Stanly Gomes
Student ID: 801118166
Homework 4

Problem1

November 4, 2022

[21]:
```
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; sns.set()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[22]:
```
breast = load_breast_cancer()
```

[23]:
```
breast_data = breast.data
breast_data.shape
```

[23]:
```
(569, 30)
```

[24]:
```
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

[24]:
```
       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]
```

[25]:
```
breast_labels = breast.target
breast_labels.shape
```

[25]:
```
(569,)
```

[26]:
```
labels = np.reshape(breast_labels,(569,1))
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

[26]:
```
(569, 31)
```

[27]:
```
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features
```

[27]:
```
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')
```

[28]:
```
features_labels = np.append(features,'Type')
breast_dataset.columns = features_labels
breast_dataset.head()
```

[28]:
```
   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]
```

[29]:

```
# Commented out due to scatter plot function needing number values
#breast_dataset['Type'].replace(0, 'Benign',inplace=True)
#breast_dataset['Type'].replace(1, 'Malignant',inplace=True)
```

[30]:

```
breast_dataset.tail()
```

[30]:

```
     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   0.0
565                0.1628          0.2572                  0.06637   0.0
566                0.1418          0.2218                  0.07820   0.0
567                0.2650          0.4087                  0.12400   0.0
568                0.0000          0.2871                  0.07039   1.0

[5 rows x 31 columns]
```

[31]:
```python
# 80% Training split
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]
```

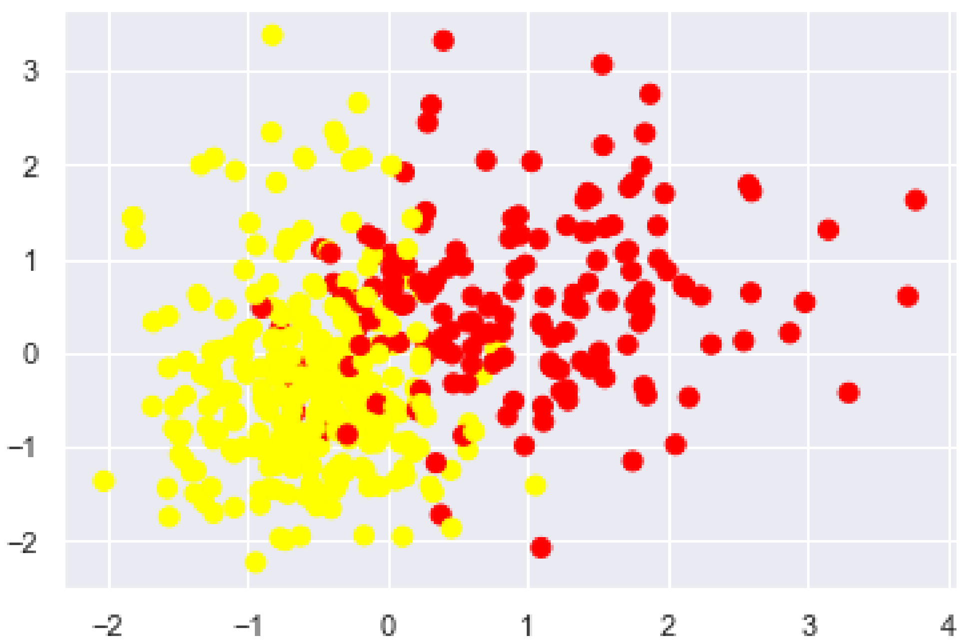
[31]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

1 Problem 1

Use the cancer dataset to build an SVM classifier to classify the type of cancer (Malignant vs. benign). Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K).

1. Identify the optimum number of K, principal components that achieve the highest classification accuracy.
2. Plot your classification accuracy, precision, and recall over a different number of Ks.
3. Explore different kernel tricks to capture non-linearities within your data. Plot the results and compare the accuracies for different kernels.
4. Compare your results against the logistic regression that you have done in homework 3. Make sure to explain and elaborate your results.

[32]:
```
# Determine how linear datset currently is
plt.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, s=50, cmap='autumn');
```



[33]:
```
model = SVC(kernel='linear', C=1E10)
model.fit(X_train, Y_train.ravel())
```

[33]: SVC(C=10000000000.0, kernel='linear')

[34]:
```
# Perform training with K number of prinicipal components
Accuracy = []
Precision_B = []
Recall_B = []
```

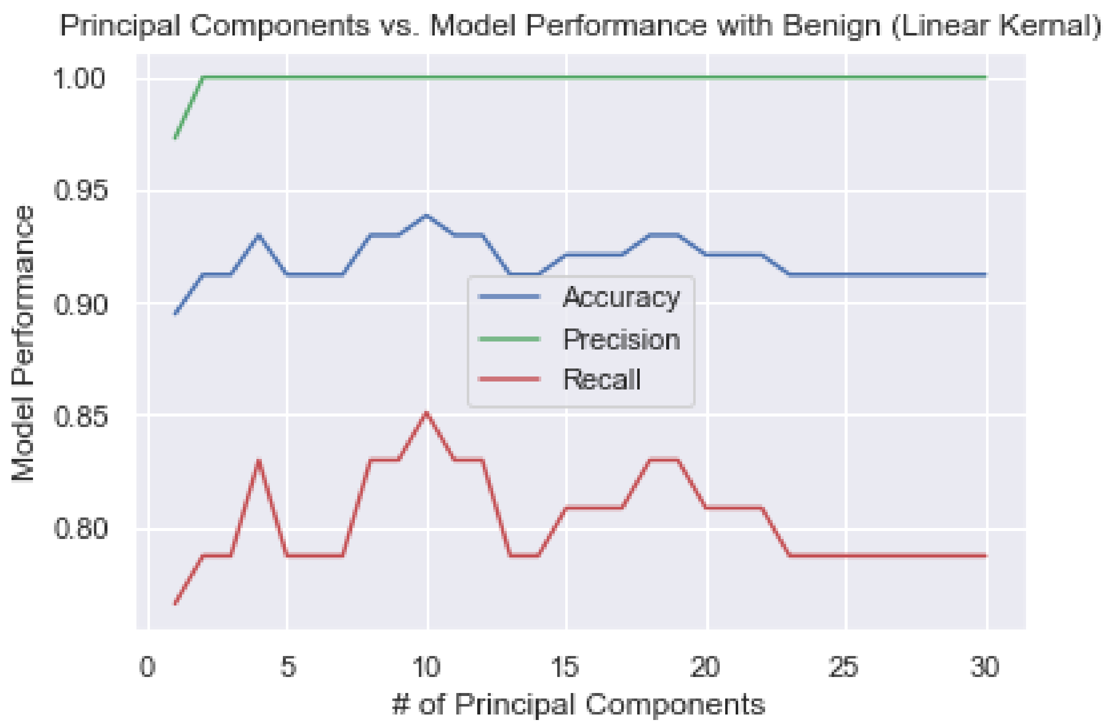
```
Precision_M = []
Recall_M = []
max_accuracy = 0;
max_k = 0;

k = range(1,len(features)+1)
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)

    principalComponents = pca_final.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_X.fit_transform(X_train)
    X_test = sc_X.fit_transform(X_test)

    svc = SVC(kernel='linear', C=0.01)
    model_SVC = svc.fit(X_train, Y_train.ravel())
    Y_pred = model_SVC.predict(X_test)
    #Accuracy.append(metrics.accuracy_score(Y_test, Y_pred))
    #Precision.append(metrics.precision_score(Y_test, Y_pred))
    #Recall.append(metrics.recall_score(Y_test, Y_pred))
    classReport = classification_report(Y_test, Y_pred, output_dict=True)
    classData = pd.DataFrame(classReport)
    Accuracy.append(classData.values[0,2])
    Precision_B.append(classData.values[0,0])
    Precision_M.append(classData.values[0,1])
    Recall_B.append(classData.values[1,0])
    Recall_M.append(classData.values[1,1])
    if Accuracy[i-1] > max_accuracy:
        max_k = i
        max_accuracy = Accuracy[i-1]
print(metrics.classification_report(Y_test, Y_pred))
print(metrics.confusion_matrix(Y_test, Y_pred))
print("The optimal number of principal components:",
      max_k, "with an accuracy of: ", max_accuracy)
```

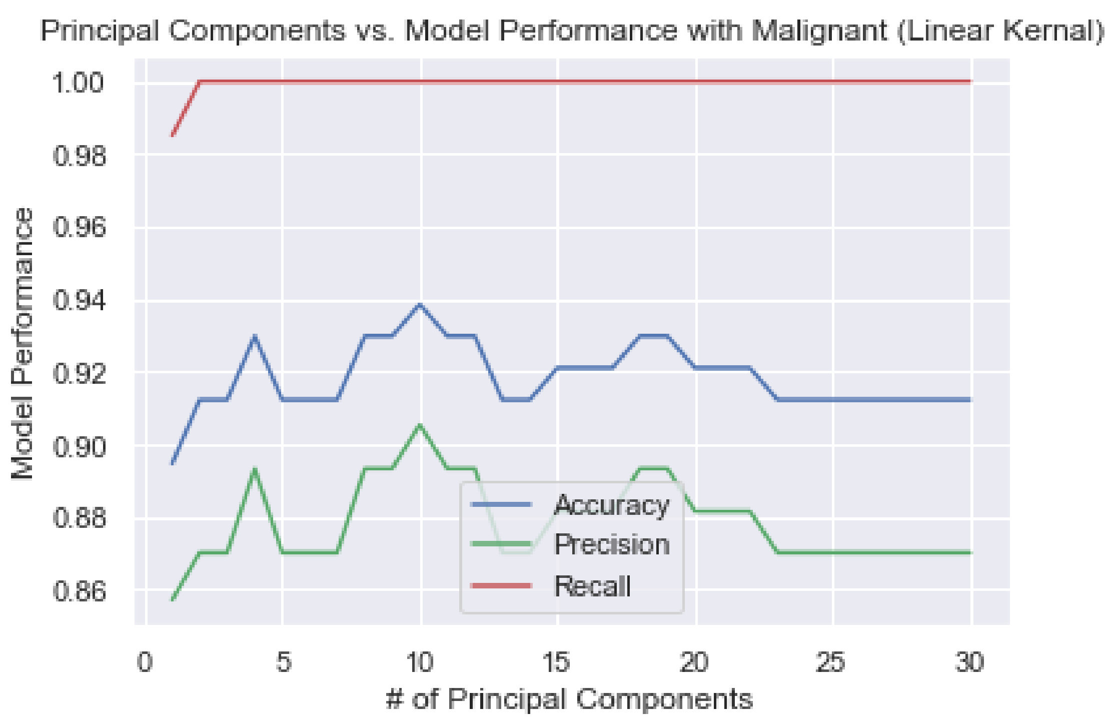
```
              precision    recall  f1-score   support

         0.0       1.00      0.79      0.88        47
         1.0       0.87      1.00      0.93        67

    accuracy                           0.91       114
   macro avg       0.94      0.89      0.91       114
```

```
weighted avg       0.92      0.91      0.91       114

[[37 10]
 [ 0 67]]
The optimal number of principal components: 10 with an accuracy of:
0.9385964912280702
```

[35]:

```python
# Linear Kernel: Benign Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_B, 'g', label="Precision")
plt.plot(k, Recall_B, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Benign (Linear␣
 ↪Kernal)")
plt.legend()
plt.show()

# Linear Kernel: Malignant Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_M, 'g', label="Precision")
plt.plot(k, Recall_M, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Malignant (Linear␣
 ↪Kernal)")
plt.legend()
plt.show()
```


Principal Components vs. Model Performance with Benign (Linear Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30


Principal Components vs. Model Performance with Malignant (Linear Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0
5
10
15
20
25
30

[36]:

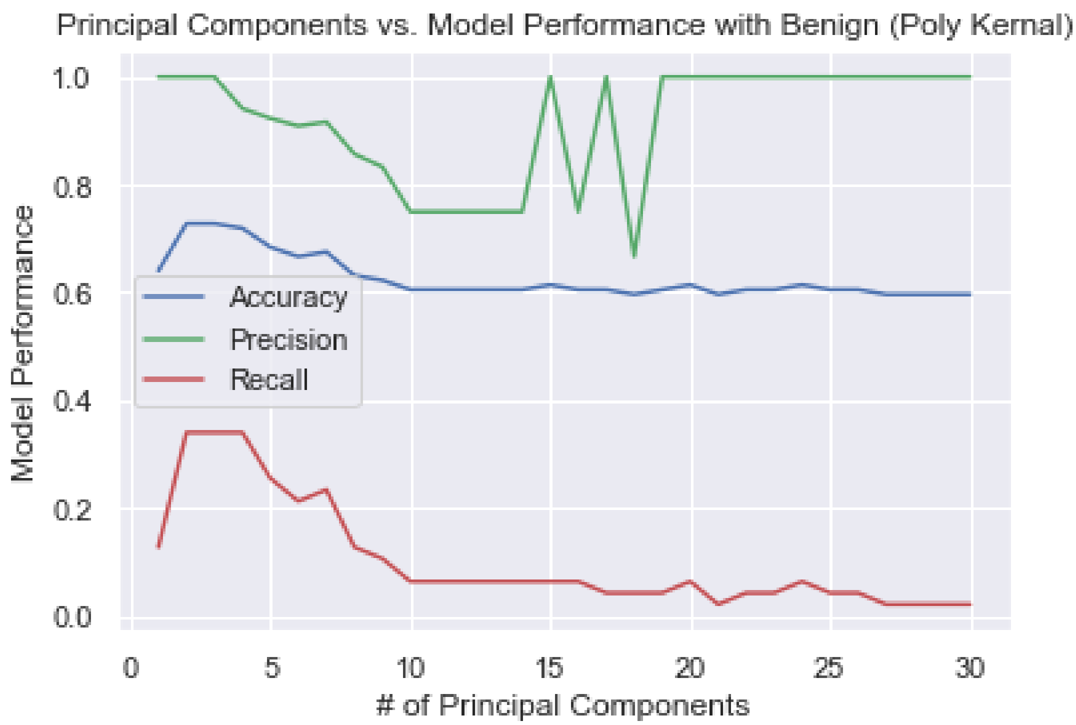
```python
# Perform training with K number of prinicipal components (Poly Kernel)
Accuracy = []
Precision_B = []
Recall_B = []
Precision_M = []
Recall_M = []
max_accuracy = 0
max_k = 0

k = range(1,len(features)+1)
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)

    principalComponents = pca_final.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_X.fit_transform(X_train)
    X_test = sc_X.fit_transform(X_test)

    svc = SVC(kernel='poly', C=0.1, degree = 15)
    model_SVC = svc.fit(X_train, Y_train.ravel())
    Y_pred = model_SVC.predict(X_test)
    #Accuracy.append(metrics.accuracy_score(Y_test, Y_pred))
    #Precision.append(metrics.precision_score(Y_test, Y_pred))
    #Recall.append(metrics.recall_score(Y_test, Y_pred))
    classReport = classification_report(Y_test, Y_pred, output_dict=True)
    classData = pd.DataFrame(classReport)
    Accuracy.append(classData.values[0,2])
    Precision_B.append(classData.values[0,0])
    Precision_M.append(classData.values[0,1])
    Recall_B.append(classData.values[1,0])
    Recall_M.append(classData.values[1,1])
    if Accuracy[i-1] > max_accuracy:
        max_k = i
        max_accuracy = Accuracy[i-1]
print(metrics.classification_report(Y_test, Y_pred))
print(metrics.confusion_matrix(Y_test, Y_pred))
print("The optimal number of principal components:",
      max_k, "with an accuracy of: ", max_accuracy)
```

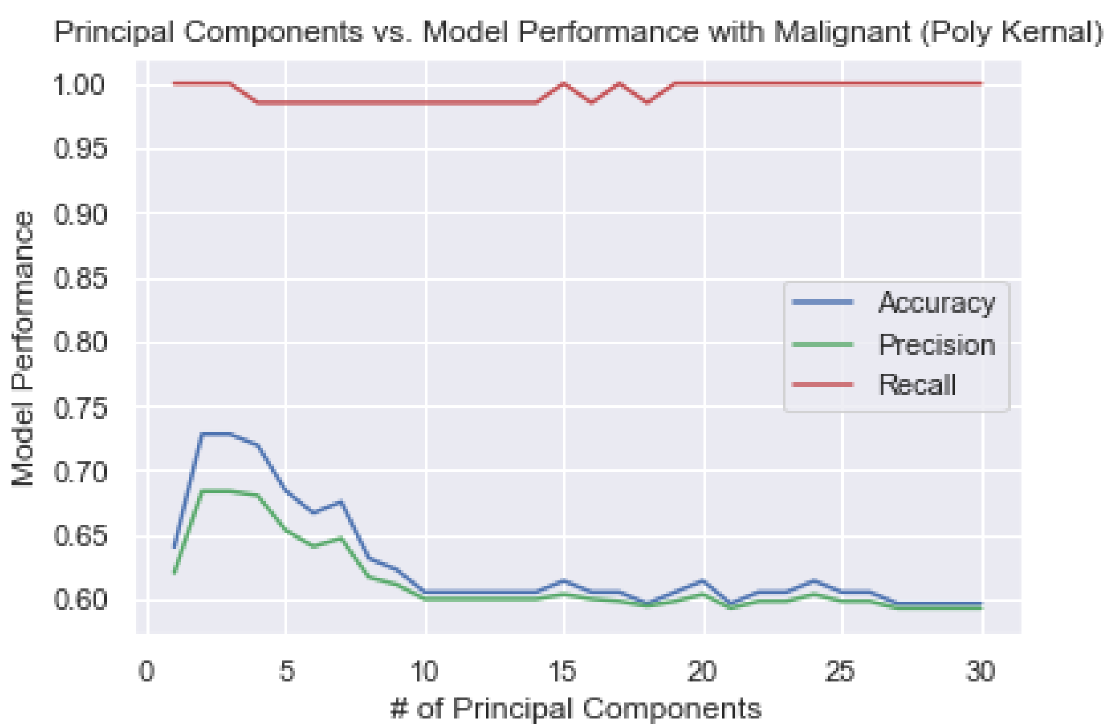
```
              precision    recall  f1-score   support

         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

[[ 1 46]
 [ 0 67]]
The optimal number of principal components: 2 with an accuracy of:
0.7280701754385965
```

[37]:

```
# Poly Kernel: Benign Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_B, 'g', label="Precision")
plt.plot(k, Recall_B, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Benign (Poly␣
 ↪Kernal)")
plt.legend()
plt.show()

# Poly Kernel: Malignant Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_M, 'g', label="Precision")
plt.plot(k, Recall_M, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Malignant (Poly␣
 ↪Kernal)")
plt.legend()
plt.show()
```


Principal Components vs. Model Performance with Benign (Poly Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30


Principal Components vs. Model Performance with Malignant (Poly Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
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

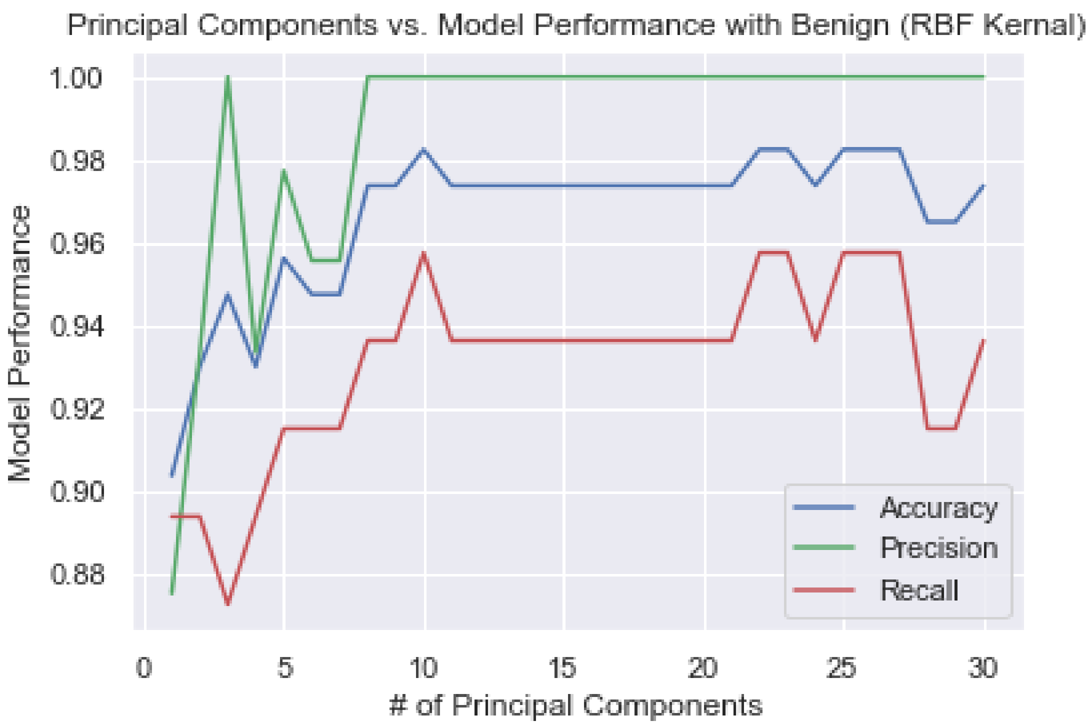
[38]:
```python
# Perform training with K number of prinicipal components (rbf kernel)
Accuracy = []
Precision_B = []
Recall_B = []
Precision_M = []
Recall_M = []
max_accuracy = 0
max_k

k = range(1,len(features)+1)
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)

    principalComponents = pca_final.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_X.fit_transform(X_train)
    X_test = sc_X.fit_transform(X_test)

    svc = SVC(kernel='rbf', C=10, gamma=0.01)
    model_SVC = svc.fit(X_train, Y_train.ravel())
    Y_pred = model_SVC.predict(X_test)
    #Accuracy.append(metrics.accuracy_score(Y_test, Y_pred))
    #Precision.append(metrics.precision_score(Y_test, Y_pred))
    #Recall.append(metrics.recall_score(Y_test, Y_pred))
    classReport = classification_report(Y_test, Y_pred, output_dict=True)
    classData = pd.DataFrame(classReport)
    Accuracy.append(classData.values[0,2])
    Precision_B.append(classData.values[0,0])
    Precision_M.append(classData.values[0,1])
    Recall_B.append(classData.values[1,0])
    Recall_M.append(classData.values[1,1])
    if Accuracy[i-1] > max_accuracy:
        max_k = i
        max_accuracy = Accuracy[i-1]
print(metrics.classification_report(Y_test, Y_pred))
print(metrics.confusion_matrix(Y_test, Y_pred))
print("The optimal number of principal components:",
      max_k, "with an accuracy of: ", max_accuracy)
```

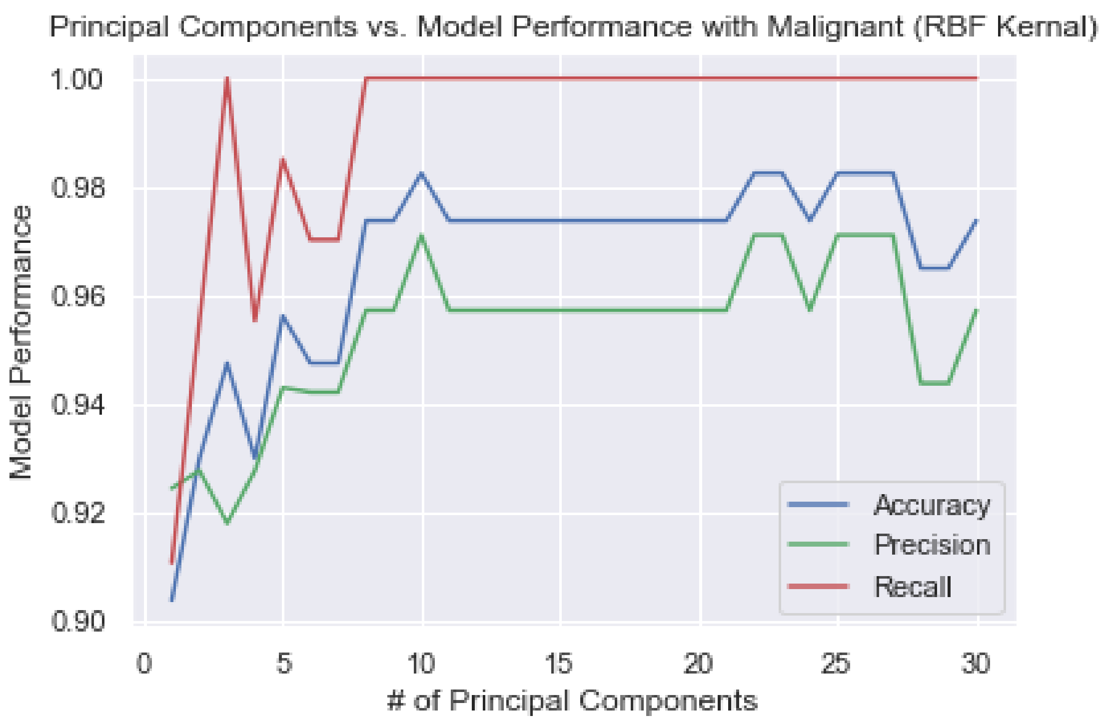
```
              precision    recall  f1-score   support

         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

[[44  3]
 [ 0 67]]
The optimal number of principal components: 10 with an accuracy of:
0.9824561403508771
```

[39]:

```
# RBF Kernel: Benign Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_B, 'g', label="Precision")
plt.plot(k, Recall_B, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Benign (RBF Kernal)")
plt.legend()
plt.show()

# RBF Kernel: Malignant Performance
plt.plot(k, Accuracy, 'b', label="Accuracy")
plt.plot(k, Precision_M, 'g', label="Precision")
plt.plot(k, Recall_M, 'r', label="Recall")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Model Performance with Malignant (RBF␣
 ↪Kernal)")
plt.legend()
plt.show()
```


Principal Components vs. Model Performance with Benign (RBF Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
5
10
15
20
25
30


Principal Components vs. Model Performance with Malignant (RBF Kernal)
Model Performance
# of Principal Components
Accuracy
Precision
Recall
1.00
0.98
0.96
0.94
0.92
0.90
0
5
10
15
20
25
30

```
[ ]:
```

```
[ ]:
```

Problem2

November 4, 2022

[140]:

```
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; sns.set()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[141]:

```
housing = pd.DataFrame(pd.read_csv("./Housing.csv"))
housing.head()
```

[141]:

```
      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
```

[142]:
```
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']
# Define map function
def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

# Applying the function to housing list
housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[142]:
```
      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
```

1 Problem 2

Develop a SVR regression model that predicts housing price based on the following input variables:

Area, bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, prefarea

1. Plot your regression model for SVR similar to the sample code provided on Canvas.
2. Compare your results against linear regression with regularization loss that you already did in homework1.
3. Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K). Identify the optimum number of K, principal components that achieve the highest regression accuracy.
4. Explore different kernel tricks to capture non-linearities within your data. Plot the results and compare the accuracies for different kernels.

[143]:
```python
# Obtain SVR model and compare performance with linear regression
input_vars = ['area', 'bedrooms', 'bathrooms', 'stories',
            'mainroad', 'guestroom', 'basement', 'hotwaterheating',
            'airconditioning', 'parking', 'prefarea']
x = housing[input_vars]
y = housing['price']
y.head()
```

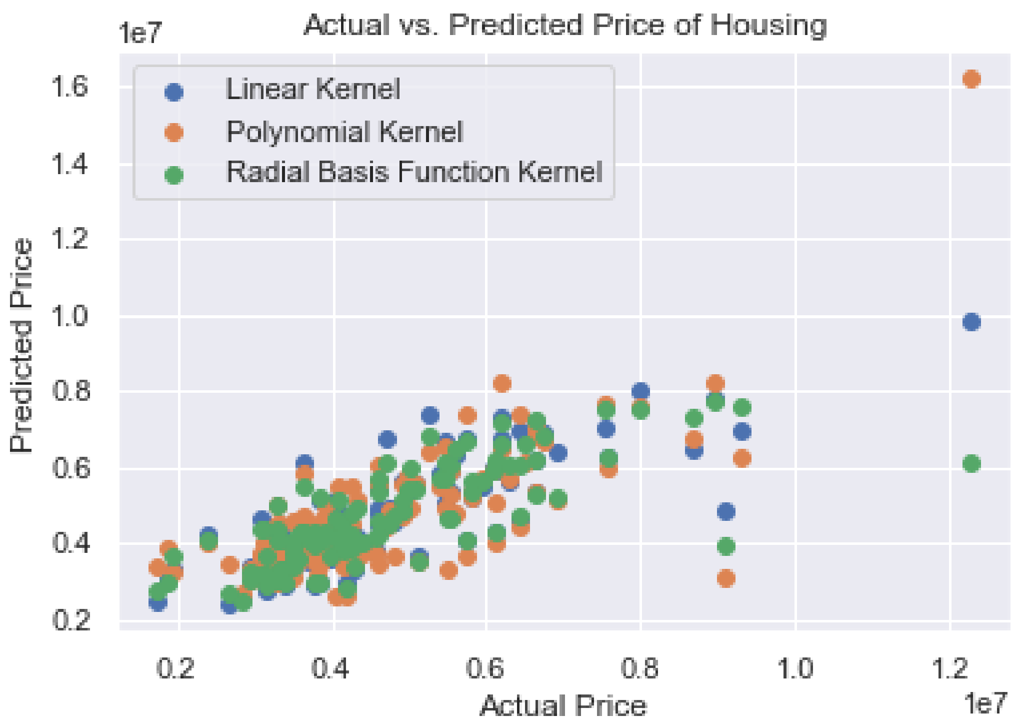
[143]:
```
0    13300000
1    12250000
2    12250000
3    12215000
4    11410000
Name: price, dtype: int64
```

[144]:
```python
# Training split and feature scaling
standScale = StandardScaler()
# x = standScale.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_train.head()

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)

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
```

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

[152]:
```python
svr_lin = SVR(kernel='linear', C=1e6)
y_lin = svr_lin.fit(X_train, Y_train).predict(X_test)
svr_poly = SVR(kernel='poly', C=1e6, degree=2)
y_poly = svr_poly.fit(X_train, Y_train).predict(X_test)
svr_rbf = SVR(kernel='rbf', C=1e6, gamma=0.1)
y_rbf = svr_rbf.fit(X_train, Y_train).predict(X_test)
```

[151]:
```python
plt.scatter(Y_test, y_lin, label = 'Linear Kernel')
plt.scatter(Y_test, y_poly, label = 'Polynomial Kernel')
plt.scatter(Y_test, y_rbf, label = 'Radial Basis Function Kernel')

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



[147]:

```
# Linear Kernel
Accuracy_lin = []
max_accuracy_lin = 0
max_k_lin = 0

k = range(1,len(input_vars)+1)
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)

    principalComponents = pca_final.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)
    print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)

    svr = SVR(kernel='linear', C=1e6)
```

```
    model_SVR = svr.fit(X_train, Y_train)
    Y_pred = model_SVR.predict(X_test)
    #Accuracy_lin.append(mean_squared_error(Y_test, Y_pred))
    Accuracy_lin.append(r2_score(Y_test, Y_pred))
    #Accuracy_lin.append(model_SVR.score(Y_test, Y_pred))
    if Accuracy_lin[i-1] > max_accuracy_lin:
        max_k_lin = i
        max_accuracy_lin = Accuracy_lin[i-1]
print("The optimal number of principal components:",
      max_k_lin, "with an accuracy of: ", max_accuracy_lin)
```

```
(436, 1) (109, 1) (436,) (109,)
(436, 2) (109, 2) (436,) (109,)
(436, 3) (109, 3) (436,) (109,)
(436, 4) (109, 4) (436,) (109,)
(436, 5) (109, 5) (436,) (109,)
(436, 6) (109, 6) (436,) (109,)
(436, 7) (109, 7) (436,) (109,)
(436, 8) (109, 8) (436,) (109,)
(436, 9) (109, 9) (436,) (109,)
(436, 10) (109, 10) (436,) (109,)
(436, 11) (109, 11) (436,) (109,)
The optimal number of principal components: 8 with an accuracy of:
0.6869222993273993
```

[148]:

```
# Poly Kernel
Accuracy_poly = []
max_accuracy_poly = 0
max_k_poly = 0

k = range(1,len(input_vars)+1)
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)

    principalComponents = pca_final.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)
    print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)
```

```
    svr = SVR(kernel='poly', C=1e6)
    model_SVR = svr.fit(X_train, Y_train)
    Y_pred = model_SVR.predict(X_test)
    #Accuracy_poly.append(mean_squared_error(Y_test, Y_pred))
    Accuracy_poly.append(r2_score(Y_test, Y_pred))
    #Accuracy_poly.append(model_SVR.score(Y_train, Y_test))
    if Accuracy_poly[i-1] > max_accuracy_poly:
        max_k_poly = i
        max_accuracy_poly = Accuracy_poly[i-1]
print("The optimal number of principal components:",
      max_k_poly, "with an accuracy of: ", max_accuracy_poly)
```

```
(436, 1) (109, 1) (436,) (109,)
(436, 2) (109, 2) (436,) (109,)
(436, 3) (109, 3) (436,) (109,)
(436, 4) (109, 4) (436,) (109,)
(436, 5) (109, 5) (436,) (109,)
(436, 6) (109, 6) (436,) (109,)
(436, 7) (109, 7) (436,) (109,)
(436, 8) (109, 8) (436,) (109,)
(436, 9) (109, 9) (436,) (109,)
(436, 10) (109, 10) (436,) (109,)
(436, 11) (109, 11) (436,) (109,)
The optimal number of principal components: 5 with an accuracy of:
0.5109504213287848
```

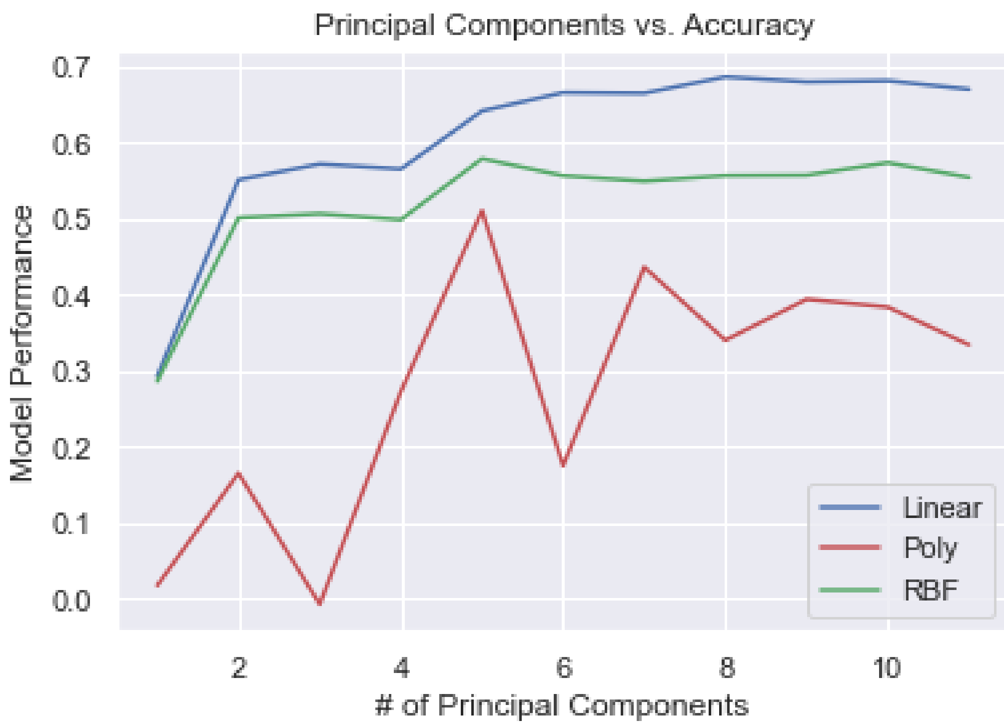
[149]:

```
# RBF Kernel
Accuracy_rbf = []
max_accuracy_rbf = 0
max_k_rbf = 0

k = range(1,len(input_vars)+1)
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)

    principalComponents = pca_final.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)
    print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)
```

```
    svr = SVR(kernel='rbf', C=1e6)
    model_SVR = svr.fit(X_train, Y_train)
    Y_pred = model_SVR.predict(X_test)
    #Accuracy_rbf.append(mean_squared_error(Y_test, Y_pred))
    Accuracy_rbf.append(r2_score(Y_test, Y_pred))
    #Accuracy_rbf.append(model_SVR.score(Y_train, Y_test))
    if Accuracy_rbf[i-1] > max_accuracy_rbf:
        max_k_rbf = i
        max_accuracy_rbf = Accuracy_rbf[i-1]
print("The optimal number of principal components:",
      max_k_rbf, "with an accuracy of: ", max_accuracy_rbf)
```

```
(436, 1) (109, 1) (436,) (109,)
(436, 2) (109, 2) (436,) (109,)
(436, 3) (109, 3) (436,) (109,)
(436, 4) (109, 4) (436,) (109,)
(436, 5) (109, 5) (436,) (109,)
(436, 6) (109, 6) (436,) (109,)
(436, 7) (109, 7) (436,) (109,)
(436, 8) (109, 8) (436,) (109,)
(436, 9) (109, 9) (436,) (109,)
(436, 10) (109, 10) (436,) (109,)
(436, 11) (109, 11) (436,) (109,)
The optimal number of principal components: 5 with an accuracy of:
0.5794939259825025
```

[150]:
```
plt.plot(k, Accuracy_lin, 'b', label="Linear")
plt.plot(k, Accuracy_poly, 'r', label="Poly")
plt.plot(k, Accuracy_rbf, 'g', label="RBF")
plt.xlabel("# of Principal Components")
plt.ylabel("Model Performance")
plt.title("Principal Components vs. Accuracy")
plt.legend()
plt.show()
```


Principal Components vs. Accuracy
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Model Performance
2
4
6
8
10
# of Principal Components
Linear
Poly
RBF

[ ]:

[ ]: