Homework 4

November 24, 2021

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings

import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from matplotlib.colors import ListedColormap
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.svm import SVR
```

[6]:
```
# Importing and cleaning the data
breast = load_breast_cancer()
breast_data = breast.data
breast_input = pd.DataFrame(breast_data)
breast_labels = breast.target
labels = np.reshape(breast_labels,(569,1))
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features_labels = np.append(features,'label')
breast_dataset.columns = features_labels
```

[3]:
```
X = breast_dataset.iloc[:,:30].values
Y = breast_dataset.iloc[:, 30].values

# Scaling the features
sc_X = StandardScaler()
std_X = sc_X.fit_transform(X)
```

[18]:
```
# Problem 1
# Feature extraction using PCA
# Scaling the features
accuracy = []
precision = []
recall = []


# Using a linear kernal
for i in range(30):
    print(i+1)
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVC(kernel='linear', C=1E6)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_pred = model.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))

# Plotting the accuracy, precision, and recall against the iterations
plt.figure()
plt.plot(range(30),accuracy)
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Figure 1: Accuracy vs the Iterations for Linear Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),precision)
plt.xlabel('Iterations')
plt.ylabel('Precision')
plt.title('Figure 2: Precision vs the Iterations for Linear Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),recall)
plt.xlabel('Iterations')
```

```
plt.ylabel('Recall')
plt.title('Figure 3: Recall vs the Iterations for Linear Kernel')
plt.xticks(range(30))
plt.show()
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Figure 1: Accuracy vs the Iterations for Linear Kernel

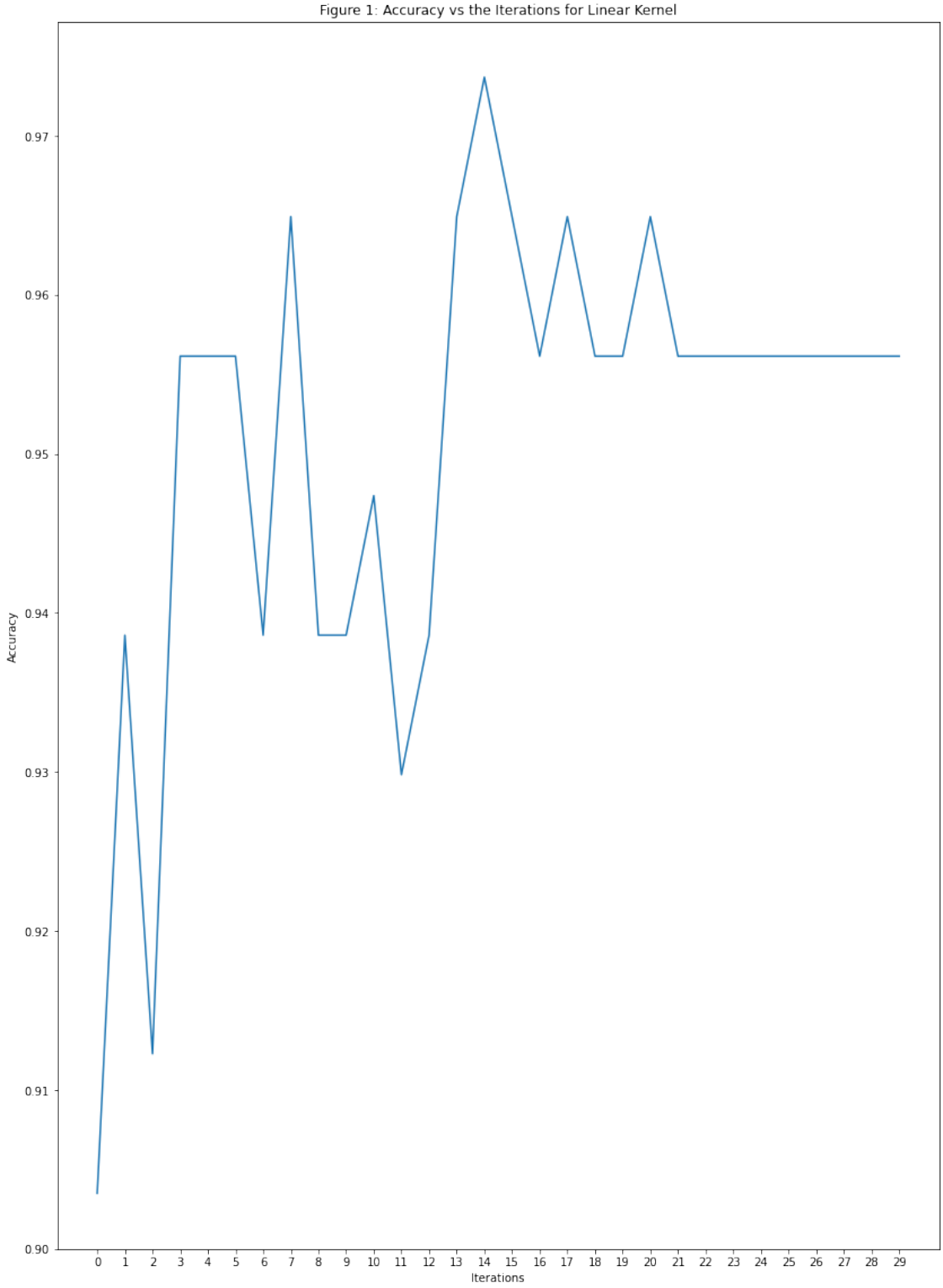


Figure 2: Precision vs the Iterations for Linear Kernel

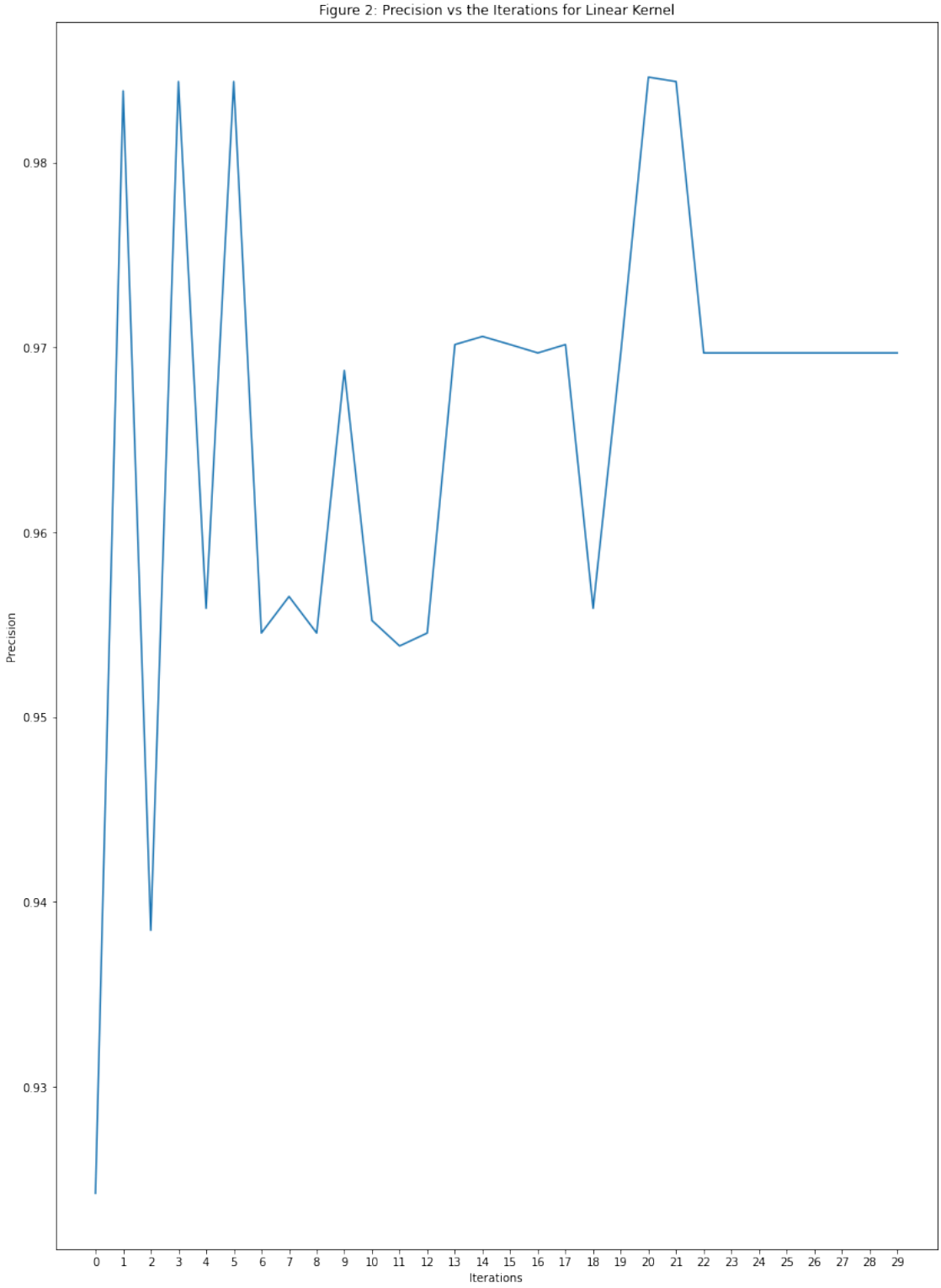
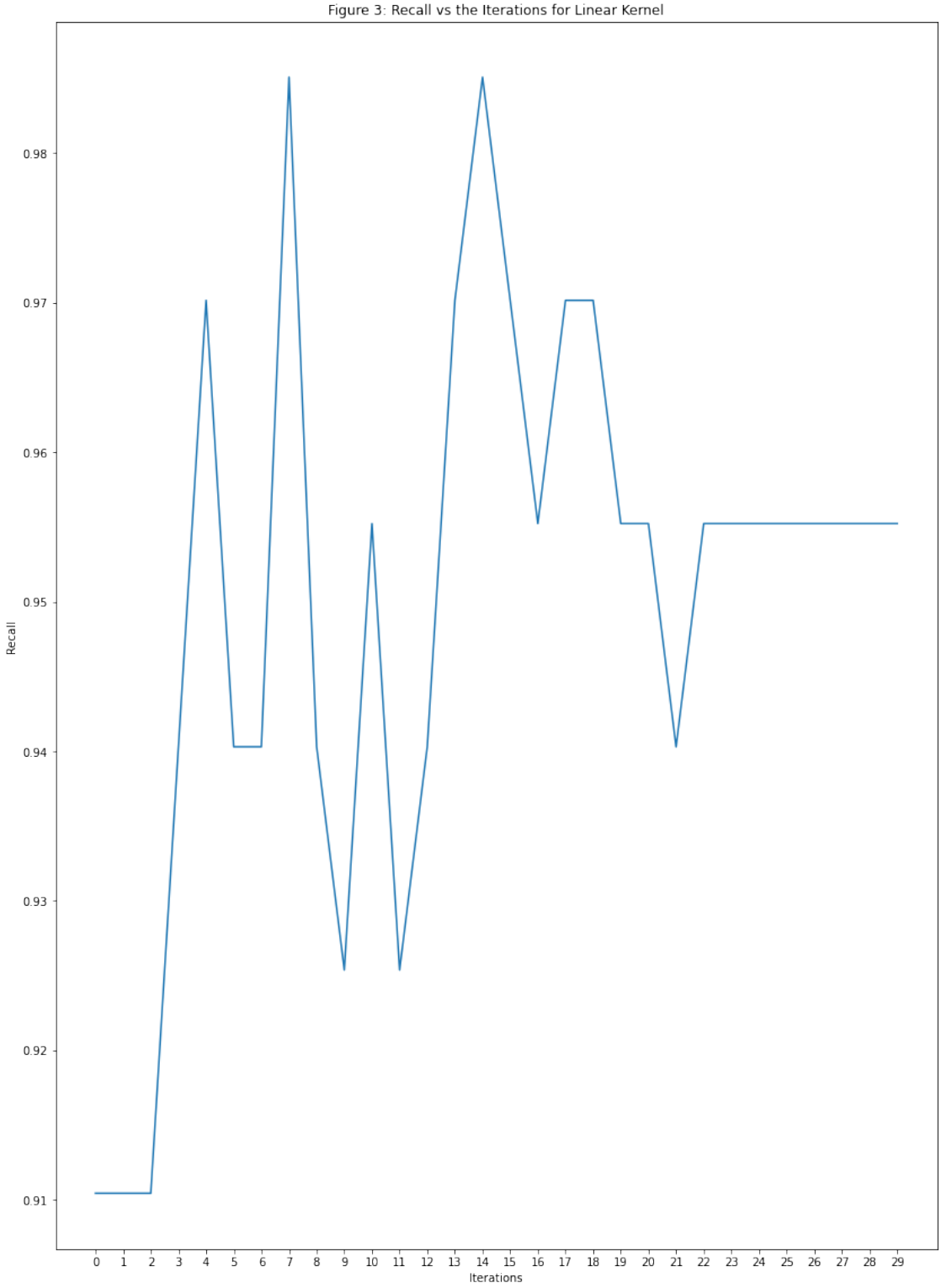


Figure 3: Recall vs the Iterations for Linear Kernel



[16]:
```
# Using a poly kernal
accuracy = []
precision = []
recall = []
for i in range(30):
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVC(kernel='poly', C=1E6)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_pred = model.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))

# Plotting the accuracy, precision, and recall against the iterations
plt.figure()
plt.plot(range(30),accuracy)
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Figure 4: Accuracy vs the Iterations for Poly Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),precision)
plt.xlabel('Iterations')
plt.ylabel('Precision')
plt.title('Figure 5: Precision vs the Iterations for Poly Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),recall)
plt.xlabel('Iterations')
plt.ylabel('Recall')
plt.title('Figure 6: Recall vs the Iterations for Poly Kernel')
plt.xticks(range(30))
plt.show()
```

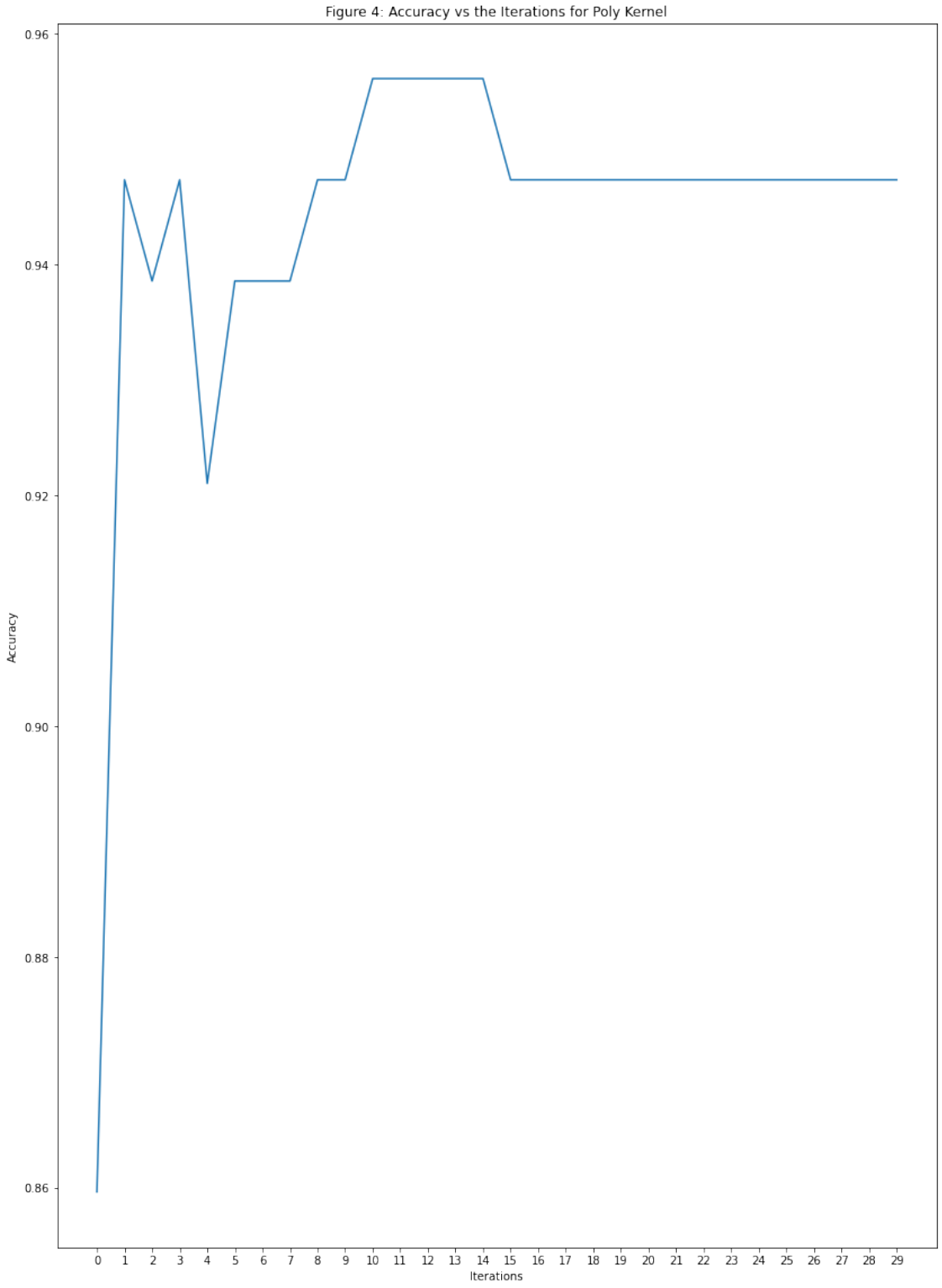


Figure 4: Accuracy vs the Iterations for Poly Kernel
Accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Iterations

