Problem_1

November 3, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.decomposition import PCA
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[2]:
```
breast = load_breast_cancer()
breast_data = breast.data
breast_data.shape
```

[2]:
```
(569, 30)
```

[3]:
```
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

[3]:
```
       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]
```

[4]:
```python
breast_labels = breast.target
breast_labels.shape
```

[4]:
```
(569,)
```

[5]:
```python
labels = np.reshape(breast_labels,(569,1))
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

[5]:
```
(569, 31)
```

[6]:
```python
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features
```

[6]:
```
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')
```

[7]:
```python
features_labels = np.append(features,'label')
breast_dataset.columns = features_labels
breast_dataset.head()
```

[7]:
```
   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  label
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]
```

[8]:
```
X = breast_dataset.values[:,0:30]
print('X =', X[0:5])
Y = breast_dataset.values[:,30]
print('Y =', Y[0:5])
```

```
X = [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
```

```
 1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
 9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
 2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
 6.638e-01 1.730e-01]
[2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
 1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
 1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
 1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
 2.364e-01 7.678e-02]]
Y = [0. 0. 0. 0. 0.]
```

[9]:

```
# Using SVM Classifier with the linear kernal at different principal components␣
↪and storing them to be plotted later.

Prin_Comp = []
Precision_0 = []
Precision_1 = []
Recall_0 = []
Recall_1 = []
Accuracy = []
max_accuracy = 0
high_comp = 0

N_Comp = range(1,30)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,␣
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model = SVC(kernel = 'linear', C = 0.01)
    model.fit(X_train_sc, Y_train)
    predicted = model.predict(X_test_sc)
    matrix = confusion_matrix(Y_test, predicted)
    report = classification_report(Y_test, predicted)
    report_data = classification_report(Y_test, predicted, output_dict=True)
    data = pd.DataFrame(report_data)
    Precision_0.append(data.values[0,0])
    Precision_1.append(data.values[0,1])
    Recall_0.append(data.values[1,0])
    Recall_1.append(data.values[1,1])
    Accuracy.append(data.values[0,2])
    if data.values[0,2] > max_accuracy:
```

```
        high_comp = N
        max_accuracy = data.values[0,2]

# Latest Confusion Matrix and Classification Report
print("Confusion Matrix: \n",matrix)
print("\n")
print("Classification Report: \n", report)
```

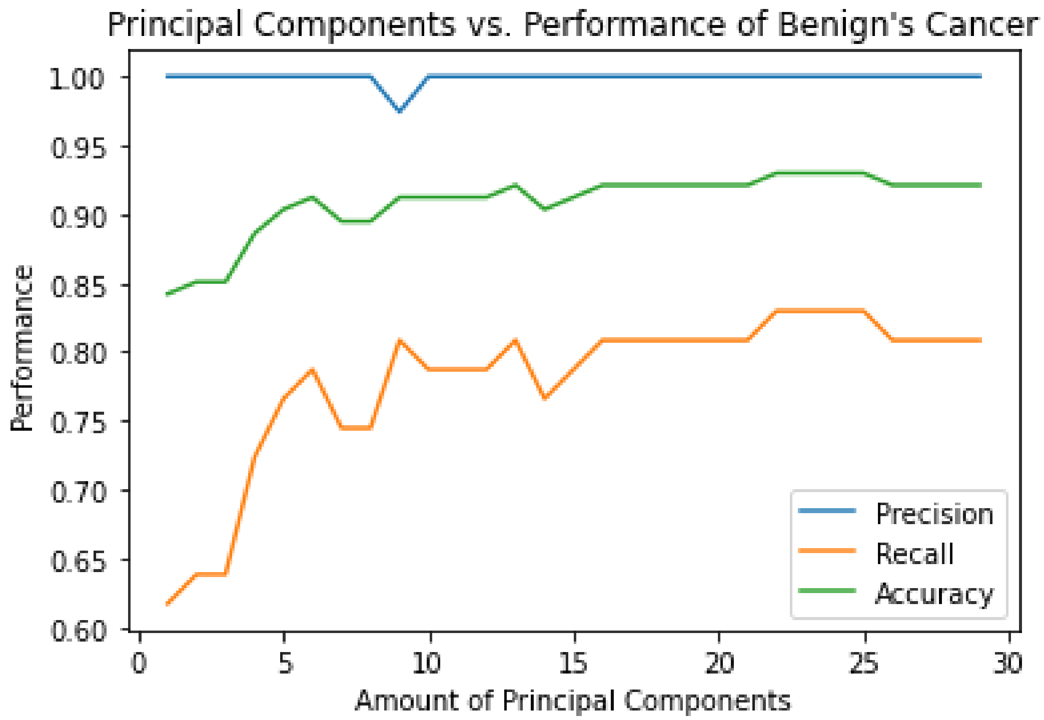
```
Confusion Matrix:
 [[38  9]
 [ 0 67]]


Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      0.81      0.89        47
         1.0       0.88      1.00      0.94        67

    accuracy                           0.92       114
   macro avg       0.94      0.90      0.92       114
weighted avg       0.93      0.92      0.92       114
```

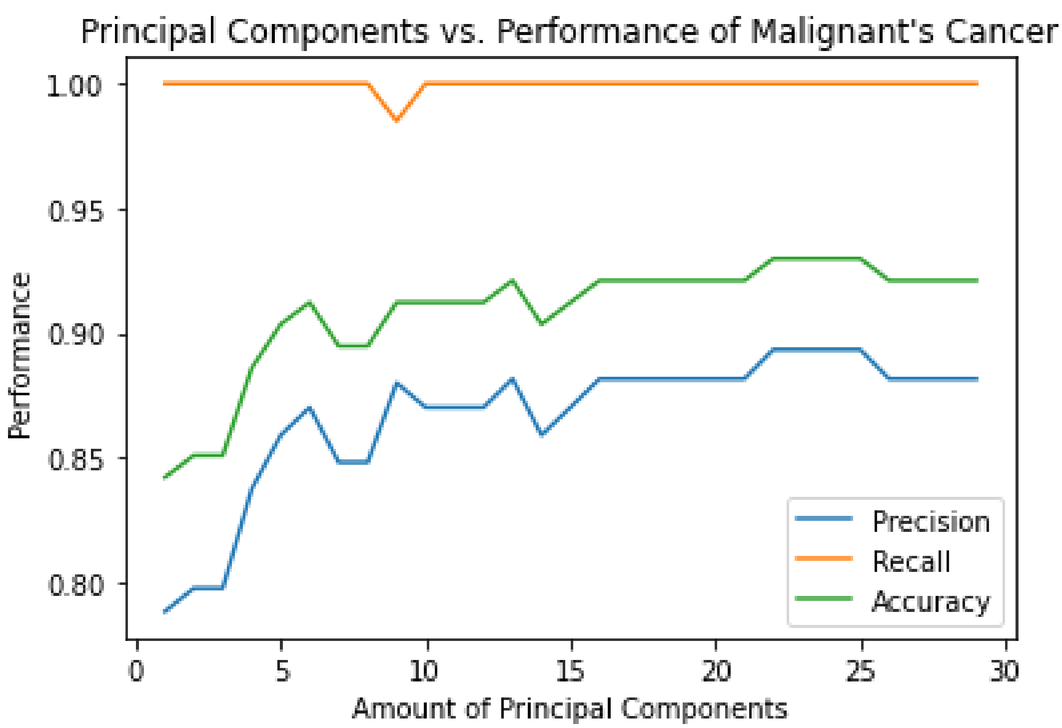
[10]:
```
print("The N of principal components with the highest accuracy is:", high_comp,␣
  ↪"components with the accuracy of", max_accuracy)
```

```
The N of principal components with the highest accuracy is: 22 components with
the accuracy of 0.9298245614035088
```

[11]:
```
print("Using Linear Kernal:")
plt.plot(Prin_Comp, Precision_0, label = "Precision")
plt.plot(Prin_Comp, Recall_0, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Benign's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
plt.plot(Prin_Comp, Precision_1, label = "Precision")
plt.plot(Prin_Comp, Recall_1, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Malignant's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
```

```
Using Linear Kernal:
```


Principal Components vs. Performance of Benign's Cancer
Performance
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25
30
Amount of Principal Components
Precision
Recall
Accuracy


Principal Components vs. Performance of Malignant's Cancer
Performance
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30
Amount of Principal Components
Precision
Recall
Accuracy

[12]:
```
# Using SVM Classifier with the poly kernal at different principal components␣
↪and storing them to be plotted later.

Prin_Comp = []
Precision_0 = []
Precision_1 = []
Recall_0 = []
Recall_1 = []
Accuracy = []
max_accuracy = 0
high_comp = 0

N_Comp = range(1,30)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,␣
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model = SVC(kernel = 'poly', C = 0.1, degree = 15)
    model.fit(X_train_sc, Y_train)
    predicted = model.predict(X_test_sc)
    matrix = confusion_matrix(Y_test, predicted)
    report = classification_report(Y_test, predicted)
    report_data = classification_report(Y_test, predicted, output_dict=True)
    data = pd.DataFrame(report_data)
    Precision_0.append(data.values[0,0])
    Precision_1.append(data.values[0,1])
    Recall_0.append(data.values[1,0])
    Recall_1.append(data.values[1,1])
    Accuracy.append(data.values[0,2])
    if data.values[0,2] > max_accuracy:
        high_comp = N
        max_accuracy = data.values[0,2]

# Latest Confusion Matrix and Classification Report
print("Confusion Matrix: \n",matrix)
print("\n")
print("Classification Report: \n", report)
```

```
Confusion Matrix:
 [[ 1 46]
 [ 0 67]]
```

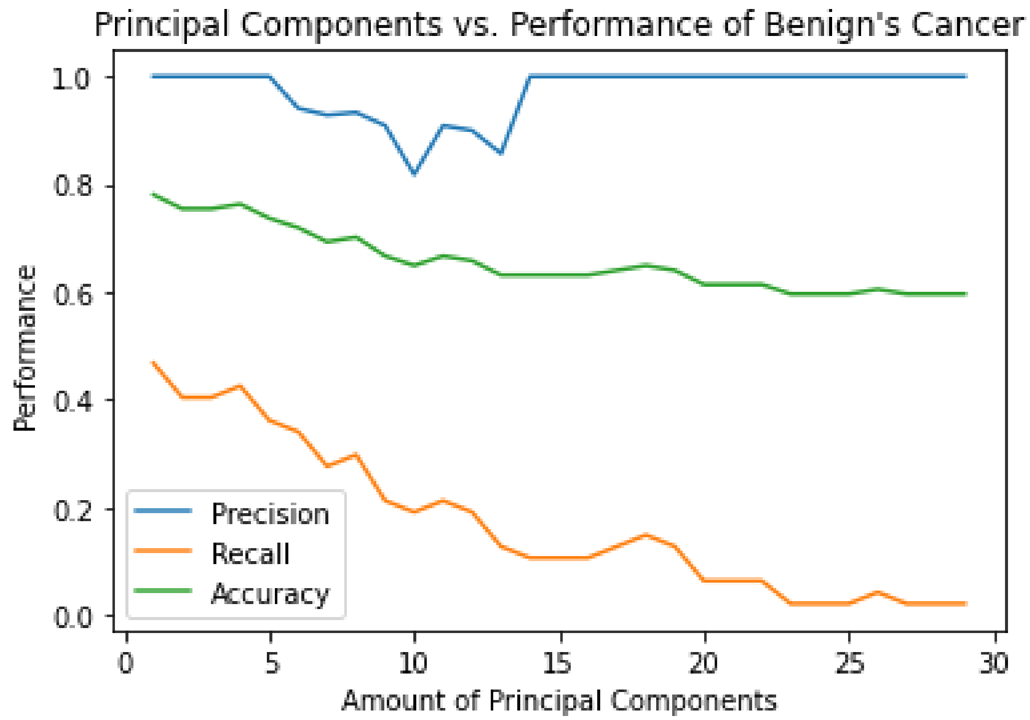
```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.02      0.04        47
         1.0       0.59      1.00      0.74        67

    accuracy                           0.60       114
   macro avg       0.80      0.51      0.39       114
weighted avg       0.76      0.60      0.45       114
```

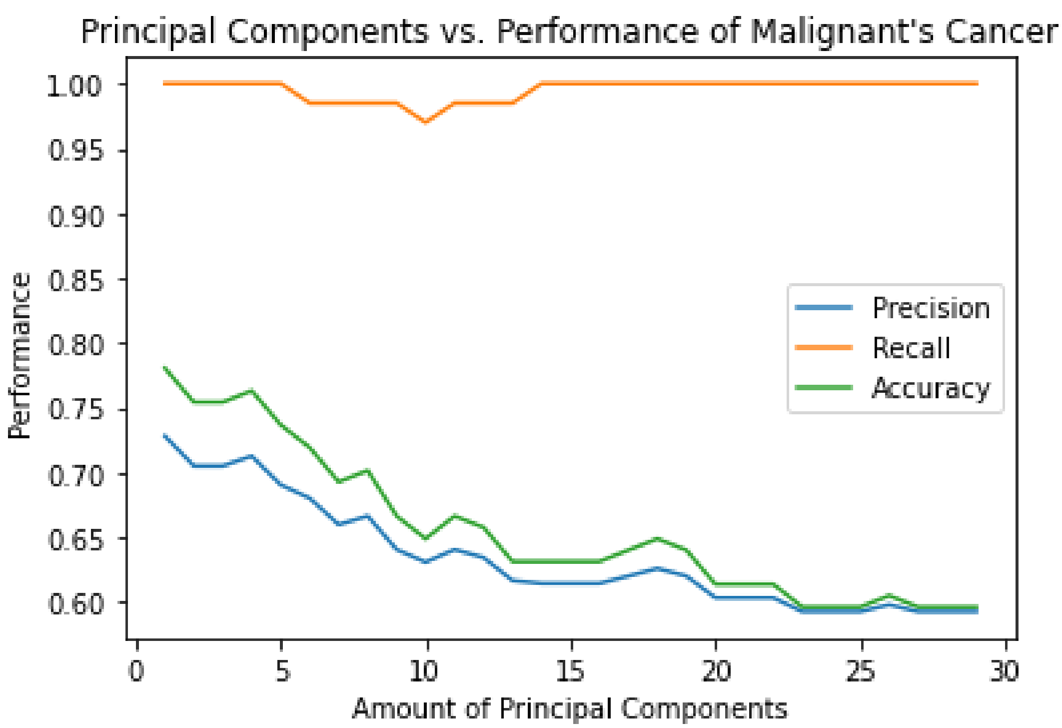
[13]:
```python
print("The N of principal components with the highest accuracy is:", high_comp,␣
↪"components with the accuracy of", max_accuracy)
```

```
The N of principal components with the highest accuracy is: 1 components with
the accuracy of 0.7807017543859649
```

[14]:
```python
print("Using Polynomial Kernal:")
plt.plot(Prin_Comp, Precision_0, label = "Precision")
plt.plot(Prin_Comp, Recall_0, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Benign's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
plt.plot(Prin_Comp, Precision_1, label = "Precision")
plt.plot(Prin_Comp, Recall_1, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Malignant's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
```

```
Using Polynomial Kernal:
```


Principal Components vs. Performance of Benign's Cancer
Performance
Amount of Principal Components
Precision
Recall
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30


Principal Components vs. Performance of Malignant's Cancer
Performance
Amount of Principal Components
Precision
Recall
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25
30

[15]:
```
# Using SVM Classifier with the rbf kernal at different principal components
↪and storing them to be plotted later.

Prin_Comp = []
Precision_0 = []
Precision_1 = []
Recall_0 = []
Recall_1 = []
Accuracy = []
max_accuracy = 0
high_comp = 0

N_Comp = range(1,30)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model = SVC(kernel = 'rbf', C = 10, gamma = 0.01)
    model.fit(X_train_sc, Y_train)
    predicted = model.predict(X_test_sc)
    matrix = confusion_matrix(Y_test, predicted)
    report = classification_report(Y_test, predicted)
    report_data = classification_report(Y_test, predicted, output_dict=True)
    data = pd.DataFrame(report_data)
    Precision_0.append(data.values[0,0])
    Precision_1.append(data.values[0,1])
    Recall_0.append(data.values[1,0])
    Recall_1.append(data.values[1,1])
    Accuracy.append(data.values[0,2])
    if data.values[0,2] > max_accuracy:
        high_comp = N
        max_accuracy = data.values[0,2]

# Latest Confusion Matrix and Classification Report
print("Confusion Matrix: \n",matrix)
print("\n")
print("Classification Report: \n", report)
```

```
Confusion Matrix:
 [[44  3]
 [ 0 67]]
```

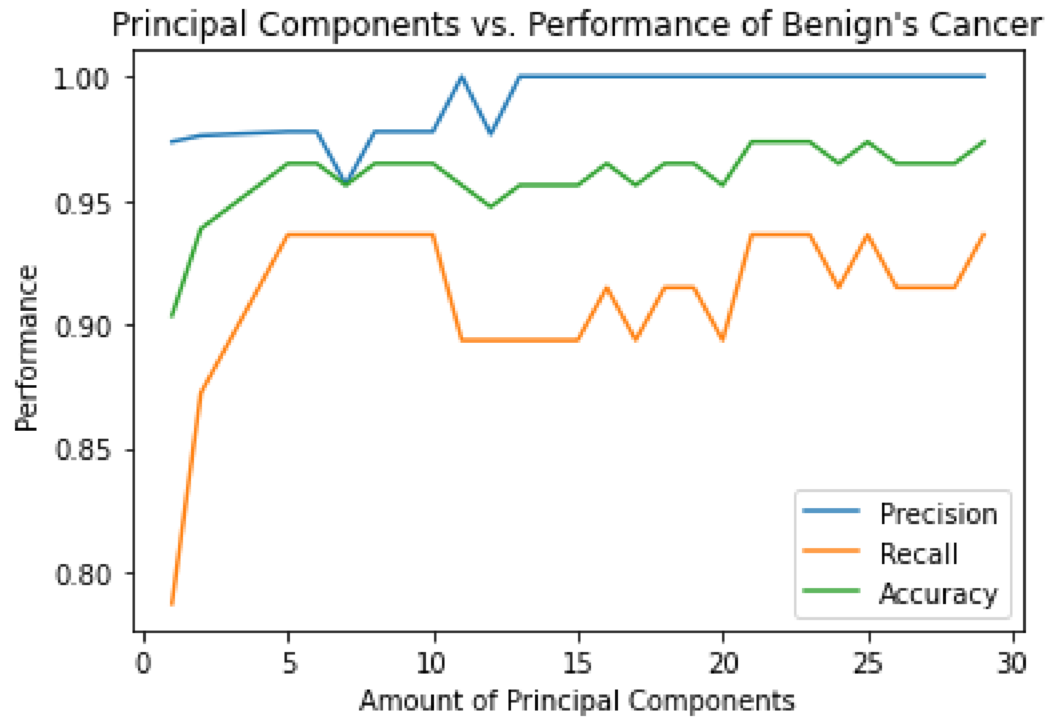
```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.94      0.97        47
         1.0       0.96      1.00      0.98        67

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

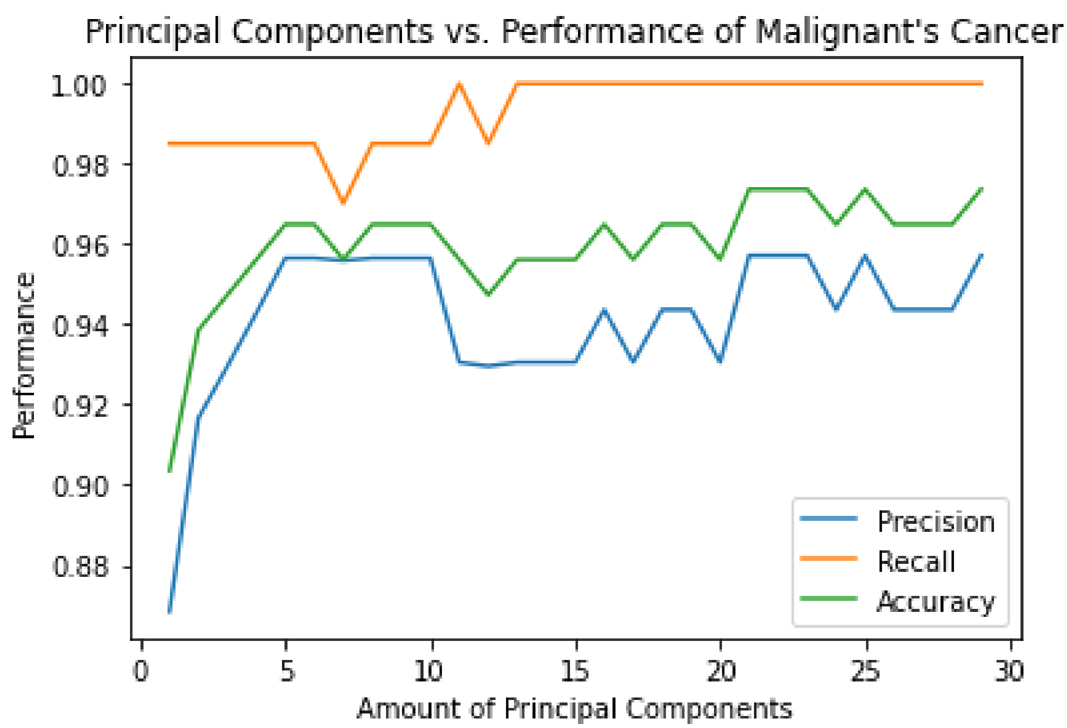
[16]:
```
print("The N of principal components with the highest accuracy is:", high_comp,␣
↪"components with the accuracy of", max_accuracy)
```

```
The N of principal components with the highest accuracy is: 21 components with
the accuracy of 0.9736842105263158
```

[17]:
```
print("Using Guassian Radial Basis Function Kernal:")
plt.plot(Prin_Comp, Precision_0, label = "Precision")
plt.plot(Prin_Comp, Recall_0, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Benign's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
plt.plot(Prin_Comp, Precision_1, label = "Precision")
plt.plot(Prin_Comp, Recall_1, label = "Recall")
plt.plot(Prin_Comp, Accuracy, label = "Accuracy")
plt.title("Principal Components vs. Performance of Malignant's Cancer")
plt.xlabel("Amount of Principal Components")
plt.ylabel("Performance")
plt.legend()
plt.show()
```

```
Using Guassian Radial Basis Function Kernal:
```


Principal Components vs. Performance of Benign's Cancer
Performance
Amount of Principal Components
Precision
Recall
Accuracy
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30


Principal Components vs. Performance of Malignant's Cancer
Performance
Amount of Principal Components
Precision
Recall
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
5
10
15
20
25
30

Problem_2

November 3, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn import datasets
from sklearn import metrics
from sklearn.decomposition import PCA
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:
```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
```

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})
```

```
housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:
```
      price  area  bedrooms  bathrooms  stories  mainroad  guestroom  \
0  13300000  7420         4          2        3         1          0
1  12250000  8960         4          4        4         1          0
2  12250000  9960         3          2        2         1          0
3  12215000  7500         4          2        2         1          0
4  11410000  7420         4          1        2         1          1

   basement  hotwaterheating  airconditioning  parking  prefarea  \
0         0                0                1        2         1
1         0                0                1        3         0
2         1                0                0        2         1
3         1                0                1        3         1
4         1                0                1        2         0

  furnishingstatus
0        furnished
1        furnished
2   semi-furnished
3        furnished
4        furnished
```

[4]:
```
X = housing.values[:,1:12]
print('X =', X[0:5])
Y = housing.values[:,0]
print('Y =', Y[0:5])
```

```
X = [[7420 4 2 3 1 0 0 0 1 2 1]
 [8960 4 4 4 1 0 0 0 1 3 0]
 [9960 3 2 2 1 0 1 0 0 2 1]
 [7500 4 2 2 1 0 1 0 1 3 1]
 [7420 4 1 2 1 1 1 0 1 2 0]]
Y = [13300000 12250000 12250000 12215000 11410000]
```

[5]:
```
# Splitting the datasets to training and validation sets.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state = 0)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(436, 11) (109, 11) (436,) (109,)
```

[6]:
```
# Feature scaling between 0 and 1 for independent variables using␣
 ↪Standardization.
```

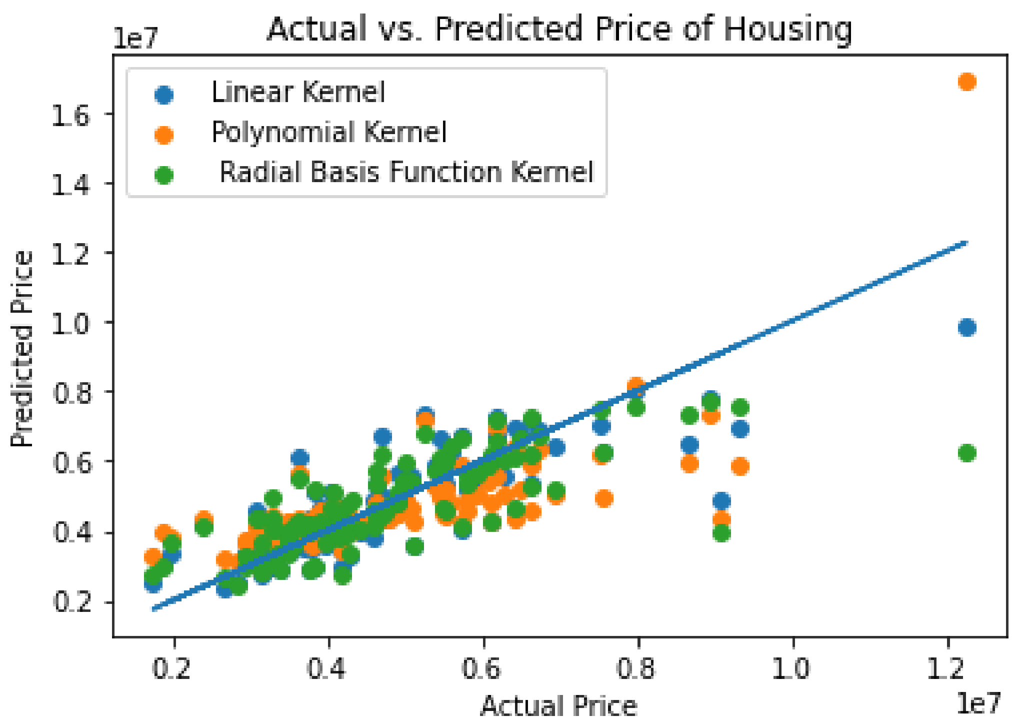
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc_X = StandardScaler()
X_train_sc = sc_X.fit_transform(X_train)
X_test_sc = sc_X.fit_transform(X_test)
```

[7]:
```
model_lin = SVR(kernel = 'linear', C = 1e6)
model_lin.fit(X_train_sc,Y_train)
Y_pred_lin = model_lin.predict(X_test_sc)
model_poly = SVR(kernel = 'poly', C = 1e5)
model_poly.fit(X_train_sc,Y_train)
Y_pred_poly = model_poly.predict(X_test_sc)
model_rbf = SVR(kernel = 'rbf', C = 1e6)
model_rbf.fit(X_train_sc,Y_train)
Y_pred_rbf = model_rbf.predict(X_test_sc)
predictions = pd.DataFrame({'Y_test':Y_test, 'Y_pred_lin':Y_pred_lin,␣
↪'Y_pred_poly':Y_pred_poly, 'Y_pred_rbf':Y_pred_rbf})
predictions.head()
```

[7]:

	Y_test	Y_pred_lin	Y_pred_poly	Y_pred_rbf
0	4585000	3.820968e+06	4.239322e+06	4.096547e+06
1	6083000	5.952978e+06	5.423169e+06	5.845980e+06
2	4007500	4.309663e+06	4.320486e+06	4.138216e+06
3	6930000	6.401910e+06	5.029958e+06	5.216961e+06
4	2940000	3.294395e+06	3.718290e+06	3.304695e+06

[8]:
```
plt.scatter(Y_test, Y_pred_lin, label = 'Linear Kernel')
plt.scatter(Y_test, Y_pred_poly, label = 'Polynomial Kernel')
plt.scatter(Y_test, Y_pred_rbf, label = ' Radial Basis Function Kernel')
plt.plot(Y_test, Y_test)

plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs. Predicted Price of Housing")
plt.legend()
plt.show()
```



[9]:
```
print("Accuracy of Linear's model:", model_lin.score(X_test_sc, Y_test))
print("Accuracy of Polynomial's model:", model_poly.score(X_test_sc, Y_test))
print("Accuracy of Radial Basis Function's model:", model_rbf.score(X_test_sc,␣
↪Y_test))
```

```
Accuracy of Linear's model: 0.6773539534356049
Accuracy of Polynomial's model: 0.5314979804251987
Accuracy of Radial Basis Function's model: 0.5742708039205634
```

[10]:
```
# Implementation SVR Regression (Linear Kernel) over the numbers of principal␣
↪components from PCA extraction scaling.

Prin_Comp = []
Accuracy_lin = []
max_accuracy_lin = 0
max_comp_lin = 0

N_Comp = range(1,11)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
```

```
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,␣
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model_lin = SVR(kernel = 'linear', C = 1e6)
    model_lin.fit(X_train_sc,Y_train)
    Y_pred_lin = model_lin.predict(X_test_sc)
    Accuracy_lin.append(model_lin.score(X_test_sc, Y_test))
    if model_lin.score(X_test_sc, Y_test) > max_accuracy_lin:
        max_comp_lin = N
        max_accuracy_lin = model_lin.score(X_test_sc, Y_test)
print("The N of principal components with the highest accuracy is:",␣
↪max_comp_lin, "components with the accuracy of", max_accuracy_lin)
```

```
The N of principal components with the highest accuracy is: 8 components with
the accuracy of 0.6881920717708833
```

[11]:
```
# Implementation SVR Regression (Polynomial Kernel) over the numbers of␣
↪principal components from PCA extraction scaling.

Prin_Comp = []
Accuracy_poly = []
max_accuracy_poly = 0
max_comp_poly = 0

N_Comp = range(1,11)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,␣
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model_poly = SVR(kernel = 'poly', C = 1e5)
    model_poly.fit(X_train_sc,Y_train)
    Y_pred_poly = model_poly.predict(X_test_sc)
    Accuracy_poly.append(model_poly.score(X_test_sc, Y_test))
    if model_poly.score(X_test_sc, Y_test) > max_accuracy_poly:
        max_comp_poly = N
        max_accuracy_poly = model_poly.score(X_test_sc, Y_test)
print("The N of principal components with the highest accuracy is:",␣
↪max_comp_poly, "components with the accuracy of", max_accuracy_poly)
```

```
The N of principal components with the highest accuracy is: 5 components with
```

```
the accuracy of 0.5354556738228378
```

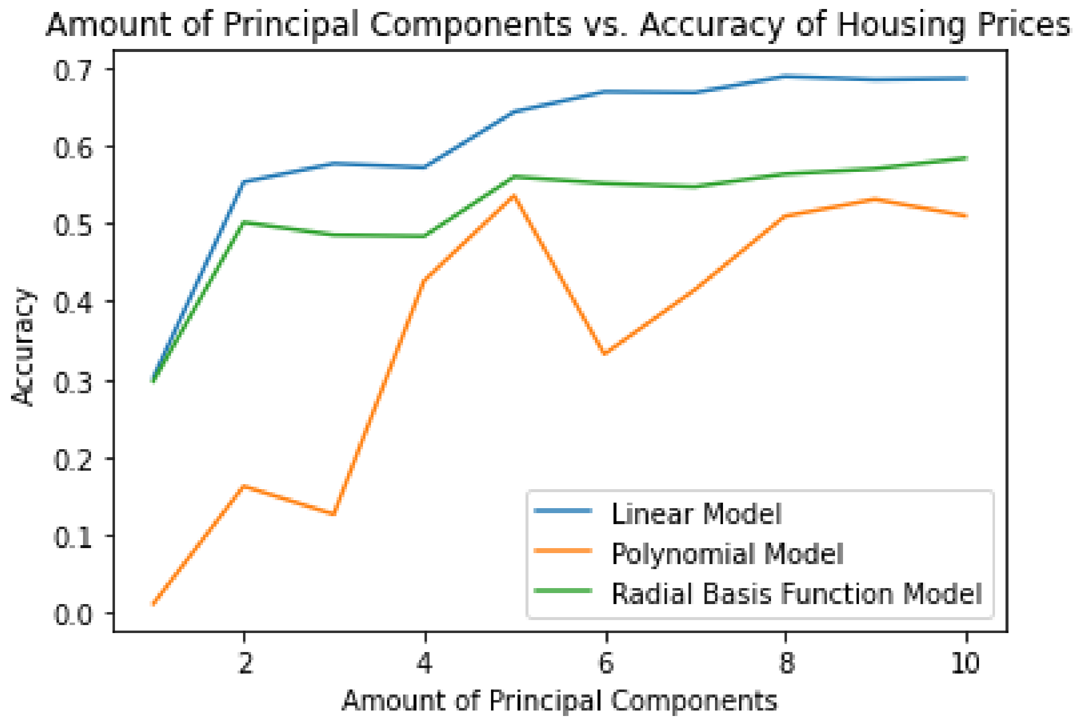
[12]:

```
# Implementation SVR Regression (Radial Basis Function Kernel) over the numbers␣
↪of principal components from PCA extraction scaling.

Prin_Comp = []
Accuracy_rbf = []
max_accuracy_rbf = 0
max_comp_rbf = 0

N_Comp = range(1,11)
for N in N_Comp:
    Prin_Comp.append(N)
    pca = PCA(n_components = N)
    principalComponents = pca.fit_transform(X)
    principalDF = pd.DataFrame(data = principalComponents)
    X_train, X_test, Y_train, Y_test = train_test_split(principalDF, Y,␣
↪test_size=0.2, random_state = 0)
    sc_X = StandardScaler()
    X_train_sc = sc_X.fit_transform(X_train)
    X_test_sc = sc_X.fit_transform(X_test)
    model_rbf = SVR(kernel = 'rbf', C = 1e6)
    model_rbf.fit(X_train_sc,Y_train)
    Y_pred_rbf = model_rbf.predict(X_test_sc)
    Accuracy_rbf.append(model_rbf.score(X_test_sc, Y_test))
    if model_rbf.score(X_test_sc, Y_test) > max_accuracy_rbf:
        max_comp_rbf = N
        max_accuracy_rbf = model_rbf.score(X_test_sc, Y_test)
print("The N of principal components with the highest accuracy is:",␣
↪max_comp_rbf, "components with the accuracy of", max_accuracy_rbf)
```

```
The N of principal components with the highest accuracy is: 10 components with
the accuracy of 0.5834690324496696
```

[13]:

```
plt.plot(Prin_Comp, Accuracy_lin, label = 'Linear Model')
plt.plot(Prin_Comp, Accuracy_poly, label = 'Polynomial Model')
plt.plot(Prin_Comp, Accuracy_rbf, label = 'Radial Basis Function Model')
plt.xlabel("Amount of Principal Components")
plt.ylabel("Accuracy")
plt.title("Amount of Principal Components vs. Accuracy of Housing Prices")
plt.legend()
plt.show()
```


Amount of Principal Components vs. Accuracy of Housing Prices
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Accuracy
2
4
6
8
10
Amount of Principal Components
Linear Model
Polynomial Model
Radial Basis Function Model

[ ]: