_pca,␣
↪pos_label='Benign'))
```

[32]:
```
plt.plot(k, Accuracy_NB, 'b', label="Accuracy")
plt.plot(k, Precision_NB, 'g', label="Precision")
plt.plot(k, Recall_NB, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.legend()
plt.show()
```



[ ]:
```
```