Figure 5: Precision vs the Iterations for Poly Kernel

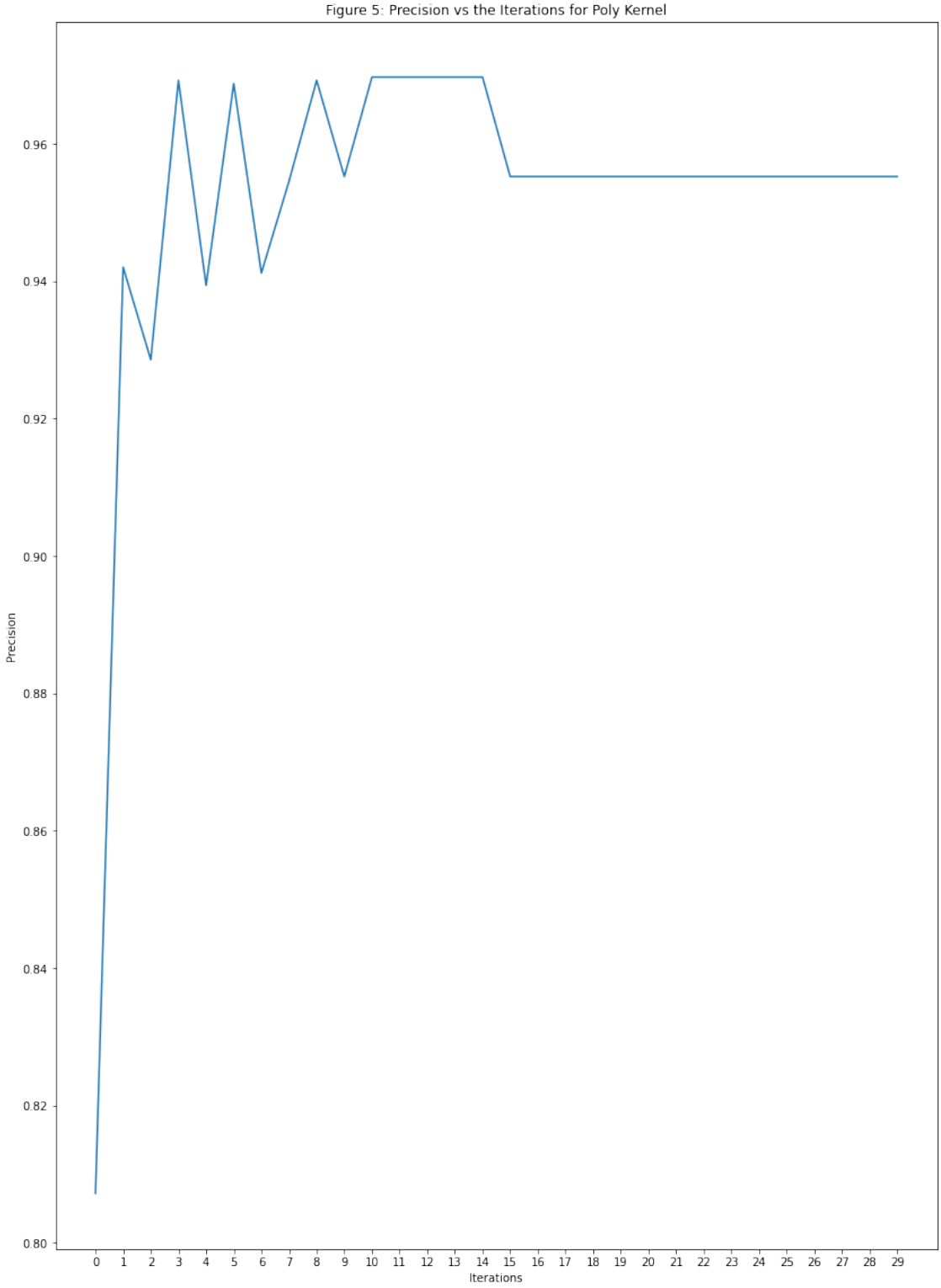
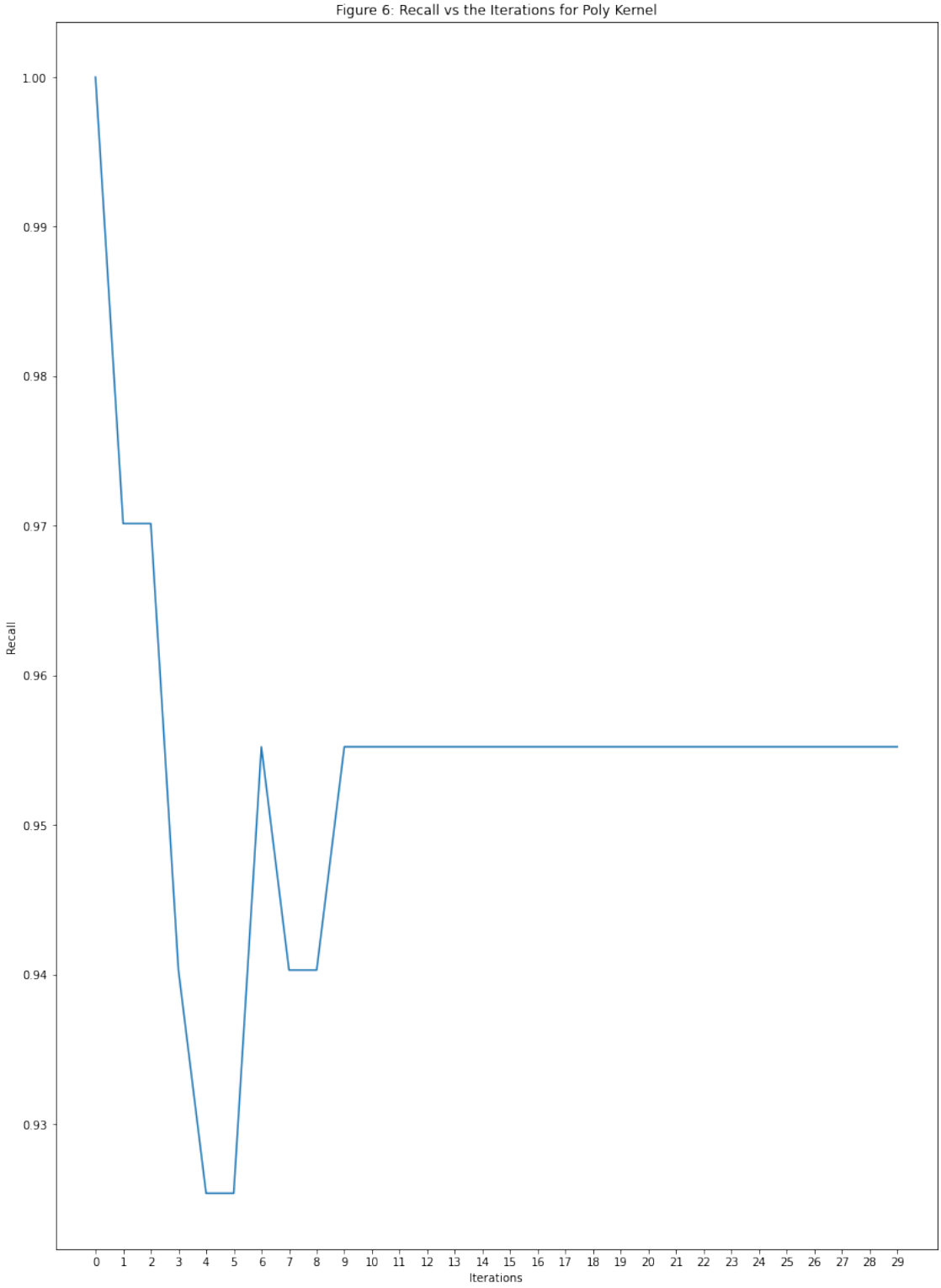


Figure 6: Recall vs the Iterations for Poly Kernel



[17]:
```
# Using an rbf kernal
accuracy = []
precision = []
recall = []
for i in range(30):
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVC(kernel='rbf', C=1E6)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_pred = model.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))

# Plotting the accuracy, precision, and recall against the iterations
plt.figure()
plt.plot(range(30),accuracy)
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Figure 7: Accuracy vs the Iterations for RBF Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),precision)
plt.xlabel('Iterations')
plt.ylabel('Precision')
plt.title('Figure 8: Precision vs the Iterations for RBF Kernel')
plt.xticks(range(30))
plt.show()

plt.figure()
plt.plot(range(30),recall)
plt.xlabel('Iterations')
plt.ylabel('Recall')
plt.title('Figure 9: Recall vs the Iterations for RBF Kernel')
plt.xticks(range(30))
plt.show()
```

