Claude Shyaka

ID#: 801326243

Homework 4: SVM and SVR Models

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.svm import SVC, SVR
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, Ridge
from sklearn import metrics

# Seed for random state
seed = 42
```

Problem 1

In [2]:
```
# Load the cancer dataset into a pandas dataframe
breast_cancer = load_breast_cancer()
breast_cancer_data = breast_cancer.data
breast_cancer_labels = breast_cancer.target
labels = np.reshape(breast_cancer_labels, (569,1))
final_breast_cancer_data = np.concatenate([breast_cancer_data, labels], axis=1)
breast_cancer_dataset = pd.DataFrame(final_breast_cancer_data)

# Add feature names to the dataframe
features = breast_cancer.feature_names
features_labels = np.append(features, 'label')
breast_cancer_dataset.columns = features_labels

# Replace labels names
breast_cancer_dataset['label'].replace(0, 'Benign', inplace=True)
breast_cancer_dataset['label'].replace(1, 'Malignant', inplace=True)

# Rename labels
breast_cancer_dataset['label'].replace('Benign', 0, inplace=True)
breast_cancer_dataset['label'].replace('Malignant', 1, inplace=True)

# Preview of the dataset
breast_cancer_dataset.tail()
```

Out[2]:

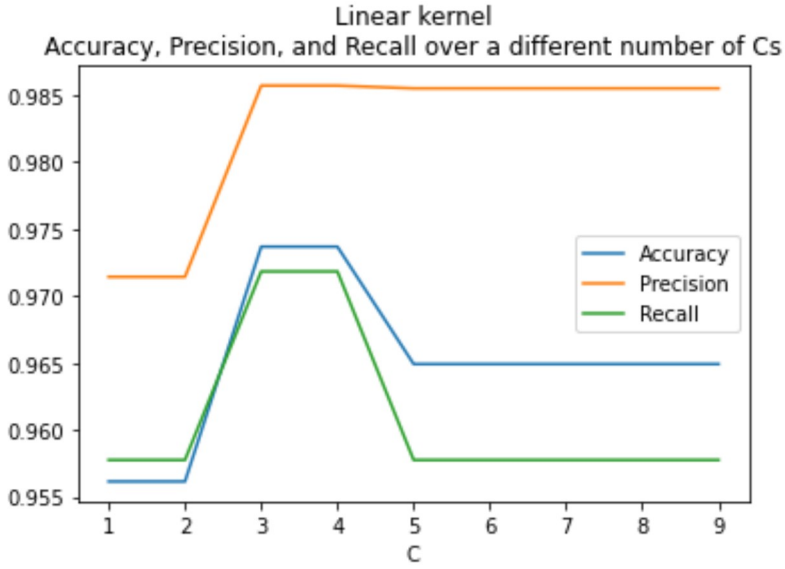
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1

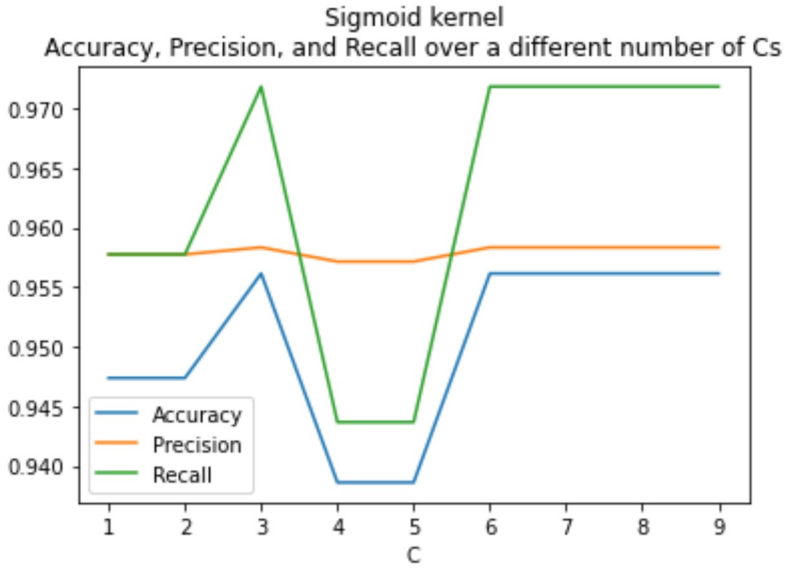
5 rows × 31 columns

In [3]:
```
# Load features and labels
X = breast_cancer_dataset.iloc[:, :30].values
Y = breast_cancer_dataset.iloc[:, 30].values

# Create a scaler object
# Fit the scaler on the training data and transform
sc = StandardScaler()
X_scaled = sc.fit_transform(X)
```

In [4]:
```
def OptimalRegularizationStrength(kernel, C_array):
    # Split the data into training and test set.
    # Train set is 80% of the data and the test set is 20%
    X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size =

    acc_list = []
    prec_list = []
    rec_list = []
    for c in C_array:
        # Linear support vector classifier
        clf = SVC(kernel=kernel, C=c)
        clf.fit(X_train, Y_train)
        Y_pred = clf.predict(X_test)
        acc_list.append(metrics.accuracy_score(Y_test, Y_pred))
        prec_list.append(metrics.precision_score(Y_test, Y_pred))
        rec_list.append(metrics.recall_score(Y_test, Y_pred))

    plt.plot(C_array, acc_list, label="Accuracy")
    plt.plot(C_array, prec_list, label="Precision")
    plt.plot(C_array, rec_list, label="Recall")
    plt.xlabel("C")
    plt.title(kernel.capitalize() + " kernel\nAccuracy, Precision, and Recall ov
    plt.legend()
    plt.show()
```

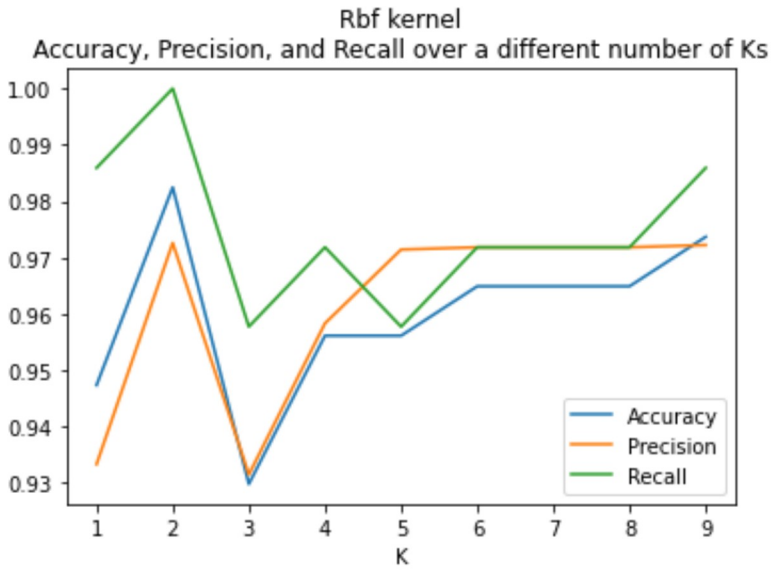
In [5]:
```
# Extract the optimal regularization strength
C_array = np.arange(1., 10., 1)
OptimalRegularizationStrength("linear", C_array=C_array)
OptimalRegularizationStrength("sigmoid", C_array=C_array)
```

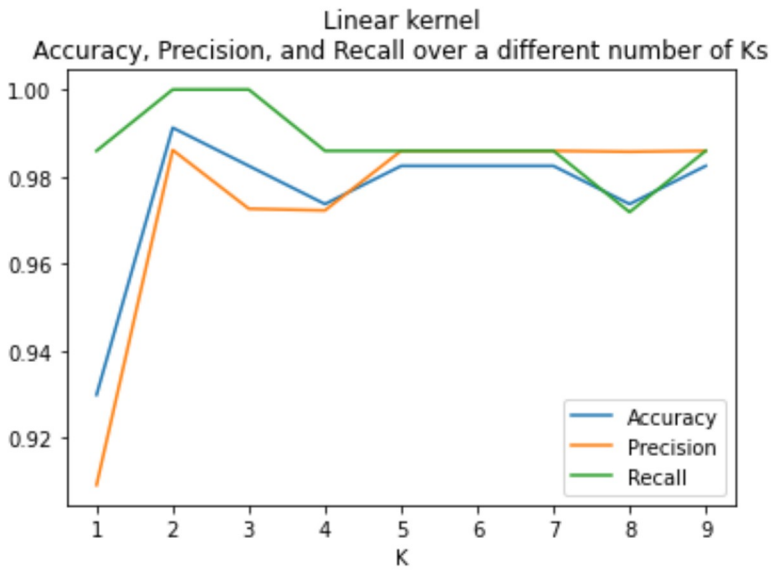

Linear kernel
Accuracy, Precision, and Recall over a different number of Cs
0.985
0.980
0.975
0.970
0.965
0.960
0.955
1
2
3
4
5
6
7
8
9
C
Accuracy
Precision
Recall

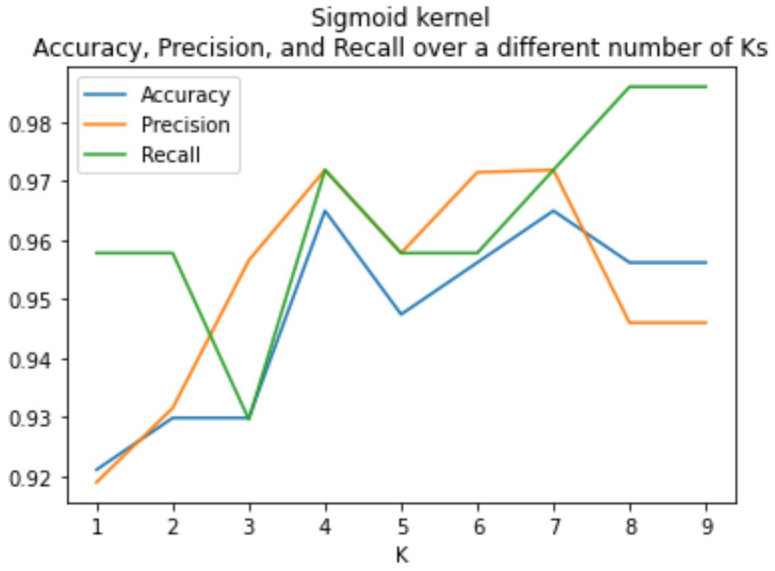

Sigmoid kernel
Accuracy, Precision, and Recall over a different number of Cs
0.970
0.965
0.960
0.955
0.950
0.945
0.940
1
2
3
4
5
6
7
8
9
C
Accuracy
Precision
Recall

In [6]:
```
def TrainSVCWithPCA(X, Y, n_trainings, kernel, seed):
    '''
    Using PCA feature extraction.
    Training a Classifier n_trainings independent times.
    '''
    acc_list = []
    prec_list = []
    rec_list = []
    K_vals = list(range(1, n_trainings))
    for k in K_vals:

        pca = PCA(n_components=k)
        principalComponents = pca.fit_transform(X)

        # Split the data into training and test set.
        # Train set is 80% of the data and the test set is 20%
        X_train, X_test, Y_train, Y_test = train_test_split(principalComponents,

        # Linear support vector classifier
        clf = SVC(kernel=kernel, C=3)

        clf.fit(X_train, Y_train)
        Y_pred = clf.predict(X_test)
        acc_list.append(metrics.accuracy_score(Y_test, Y_pred))
        prec_list.append(metrics.precision_score(Y_test, Y_pred))
        rec_list.append(metrics.recall_score(Y_test, Y_pred))

    plt.plot(K_vals, acc_list, label="Accuracy")
    plt.plot(K_vals, prec_list, label="Precision")
    plt.plot(K_vals, rec_list, label="Recall")
    plt.xlabel("K")
    plt.title(kernel.capitalize() + " kernel\nAccuracy, Precision, and Recall ov
    plt.legend()
    plt.show()
```

In [7]:
```
TrainSVCWithPCA(X_scaled, Y, 10, "rbf", seed)
TrainSVCWithPCA(X_scaled, Y, 10, "linear", seed)
TrainSVCWithPCA(X_scaled, Y, 10, "sigmoid", seed)
```







Problem 2

In [8]:
```
# Load housing dataset
housing_dataset = pd.read_csv('./data/Housing.csv', delimiter=',')

# Prepare the dataset.

# List of variable to map to numerical values.
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'aircondition

# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

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

# preview the dataset
housing_dataset.head()
```

Out[8]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterl
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

In [9]:
```
# Extract the desired features
filter = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', '

housing_filtered = housing_dataset[filter]
housing_filtered.head()
```

Out[9]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	ai
0	7420	4	2	3	1	0	0	0	
1	8960	4	4	4	1	0	0	0	
2	9960	3	2	2	1	0	1	0	
3	7500	4	2	2	1	0	1	0	
4	7420	4	1	2	1	1	1	0	

In [10]:
```
# array of data
data = housing_filtered.values
```

In [11]:
```
# Train SVR models without using PCA
def TrainSVRModel(data, kernel, C=1., gamma='scale', degree=3):

    # Split the data in train and test set
    data_train, data_test = train_test_split(data, train_size=0.8, test_size=0.2

    # Apply the MinMaxScaler to the Datasets
    sc = MinMaxScaler()
    data_train_scaled = sc.fit_transform(data_train)
    data_test_scaled = sc.transform(data_test)

    # Get the training feature and labels
    X_train, Y_train = data_train_scaled[:, :11], data_train_scaled[:, 11]

    # Get the testing features and labels
    X_test, Y_test = data_test_scaled[:, :11], data_test_scaled[:, 11]

    # Support Vector Regression model
    clf = SVR(kernel=kernel, C=C, gamma=gamma, degree=degree)
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_test)
    return metrics.mean_squared_error(Y_test, Y_pred)
```

In [12]:
```
# Train LR to compare with the SVR results
def TrainLRModels(data):

    # Split the data in train and test set
    data_train, data_test = train_test_split(data, train_size=0.8, test_size=0.2

    # Apply the MinMaxScaler to the Datasets
    sc = MinMaxScaler()
    data_train_scaled = sc.fit_transform(data_train)
    data_test_scaled = sc.transform(data_test)

    # Get the training feature and labels
    X_train, Y_train = data_train_scaled[:, :11], data_train_scaled[:, 11]

    # Get the testing features and labels
    X_test, Y_test = data_test_scaled[:, :11], data_test_scaled[:, 11]

    # Ridge regression model
    clf = Ridge(alpha=5)
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_test)
    return metrics.mean_squared_error(Y_test, Y_pred)
```

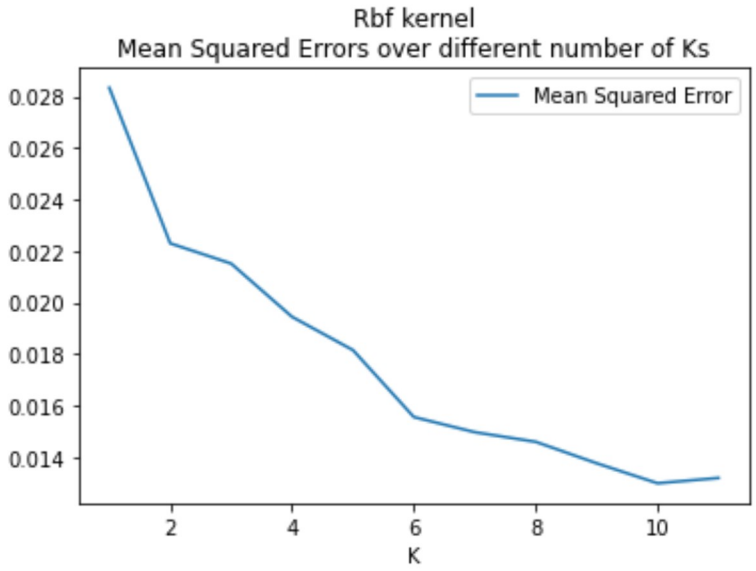
In [13]:
```
svr_mean_squared_error = TrainSVRModel(data, "rbf", C=1e1, gamma=0.01)
print("Mean Squared Error from SVR Model:", svr_mean_squared_error)
ridge_mean_squared_error = TrainLRModels(data)
print("Mean Squared Error from Ridge Model:", ridge_mean_squared_error)
print("Percent difference btn SVR and Ridge:", np.abs((svr_mean_squared_error-ri
```

```
Mean Squared Error from SVR Model: 0.015616332188484048
Mean Squared Error from Ridge Model: 0.01712989577799997
Percent difference btn SVR and Ridge: 8.835801508260198
```

In [14]:
```
def TrainSVRWithPCA(data, n_trainings, seed, kernel, C=1., gamma='scale', degree
    '''
    Using PCA feature extraction.
    Training a Classifier n_trainings independent times.
    '''

    # Apply the MinMaxScaler to the Datasets
    sc = MinMaxScaler()
    data_scaled = sc.fit_transform(data)

    X = data_scaled[:, :11]
    Y = data_scaled[:, 11]

    msr_list = []
    K_vals = list(range(1, n_trainings+1))
    for k in K_vals:

        pca = PCA(n_components=k)
        principalComponents = pca.fit_transform(X)

        # Split the data into training and test set.
        # Train set is 80% of the data and the test set is 20%
        X_train, X_test, Y_train, Y_test = train_test_split(principalComponents,

        # Linear support vector classifier
        clf = SVR(kernel=kernel, C=C, gamma=gamma, degree=degree)
        clf.fit(X_train, Y_train)
        Y_pred = clf.predict(X_test)
        msr_list.append(metrics.mean_squared_error(Y_test, Y_pred))

    print(msr_list)
    plt.plot(K_vals, msr_list, label="Mean Squared Error")
    plt.xlabel("K")
    plt.title(kernel.capitalize() + " kernel\nMean Squared Errors over different
    plt.legend()
    plt.show()
```

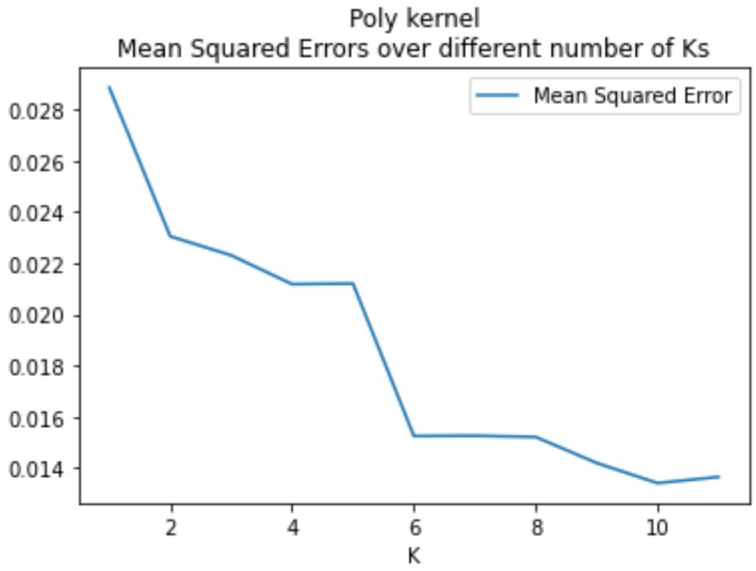
In [15]:
```
TrainSVRWithPCA(data, 11, seed, 'rbf', C=1e2, gamma=0.01)
```

```
[0.02833474958730169, 0.02230060283253544, 0.021516027021625053, 0.0194491745768
46716, 0.018158674284872202, 0.015561400471760519, 0.014974070732218153, 0.01460
1099542151134, 0.013769637560242055, 0.012994006287783667, 0.013197250125136437]
```

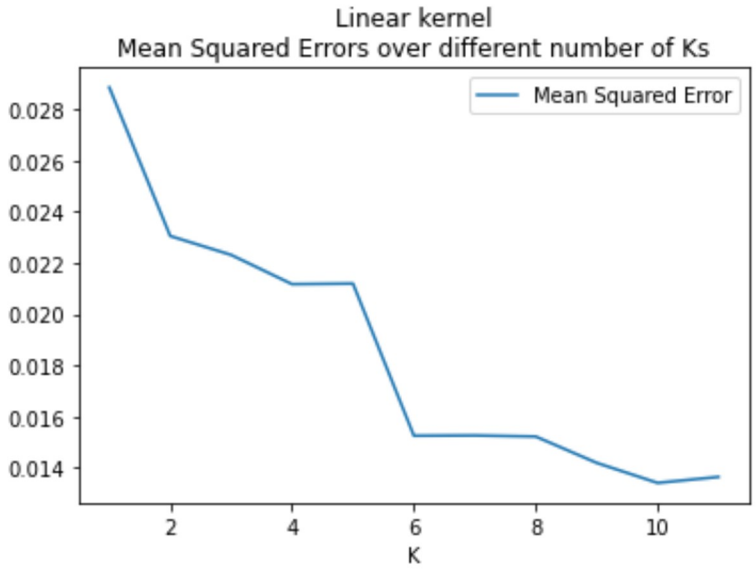


In [16]:
```
TrainSVRWithPCA(data, 11, seed, 'poly', C=1e2, degree=1)
TrainSVRWithPCA(data, 11, seed, 'linear', C=1e2)
```

```
[0.028853091908884205, 0.023049599706387088, 0.022308888977913704, 0.02118332268
169573, 0.02120595889902835, 0.0152551805908508 5, 0.015266543708707735, 0.015217
996148654522, 0.01420399979516002, 0.013417311500902949, 0.013657262359738314]
```




```
[0.028853306330748205, 0.023052746334462523, 0.022314977012825327, 0.02117330983
043208, 0.021195540013835577, 0.01525726364314325, 0.015267875788872157, 0.01522
5004445733742, 0.014197900443051577, 0.013415523542261968, 0.01364730755661225]
```



In [ ]: