Gabriel Van Dreel
800886655
November 24, 2021
ECGR 4105-C01

Homework 4

The source code developed for homework 4 was uploaded to the Github repository at the following link:

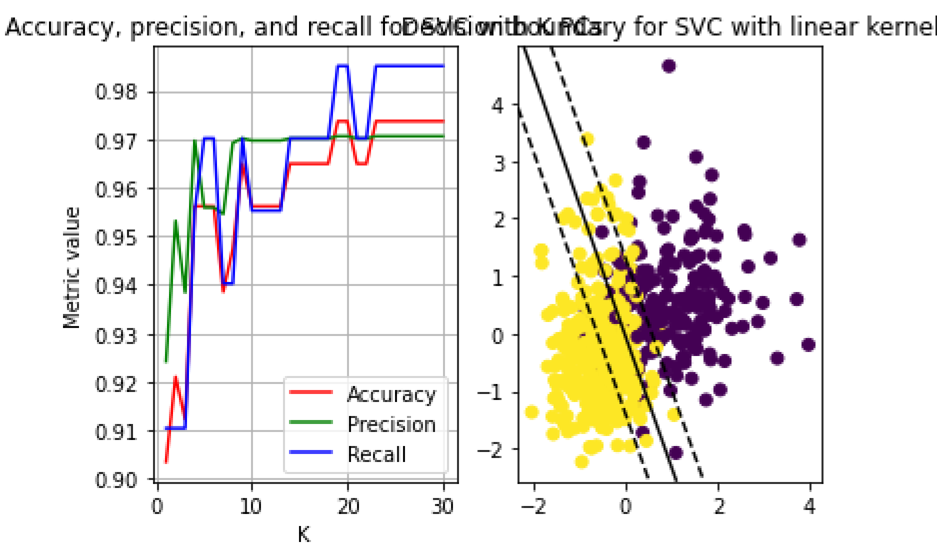
https://github.com/Gabrielvd616/ECGR4105/tree/main/Homework4

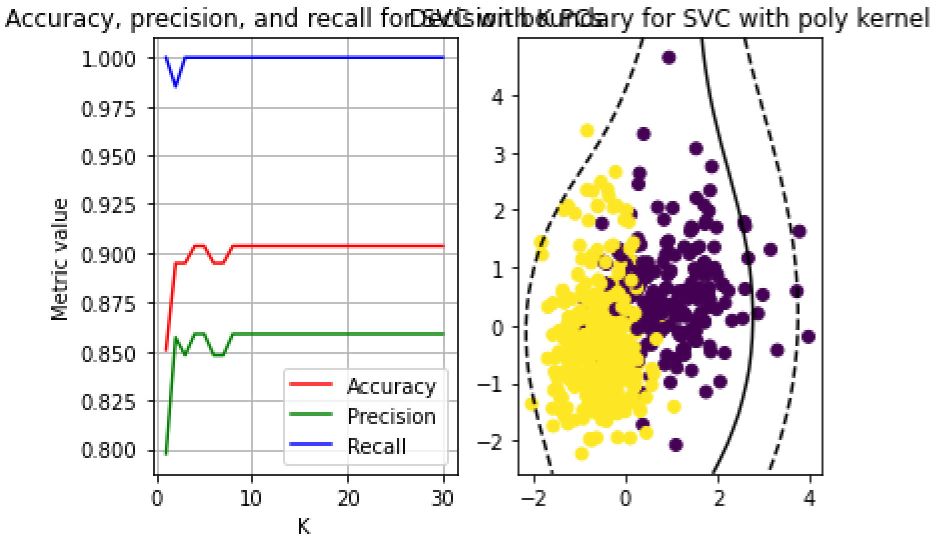
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler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

# Problem 1, Part 1
# Loads breast labelled training data
breast = load_breast_cancer()

# Formats np array since breast object contains separate fields for data and labels
breast_data = breast.data
labels = np.reshape(breast.target, (breast_data.shape[0], 1))
df = pd.DataFrame(np.concatenate([breast_data, labels], axis=1))

n = breast_data.shape[1]
x = df.values[:, :n]
y = df.values[:, n]

# Performs MIN MAX scaling
mms = MinMaxScaler()
x = mms.fit_transform(x)

# Performs standardization
ss = StandardScaler()
x = ss.fit_transform(x)

# Performs PCA on the data
pca = PCA()
pcs = pca.fit_transform(x)

# Performs 80% and 20% split of the labelled data into training and test sets
np.random.seed(0)
x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(pcs, y, train_size=0.8,
                                                            test_size=0.2,
                                                            random_state=np.random)

# Initializes evaluation metrics for SVM classifier model PCA
k = pcs.shape[1]
```

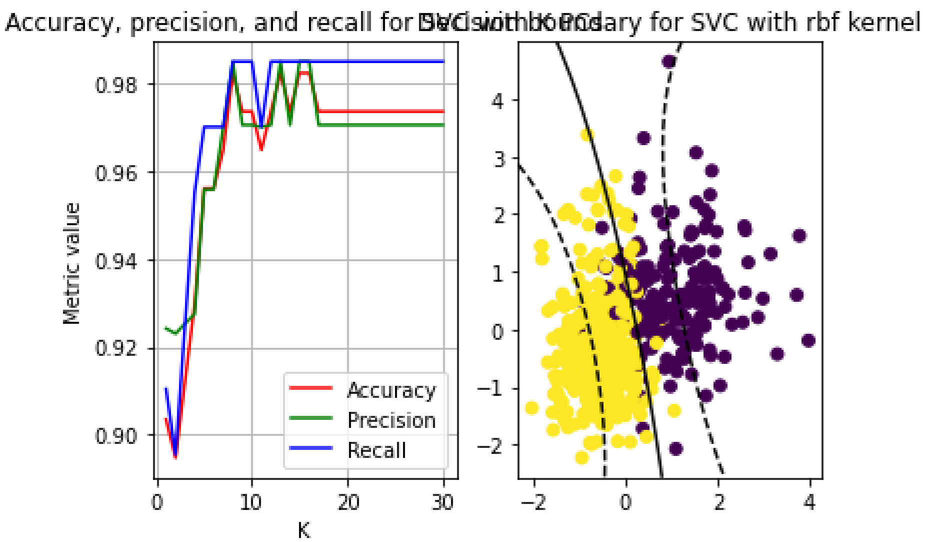
```python
accuracy = np.zeros(k)
precision = np.zeros(k)
recall = np.zeros(k)

# Iteratively evaluates different kernels for SVM classifier with optimal PCA
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
    # Iteratively trains and evaluates model for principal components
    acc_max = 0
    k_opt = 0
    svm_best = None
    for j in range(k):
        # Performs SVM classification by instantiating SVC object with linear kernel
        svm = SVC(kernel=kernels[i])
        svm.fit(x_train_p[:, :j + 1], y_train_p)
        y_pred = svm.predict(x_test_p[:, :j + 1])

        # Evaluates model using accuracy, precision, and recall evaluation metrics
        accuracy[j] = metrics.accuracy_score(y_test_p, y_pred)
        precision[j] = metrics.precision_score(y_test_p, y_pred)
        recall[j] = metrics.recall_score(y_test_p, y_pred)

        if accuracy[j] > acc_max:
            acc_max = accuracy[j]
            k_opt = j + 1
            svm_best = svm

    # Displays optimal K and corresponding accuracy, precision, and recall
    print('Optimal value of K for PCA with {} kernel:'.format(kernels[i]), k_opt)
    print('Accuracy:', acc_max)
    print('Precision:', precision[k_opt - 1])
    print('Recall:', recall[k_opt - 1])
    print()

    # Problem 1, Part 2
    # Creates subplots for metrics and 2D projection of decision boundary
    fig, (ax1, ax2) = plt.subplots(1, 2)

    # Plots accuracy, precision, and recall for varying numbers of PCs
    ax1.plot(np.linspace(1, k, k), accuracy, color='red',
             label='Accuracy')
    ax1.plot(np.linspace(1, k, k), precision, color='green',
             label='Precision')
    ax1.plot(np.linspace(1, k, k), recall, color='blue',
             label='Recall')
    ax1.grid()
    ax1.set_xlabel('K')
    ax1.set_ylabel('Metric value')
    ax1.set_title('Accuracy, precision, and recall for SVC with K PCs')
    ax1.legend()

    # Problem 1, Part 3
    # Plots data points
    ax2.scatter(x[:, 0], x[:, 1], c=y)

    # Creates grid to evaluate model
    xlim = ax2.get_xlim()
    ylim = ax2.get_ylim()
    x_grid = np.linspace(xlim[0], xlim[1], 30)
    y_grid = np.linspace(ylim[0], ylim[1], 30)
```

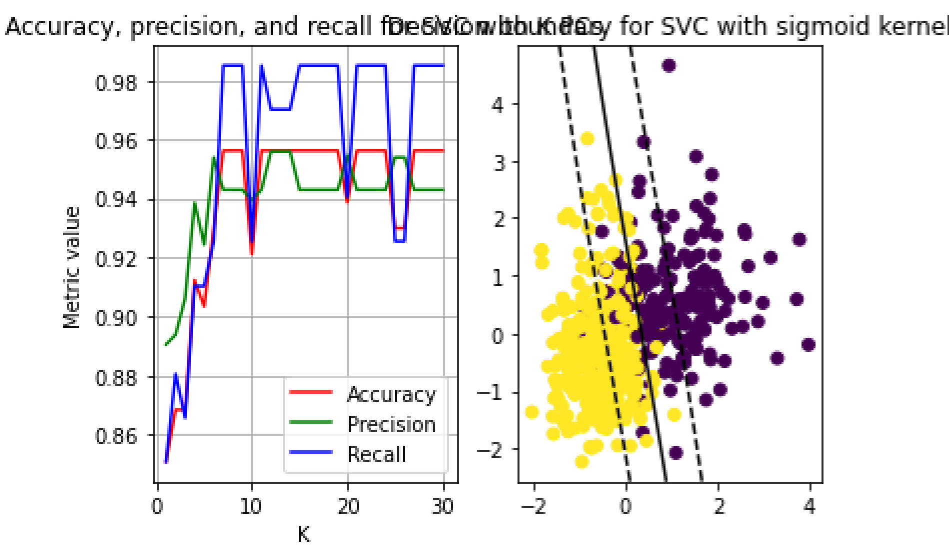
```python
    y_grid, x_grid = np.meshgrid(y_grid, x_grid)
    xy = np.vstack([x_grid.ravel(), np.zeros((k_opt - 2, 900)), y_grid.ravel()]).T
    p = svm_best.decision_function(xy).reshape(x_grid.shape)

    # Plots decision boundary and margins
    ax2.contour(x_grid, y_grid, p, colors='k', levels=[-1, 0, 1],
                linestyles=['--', '-', '--'])

    # Plots support vectors
    ax2.scatter(svm_best.support_vectors_[:, 0], svm_best.support_vectors_[:, 1],
                facecolors='none')
    ax2.set_xlim(xlim)
    ax2.set_ylim(ylim)
    ax2.set_title('Decision boundary for SVC with {} kernel'.format(kernels[i]))
```

```
Optimal value of K for PCA with linear kernel: 19
Accuracy: 0.9736842105263158
Precision: 0.9705882352941176
Recall: 0.9850746268656716

Optimal value of K for PCA with poly kernel: 4
Accuracy: 0.9035087719298246
Precision: 0.8589743589743589
Recall: 1.0

Optimal value of K for PCA with rbf kernel: 8
Accuracy: 0.9824561403508771
Precision: 0.9850746268656716
Recall: 0.9850746268656716

Optimal value of K for PCA with sigmoid kernel: 7
Accuracy: 0.956140350877193
Precision: 0.9428571428571428
Recall: 0.9850746268656716
```







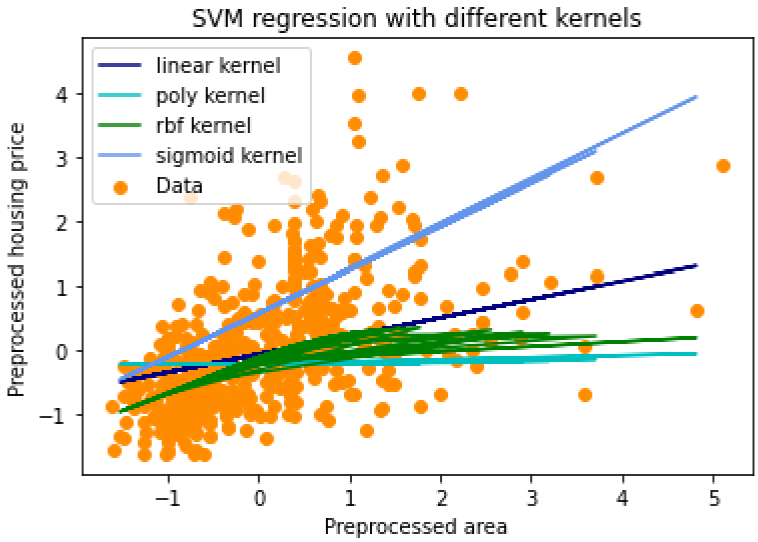


In [2]:

```
# Problem 2, Part 1
# Defines the map function to map strings to numbers in table
def binary_map(x):
    return x.map({'yes': 1, 'no': 0})
```

```python
# Reads labelled training data
housing_data = pd.read_csv(r'Housing.csv')

# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
           'prefarea']

# Applies binary_map function to and selects features from housing_data
housing_data[varlist] = housing_data[varlist].apply(binary_map)
housing_data = housing_data[housing_data.columns[:12]]

# Performs MIN MAX scaling
housing_data = mms.fit_transform(housing_data)

# Performs standardization
housing_data = ss.fit_transform(housing_data)

# Performs 80% and 20% split of the labelled data into training and test sets
x = housing_data[:, 1:12]
y = housing_data[:, 0]
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8,
                                                    test_size=0.2,
                                                    random_state=np.random)

# Fits the SVM regression model with all available kernels
svr_lin = SVR(kernel='linear')
svr_poly = SVR(kernel='poly')
svr_rbf = SVR(kernel='rbf')
svr_sig = SVR(kernel='sigmoid')

m = x_test.shape[0]
x_test_reduced = np.hstack([x_test[:, 0].reshape(m, 1), np.zeros((m, 10))])
y_lin = svr_lin.fit(x_train, y_train).predict(x_test_reduced)
y_poly = svr_poly.fit(x_train, y_train).predict(x_test_reduced)
y_rbf = svr_rbf.fit(x_train, y_train).predict(x_test_reduced)
y_sig = svr_sig.fit(x_train, y_train).predict(x_test_reduced)

a = x_test[:, 0].reshape(m, 1)

# Plots the SVM regressions and data points
plt.figure(5)
plt.scatter(x[:, 0], y, color='darkorange', label='Data')
plt.plot(x_test[:, 0], y_lin, color='navy', label='linear kernel')
plt.plot(x_test[:, 0], y_poly, color='c', label='poly kernel')
plt.plot(x_test[:, 0], y_rbf, color='g', label='rbf kernel')
plt.plot(x_test[:, 0], y_sig, color='cornflowerblue', label='sigmoid kernel')
plt.xlabel('Preprocessed area')
plt.ylabel('Preprocessed housing price')
plt.title('SVM regression with different kernels')
plt.legend()
```

Out[2]: <matplotlib.legend.Legend at 0xd2193d0>


SVM regression with different kernels
linear kernel
poly kernel
rbf kernel
sigmoid kernel
Data
Preprocessed housing price
Preprocessed area
4
3
2
1
0
−1
−1
0
1
2
3
4
5