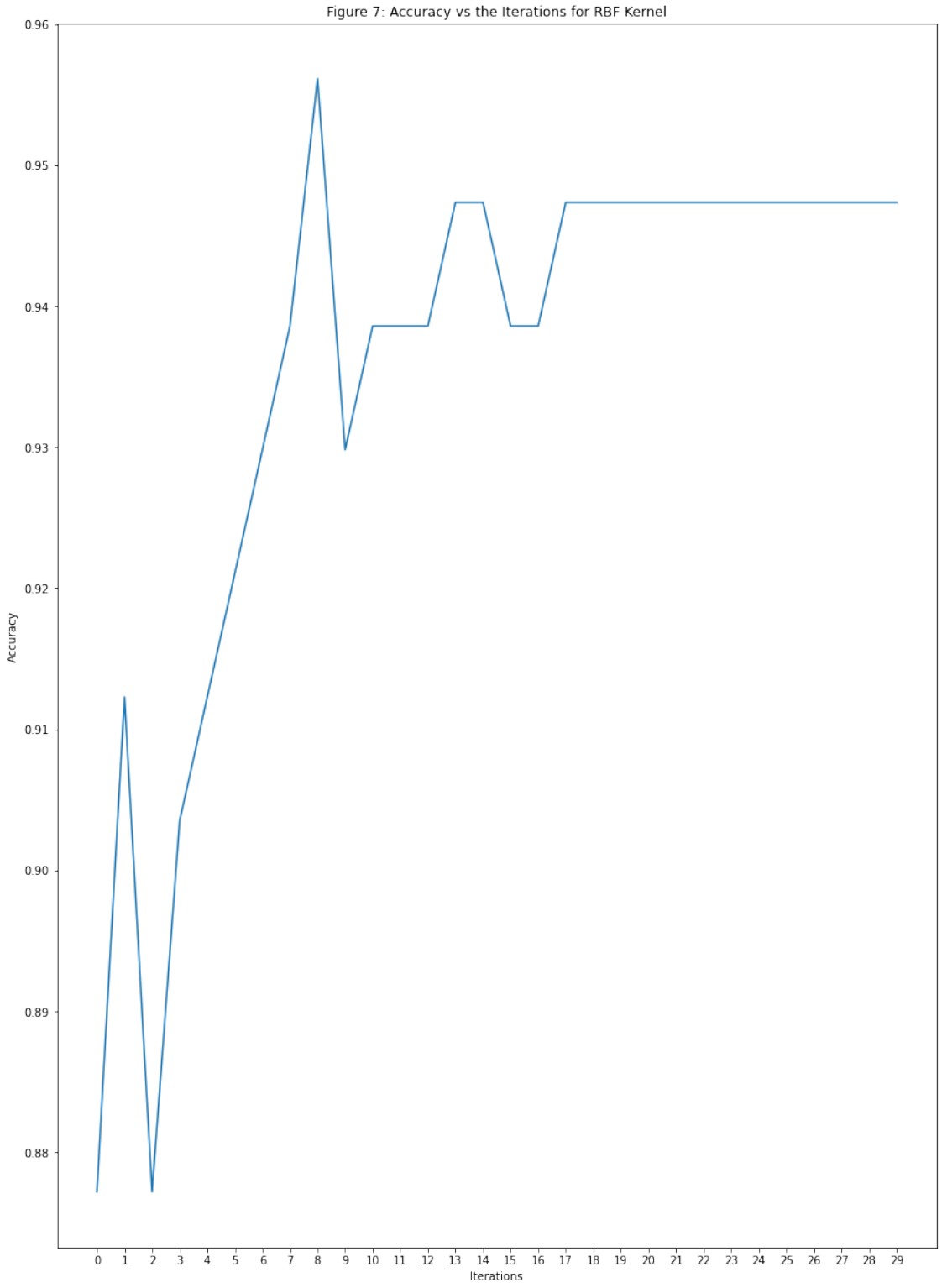


Figure 7: Accuracy vs the Iterations for RBF Kernel
Accuracy
Iterations
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0.89
0.88
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

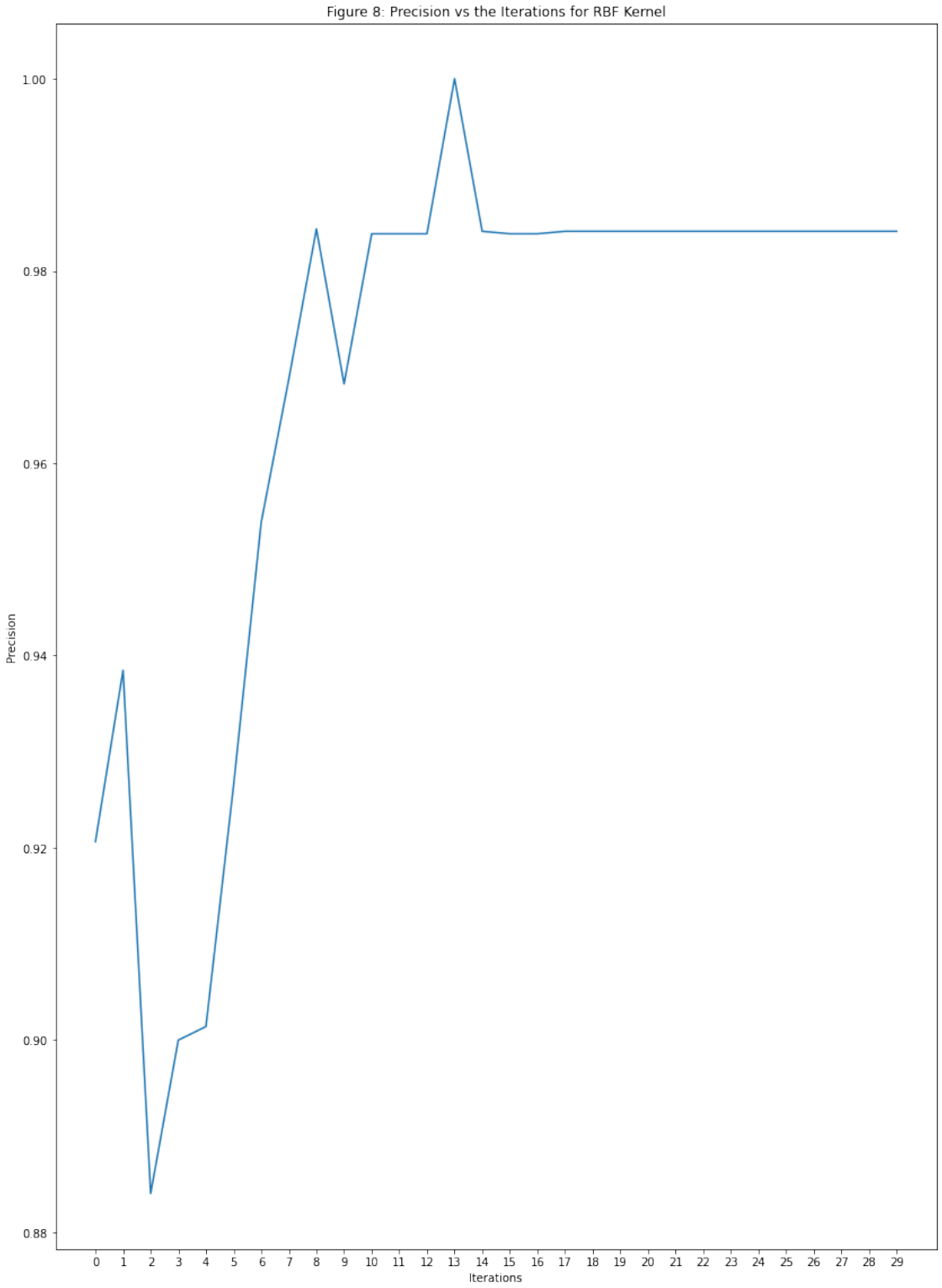
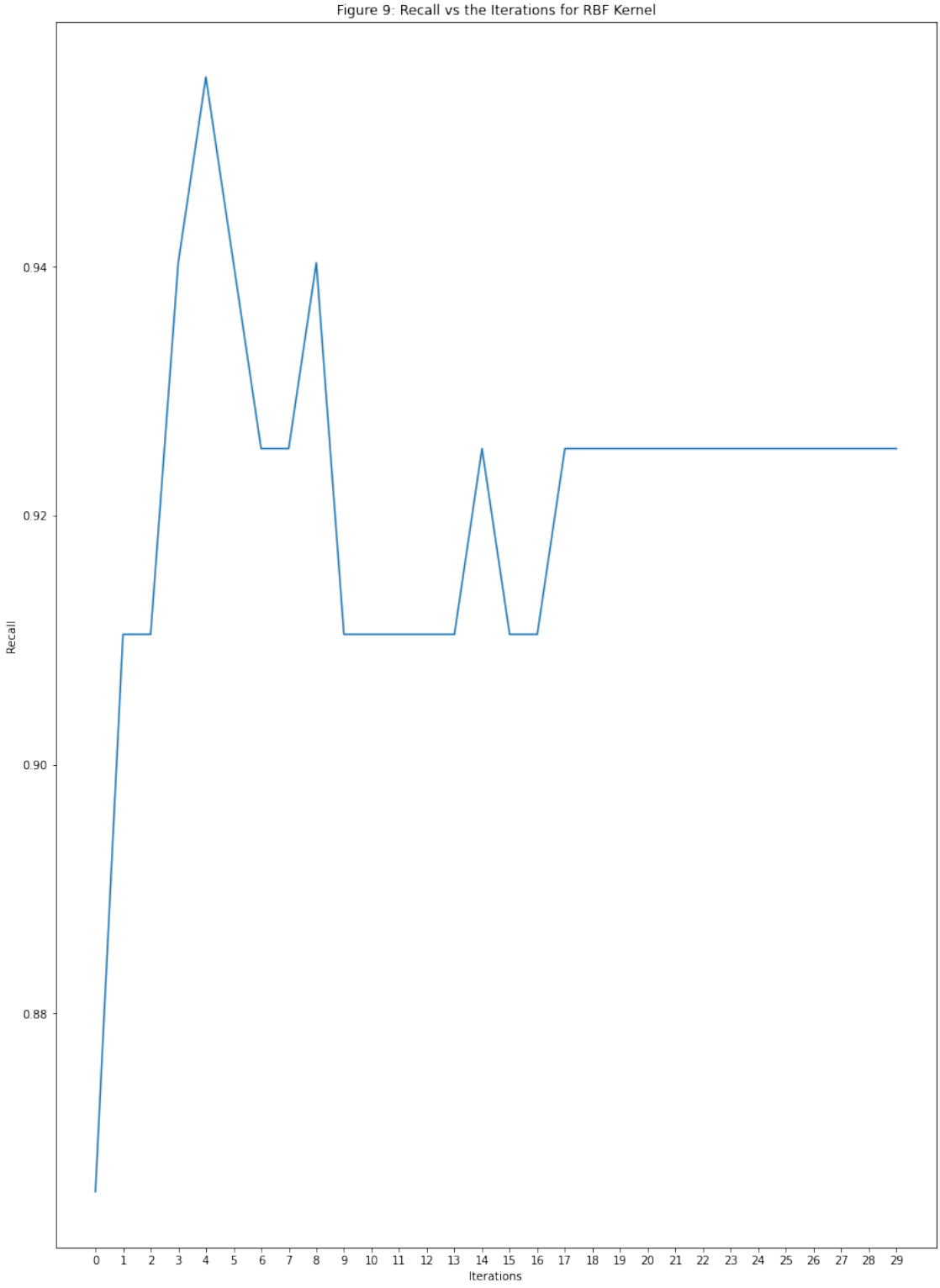


Figure 8: Precision vs the Iterations for RBF Kernel
Precision
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Iterations

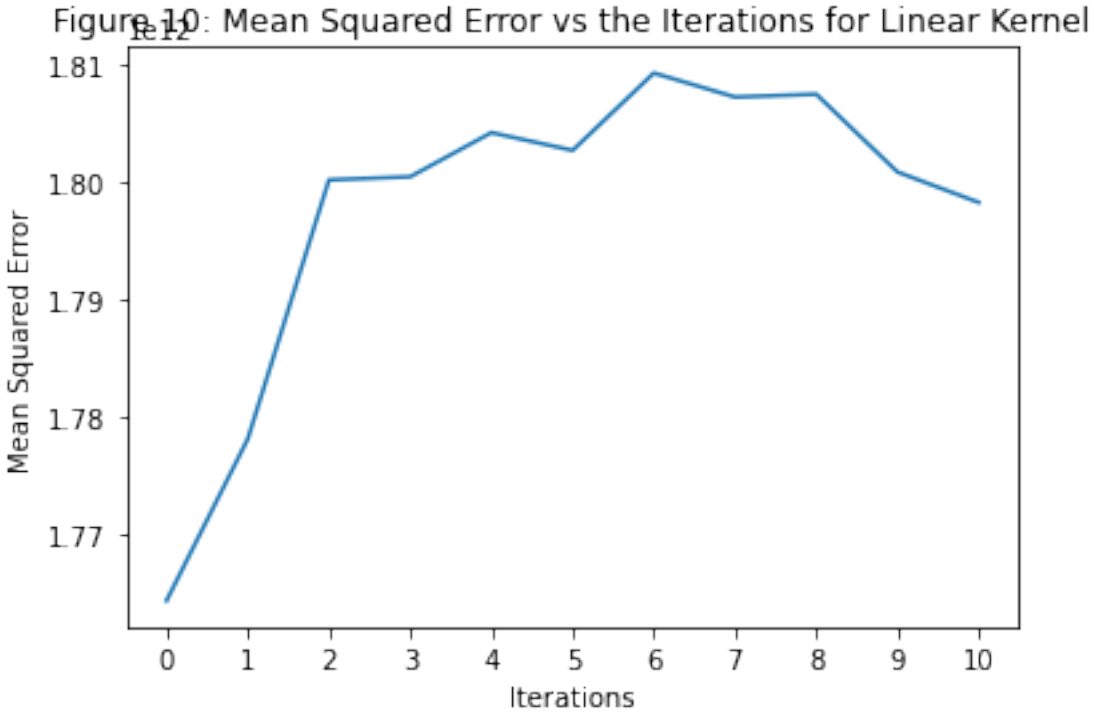
Figure 9: Recall vs the Iterations for RBF Kernel



[8]:
```
# Problem 2
# Importing the dataset
housing = pd.DataFrame(pd.read_csv("Housing.csv"))

# Maping the yes/no inputs to 1 and 0
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
 ↪["price","area","bedrooms","bathrooms","mainroad","guestroom","basement","hotwaterheating",
binarylist =␣
 ↪["mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea",]
housing[binarylist] = housing[binarylist].apply(binary_map)
housing = housing.to_numpy()

Y = housing[:,0]
X = housing[:,1:]

# Scaling the features
sc_X = StandardScaler()
std_X = sc_X.fit_transform(X)
```

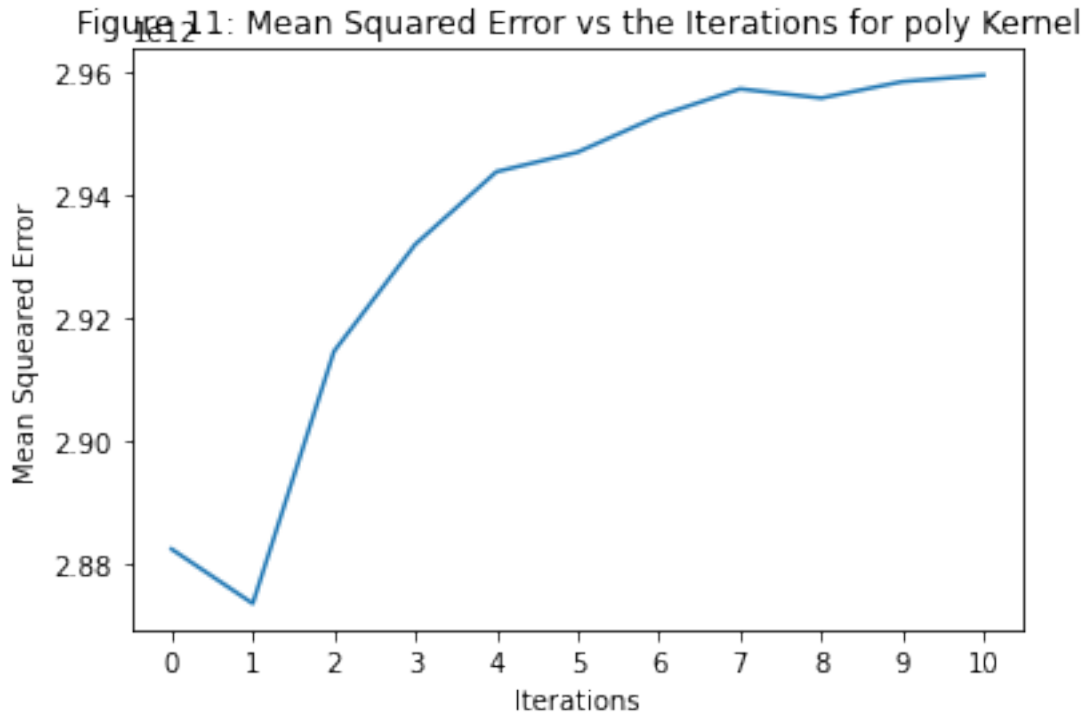
[9]:
```
# Feature extraction using PCA
# Scaling the features
mean_squared_error = []

# Using a linear kernal
for i in range(11):
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVR(kernel='linear', C=1e3)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_lin = model.predict(X_test)

    mean_squared_error.append(metrics.mean_squared_error(Y_test, Y_lin))

# Plotting the mean squared error against the iterations
plt.figure()
plt.plot(range(11),mean_squared_error)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
```

```
plt.title('Figure 10: Mean Squared Error vs the Iterations for Linear Kernel')
plt.xticks(range(11))
plt.show()
```



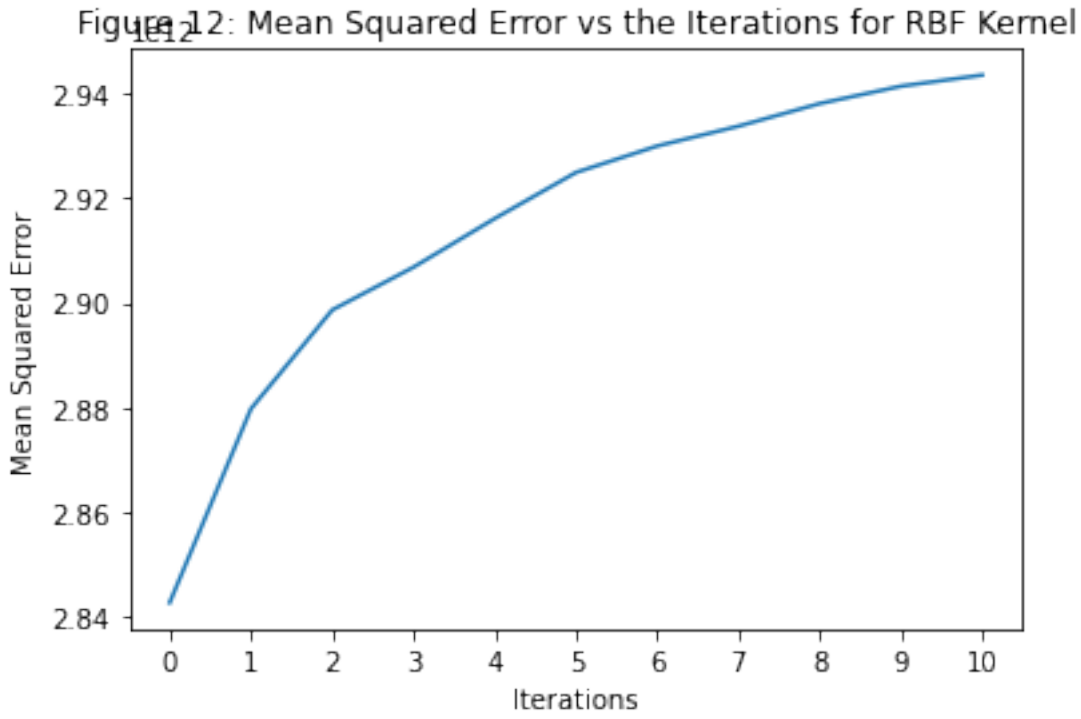
[10]:
```
# Using a poly kernal
mean_squared_error = []
for i in range(11):
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVR(kernel='poly', C=1e3, degree=2)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_poly = model.predict(X_test)

    mean_squared_error.append(metrics.mean_squared_error(Y_test, Y_poly))

# Plotting the mean squared error against the iterations
plt.figure()
```

```
plt.plot(range(11),mean_squared_error)
plt.xlabel('Iterations')
plt.ylabel('Mean Squeared Error')
plt.title('Figure 11: Mean Squared Error vs the Iterations for poly Kernel')
plt.xticks(range(11))

plt.show()
```



[11]:
```
# Using an rbf kernal
mean_squared_error = []
for i in range(11):
    pca = PCA(n_components=i+1)
    principalComponents = pca.fit_transform(std_X)
    principalDf = pd.DataFrame(data = principalComponents)

    # Creating the SVM model and fitting it
    X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y,␣
 ↪test_size = 0.20, random_state=0)
    model = SVR(kernel='rbf', C=1e3, gamma=0.1)
    model.fit(X_train, Y_train)
    # Creating predictions with the test data
    Y_rbf = model.predict(X_test)
```

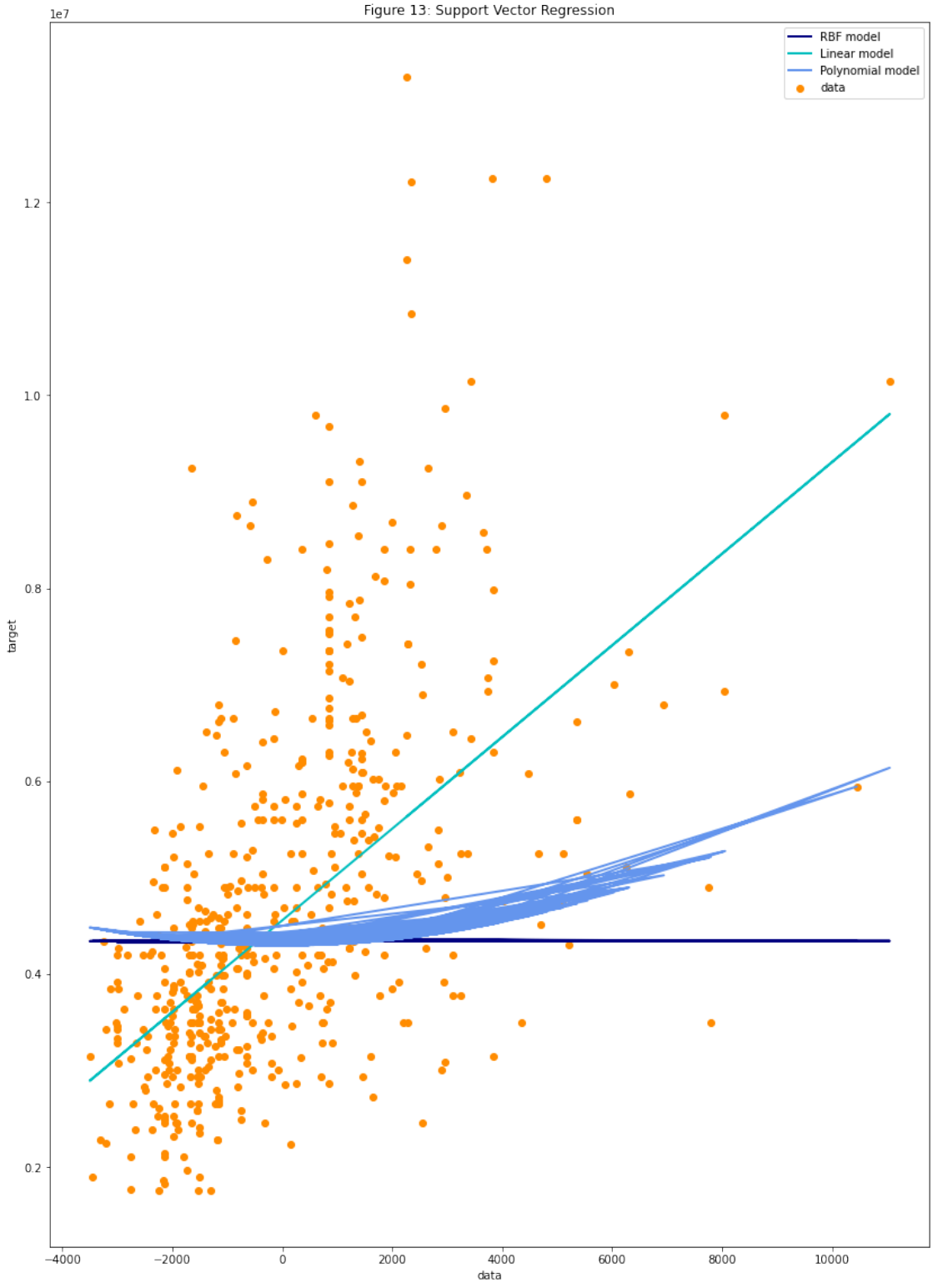
```
    mean_squared_error.append(metrics.mean_squared_error(Y_test, Y_rbf))

# Plotting the mean squared error against the iterations
plt.figure()
plt.rcParams['figure.figsize'] = [14 , 20]


plt.plot(range(11),mean_squared_error)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Figure 12: Mean Squared Error vs the Iterations for RBF Kernel')
plt.xticks(range(11))
plt.show()
```



[12]:
```
# Plotting the regression models for SVR
pca = PCA(n_components=1)
principalComponents = pca.fit_transform(X)
X = pd.DataFrame(data = principalComponents)
X = X.to_numpy()

# Creating the SVM model and fitting it
# X_train, X_test, Y_train, Y_test = train_test_split(principalDf, Y, test_size␣
 ↪= 0.20, random_state=0)
```

```
model = SVR(kernel='linear', C=1e3)
model.fit(X, Y)
Y_lin = model.predict(X)

model = SVR(kernel='poly', C=1e3, degree=2)
model.fit(X, Y)
Y_poly = model.predict(X)

model = SVR(kernel='rbf', C=1e3, gamma=0.1)
model.fit(X, Y)
Y_rbf = model.predict(X)

lw = 2
plt.figure()
plt.scatter(X, Y, color='darkorange', label='data')
plt.plot(X, Y_rbf, color='navy', lw=lw, label='RBF model')
plt.plot(X, Y_lin, color='c', lw=lw, label='Linear model')
plt.plot(X, Y_poly, color='cornflowerblue', lw=lw, label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Figure 13: Support Vector Regression')
plt.legend()
plt.rcParams['figure.figsize'] = [14 , 20]
plt.show()
```

Figure 13: Support Vector Regression



[ ]: