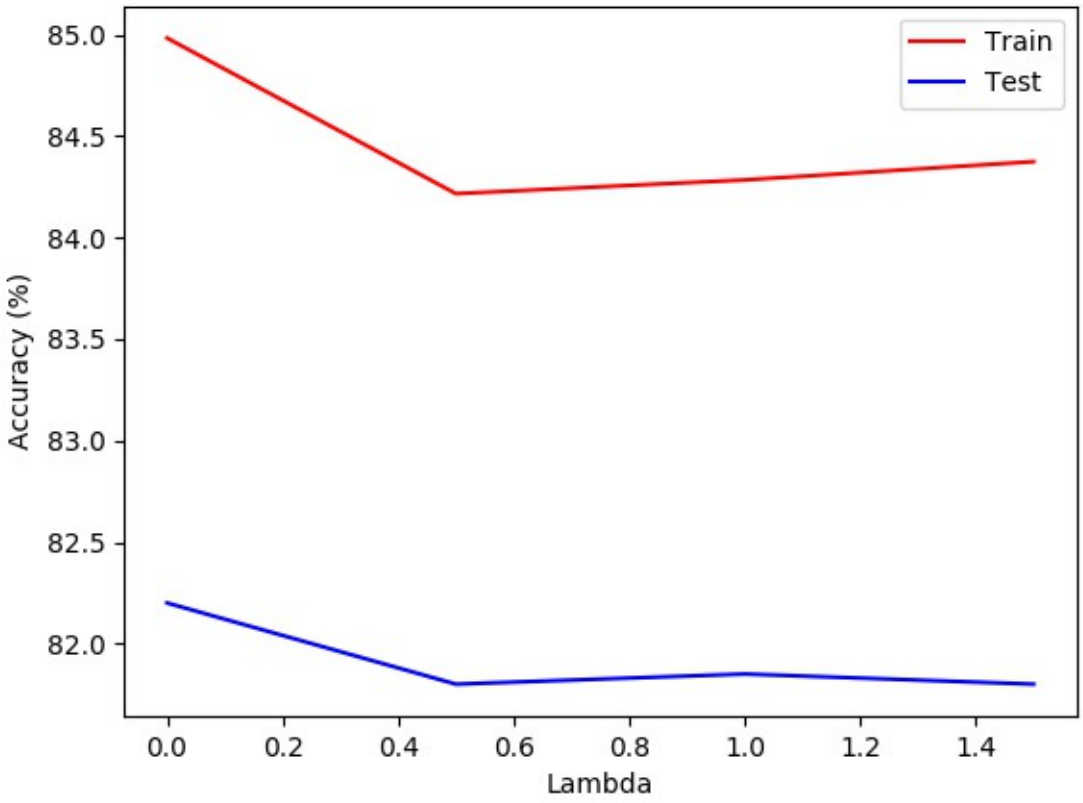
Introduction

The goal of this project was to design a machine learning system for Landsat Multi-Spectral Scanner image data classification. The system works with multi-spectral values of pixels in 3x3 neighborhoods in a satellite image and predicts class for the central pixel in each neighborhood. There are 7 classes describing different types of soil visible in the picture but the dataset does not contain records with class number 6 assigned to them. The system is based on three widely-used machine learning classification algorithms: multi-class logistic regression, neural network and support vector machines. I chose these three because they are the most popular classification algorithms, very flexible and able to deal with different machine learning problems very well. In the dataset description there is an information that cross-validation should not be used with this dataset so the models were only trained and tested, without cross-validation.

Results

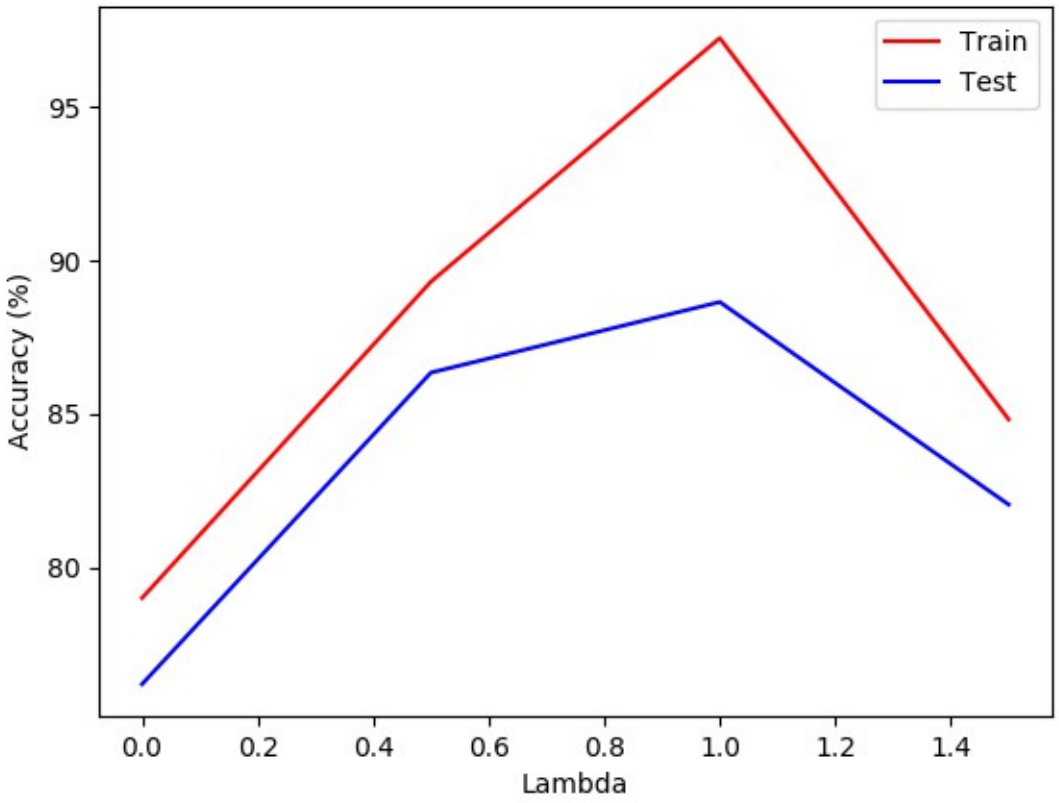
The first algorithm used was multi-class logistic regression with one-vs-all approach. Performance of the model was evaluated using accuracy values, computed for model trained and tested with different values of regularization parameter lambda. The obtained results show that for this dataset one can obtain best results both for training and test set when lambda is set to 0 (no regularization).

Accuracy training (lambda = 0.0) = 84.9830890643%
Accuracy test (lambda = 0.0) = 82.2%
Accuracy training (lambda = 0.5) = 84.2164599775%
Accuracy test (lambda = 0.5) = 81.8%
Accuracy training (lambda = 1.0) = 84.2841037204%
Accuracy test (lambda = 1.0) = 81.85%
Accuracy training (lambda = 1.5) = 84.3742953777%
Accuracy test (lambda = 1.5) = 81.8%



The second algorithm used was a three-layer neural network with random initialization of weights, one-hot encoding for labels, logistic-regression cost function and sigmoid activation function. Back-propagation was implemented using gradient descent. Performance of the model was evaluated using accuracy values, computed for model trained and tested with different values of regularization parameter lambda. The obtained results show that for this dataset one can obtain best results both for training and test set when lambda is set to 1.

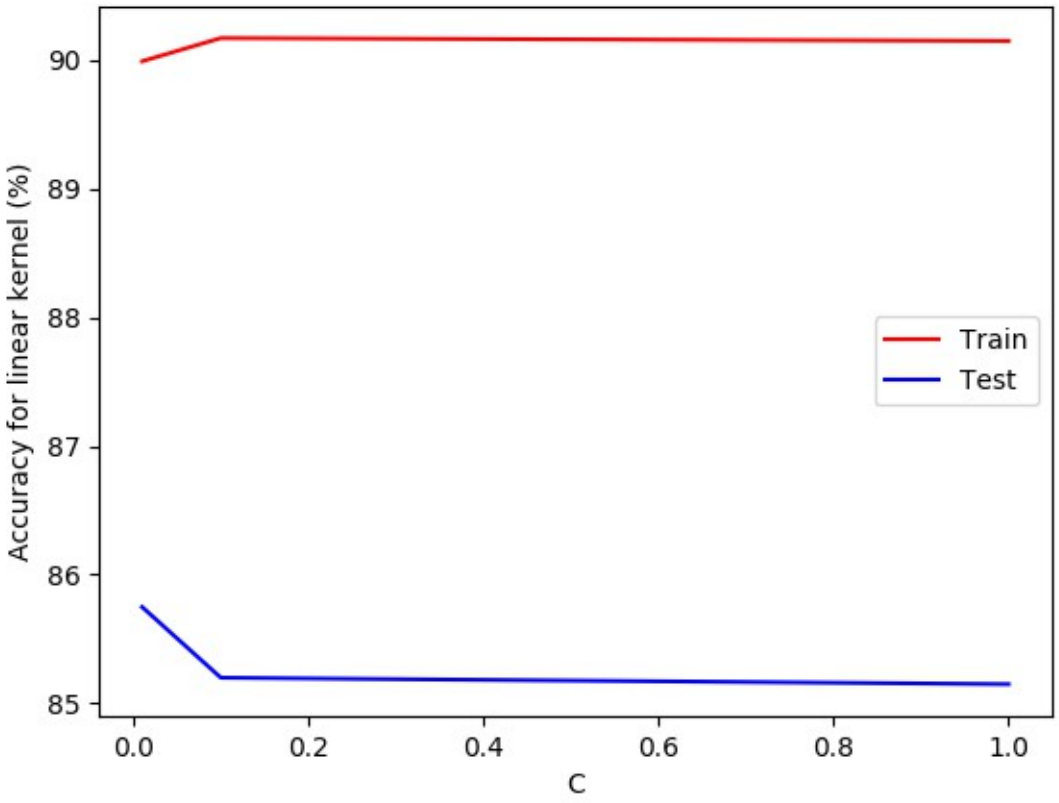
Accuracy train for lambda = 0.0 = 79.00789177%
Accuracy test for lambda = 0.0 = 76.2%
Accuracy train for lambda = 0.5 = 89.3122886133%
Accuracy test for lambda = 0.5 = 86.35%
Accuracy train for lambda = 1.0 = 97.2491544532%
Accuracy test for lambda = 1.0 = 88.65%
Accuracy train for lambda = 1.5 = 84.825253664%
Accuracy test for lambda = 1.5 = 82.05%



The third algorithm used was support vector machines implemented in scikit-learn module. Performance of the model was evaluated using accuracy values, computed for model trained and tested both with linear and RBF kernel and different values of C and sigma. The obtained results show that for linear kernel one can obtain the best results for test set when C is set to 0.01 at the expense of slightly worse performance for training set. For higher values of C the model becomes slightly overfitted, achieving better performance for training set than for test set. For RBF kernel one can obtain the best results for test set when C is set to 1 and sigma is set to 100. In order for RBF kernel to perform well on this dataset both values of C and sigma need to be large. When C is equal to 1, RBF kernel performs very well on training set (even 100% accuracy) but for low values of sigma the model gets extremely overfitted (around 23% accuracy on test set). When C is equal to 0.01 or 0.1 the model generally underfits the data, unless the sigma value is very large when it performs reasonably well.

Train accuracy for SVM with linear kernel for C = 0.01 = 89.9887260428%
Test accuracy for SVM with linear kernel for C = 0.01 = 85.75%
Train accuracy for SVM with RBF kernel for C = 0.01 and sigma = 0.01 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.01 and sigma = 0.01 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.01 and sigma = 0.1 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.01 and sigma = 0.1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.01 and sigma = 1 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.01 and sigma = 1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.01 and sigma = 10 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.01 and sigma = 10 = 23.05%

Train accuracy for SVM with RBF kernel for C = 0.01 and sigma = 100 = 79.3235625705%
Test accuracy for SVM with RBF kernel for C = 0.01 and sigma = 100 = 77.05%
Train accuracy for SVM with linear kernel for C = 0.1 = 90.1691093574%
Test accuracy for SVM with linear kernel for C = 0.1 = 85.2%
Train accuracy for SVM with RBF kernel for C = 0.1 and sigma = 0.01 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.1 and sigma = 0.01 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.1 and sigma = 0.1 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.1 and sigma = 0.1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.1 and sigma = 1 = 24.1713641488%
Test accuracy for SVM with RBF kernel for C = 0.1 and sigma = 1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 0.1 and sigma = 10 = 43.4272829763%
Test accuracy for SVM with RBF kernel for C = 0.1 and sigma = 10 = 39.75%
Train accuracy for SVM with RBF kernel for C = 0.1 and sigma = 100 = 86.5614430665%
Test accuracy for SVM with RBF kernel for C = 0.1 and sigma = 100 = 84.85%
Train accuracy for SVM with linear kernel for C = 1 = 90.1465614431%
Test accuracy for SVM with linear kernel for C = 1 = 85.15%
Train accuracy for SVM with RBF kernel for C = 1 and sigma = 0.01 = 100.0%
Test accuracy for SVM with RBF kernel for C = 1 and sigma = 0.01 = 23.05%
Train accuracy for SVM with RBF kernel for C = 1 and sigma = 0.1 = 100.0%
Test accuracy for SVM with RBF kernel for C = 1 and sigma = 0.1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 1 and sigma = 1 = 100.0%
Test accuracy for SVM with RBF kernel for C = 1 and sigma = 1 = 23.05%
Train accuracy for SVM with RBF kernel for C = 1 and sigma = 10 = 99.8196166855%
Test accuracy for SVM with RBF kernel for C = 1 and sigma = 10 = 72.55%
Train accuracy for SVM with RBF kernel for C = 1 and sigma = 100 = 89.718151071%
Test accuracy for SVM with RBF kernel for C = 1 and sigma = 100 = 88.15%



Summary

The obtained results show that for this problem the best choice would be a neural network with regularization parameter lambda set to 1, achieving 88.65% accuracy for test set. In general neural networks are much more flexible than logistic regression because they consist of multiple layers and involve much more parameters in their computations, that results in getting better fit for a given dataset. Both support vector machines and neural networks can model high-dimensional data and perform well even for very complex problems.

Code

Logistic regression

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
import pandas as pd

dataTrain = pd.read_csv('datasettrain.csv', header=None, delimiter='
')
dataTest = pd.read_csv('datasettest.csv', header=None, delimiter=' ')

yTrain = dataTrain.iloc[:, 36]
```

```
XTrain = dataTrain.iloc[:, 0:36]
yTest = dataTest.iloc[:, 36]
XTest = dataTest.iloc[:, 0:36]

XTrain.insert(0, -1, 1)
XTest.insert(0, -1, 1)

yTrain = yTrain.values
XTrain = XTrain.values
yTest = yTest.values
XTest = XTest.values

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def cost_function(theta, x, y, lambda_reg):
    m = x.shape[0]
    J = (-np.log(sigmoid(x.dot(theta)).T).dot(y) - np.log(1 -
sigmoid(x.dot(theta)).T).dot(1 - y))/m +
lambda_reg*np.sum(np.square(theta))/(2*m)
    return J

def gradient_function(theta, x, y, lambda_reg):
    m = x.shape[0]
    h = sigmoid(x.dot(theta)).reshape(-1, 1)
    y = y.reshape(m, 1)
    gradient = np.zeros((theta.shape[0], 1))
    gradient = x.T.dot(h - y)/m
    theta = theta.reshape((theta.shape[0], 1))
    gradient[1:] = gradient[1:] + (lambda_reg/m)*theta[1:]
    return gradient

def oneVsAll(X, y, num_labels, num_features, lambda_reg):
    thetas_set = np.zeros((num_labels, num_features+1))
    for i in range(0, num_labels):
        theta = np.zeros((num_features+1, 1))
        example = (y == i + 1) * 1
        result = opt.fmin_tnc(func=cost_function , x0=theta ,
fprime=gradient_function , args=(X, example, lambda_reg), disp=5)
        theta_opt = result[0]
        thetas_set[i,:] = theta_opt
    return thetas_set

def accuracy(X, y, thetas):
    activations = X.dot(thetas.T)
    y = y.reshape(len(y), 1)
    predictions = np.zeros(len(activations))
    predictions = predictions.reshape(X.shape[0], 1)
```

```
    for i in range(len(activations)):
        idx = np.argmax(activations[i])
        predictions[i] = idx + 1

    number = np.sum(predictions == y)
    accuracy = (float(number)/X.shape[0])*100
    return accuracy

accuracyTrainValues = []
accuracyTestValues = []
lambdas = np.arange(0.0, 1.6, 0.5)
for lambda_reg in lambdas:
    thetas = oneVsAll(XTrain, yTrain, 7, 36, lambda_reg)
    accuracyTrain = accuracy(XTrain, yTrain, thetas)
    accuracyTest = accuracy(XTest, yTest, thetas)
    accuracyTrainValues.append(accuracyTrain)
    accuracyTestValues.append(accuracyTest)
    print("Accuracy training (lambda = " + str(lambda_reg) + ") = " +
str(accuracyTrain) + "%")
    print("Accuracy test (lambda = " + str(lambda_reg) + ") = " +
str(accuracyTest) + "%")

plt.plot(lambdas, accuracyTrainValues, color='red', label='Train')
plt.plot(lambdas, accuracyTestValues, color='blue', label='Test')
plt.xlabel("Lambda")
plt.ylabel("Accuracy (%)")
plt.legend()
plt.show()
```

Neural Network

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
import pandas as pd

dataTrain = pd.read_csv('datasettrain.csv', header=None, delimiter='
')
dataTest = pd.read_csv('datasettest.csv', header=None, delimiter=' ')

yTrain = dataTrain.iloc[:, 36]
XTrain = dataTrain.iloc[:, 0:36]
yTest = dataTest.iloc[:, 36]
XTest = dataTest.iloc[:, 0:36]

XTrain.insert(0, -1, 1)
XTest.insert(0, -1, 1)

yTrain = yTrain.values
```

```
XTrain = XTrain.values
yTest = yTest.values
XTest = XTest.values

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

def initializeWeights(LIn, LOut):
    weights = np.random.uniform(low=-0.12, high=0.12, size=(LOut, 1 +
LIn))
    return weights

def encodeLabels(num_labels, labels):
    labels = np.array(labels)
    oneHot = np.zeros((labels.shape[0], num_labels))
    for i in range(labels.shape[0]):
        oneHot[i][labels[i]-1] = 1
    return oneHot

def costFunction(X, y, theta1, theta2, lambda_reg):
    m = X.shape[0]
    J = (1.0/m) * np.sum(np.sum((-y * np.log(X)) - ((1 - y) *
np.log(1 - X))))
    regularization = (np.sum(np.sum(np.square(theta1[:,1:]))) +
np.sum(np.sum(np.square(theta2[:,1:])))) * (float(lambda_reg)/(2*m))
    J = J + regularization
    return J

def backprop(params_nn, num_in, num_hid, num_labels, X, y,
lambda_reg):
    theta1 = np.reshape(params_nn[:num_hid*(num_in + 1)], (num_hid,
(num_in + 1)))
    theta2 = np.reshape(params_nn[num_hid*(num_in + 1):],
(num_labels, (num_hid + 1)))
    m = X.shape[0]
    y = encodeLabels(num_labels, y)
    z2 = X.dot(theta1.T)
    a2 = np.hstack((np.ones((z2.shape[0], 1)), sigmoid(z2)))
    z3 = a2.dot(theta2.T)
    a3 = sigmoid(z3)
    J = costFunction(a3, y, theta1, theta2, lambda_reg)

    theta1Gradient = np.zeros(theta1.shape)
    theta2Gradient = np.zeros(theta2.shape)

    delta3 = a3 - y
```

```
    delta2 = (theta2.T.dot(delta3.T)).T *
np.hstack((np.ones((z2.shape[0], 1)), derivative(z2)))
    delta2 = delta2[:, 1:]

    theta1Gradient = theta1Gradient + delta2.T.dot(X)
    theta2Gradient = theta2Gradient + delta3.T.dot(a2)

    theta1Gradient = (1/float(m)) * theta1Gradient
    theta2Gradient = (1/float(m)) * theta2Gradient

    theta1Gradient[:, 1:] = theta1Gradient[:, 1:] +
(float(lambda_reg)/m)*theta1[:, 1:]
    theta2Gradient[:, 1:] = theta2Gradient[:, 1:] +
(float(lambda_reg)/m)*theta2[:, 1:]

    gradients = np.concatenate((theta1Gradient, theta2Gradient),
axis=None)

    return J, gradients

def neuralNetwork(X, theta1, theta2):
    a2 = sigmoid(X.dot(theta1.T))
    a2 = np.hstack((np.ones((a2.shape[0], 1)), a2))
    a3 = sigmoid(a2.dot(theta2.T))
    predictions = np.zeros((X.shape[0], 1))
    for i in range(len(a3)):
        idx = np.argmax(a3[i])
        idx = idx + 1
        predictions[i] = idx
    return predictions

def accuracy(X, y, theta1, theta2):
    y = y.reshape(len(y), 1)
    predictions = neuralNetwork(X, theta1, theta2)
    number = np.sum(predictions == y)
    accuracy = (float(number)/X.shape[0])*100
    return accuracy

def NNTrainAndTest(XTrain, yTrain, XTest, yTest, numIn, numHid,
numOut, lambda_reg):
    thetaTrain1 = initializeWeights(numIn, numHid)
    thetaTrain2 = initializeWeights(numHid, numOut)
    thetasTrain = np.concatenate((thetaTrain1, thetaTrain2),
axis=None)
    resultTheta = opt.fmin_tnc(func=backprop, x0=thetasTrain,
fprime=None, args=(numIn, numHid, numOut, XTrain, yTrain,
lambda_reg), disp=5)
    resultTheta = resultTheta[0]
```

```
    theta1 = np.reshape(resultTheta[:numHid*(numIn+1)], ((numHid,
numIn+1)))
    theta2 = np.reshape(resultTheta[numHid*(numIn+1):], ((numOut,
numHid+1)))

    accuracyTrain = accuracy(XTrain, yTrain, theta1, theta2)
    accuracyTest = accuracy(XTest, yTest, theta1, theta2)
    print("Accuracy train for lambda = " + str(lambda_reg) + " = " +
str(accuracyTrain) + "%")
    print("Accuracy test for lambda = " + str(lambda_reg) + " = " +
str(accuracyTest) + "%")
    return accuracyTrain, accuracyTest

accuracyTrainValues = []
accuracyTestValues = []
lambdas = np.arange(0.0, 1.6, 0.5)
for lambda_reg in lambdas:
    lambda_reg = round(lambda_reg, 1)
    accuracyTrain, accuracyTest = NNTrainAndTest(XTrain, yTrain,
XTest, yTest, 36, 25, 7, lambda_reg)
    accuracyTrainValues.append(accuracyTrain)
    accuracyTestValues.append(accuracyTest)

plt.plot(lambdas, accuracyTrainValues, color='red', label='Train')
plt.plot(lambdas, accuracyTestValues, color='blue', label='Test')
plt.xlabel("Lambda")
plt.ylabel("Accuracy (%)")
plt.legend()
plt.show()
```

Support Vector Machines

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
from sklearn.svm import SVC
import pandas as pd

dataTrain = pd.read_csv('datasettrain.csv', header=None, delimiter='
')
dataTest = pd.read_csv('datasettest.csv', header=None, delimiter=' ')

yTrain = dataTrain.iloc[:, 36]
XTrain = dataTrain.iloc[:, 0:36]
yTest = dataTest.iloc[:, 36]
XTest = dataTest.iloc[:, 0:36]

yTrain = yTrain.values
XTrain = XTrain.values
```

```
yTest = yTest.values
XTest = XTest.values

def accuracy(svm, X, y):
    predictions = svm.predict(X)
    number = 0
    for i in range(len(y)):
        if predictions[i] == y[i]:
            number = number + 1
    accuracy = (float(number)/len(y))*100
    return accuracy

def SVMTrainAndTest(svm, XTrain, yTrain, XTest, yTest):
    svm.fit(XTrain, yTrain)
    accuracyTrain = accuracy(svm, XTrain, yTrain)
    accuracyTest = accuracy(svm, XTest, yTest)
    return accuracyTrain, accuracyTest

accuracyTrainLinear = []
accuracyTestLinear = []

CValues = [0.01, 0.1, 1]
SigmaValues = [0.01, 0.1, 1, 10, 100]

def SVMParameterSearch(XTrain, yTrain, XTest, yTest, CValues,
SigmaValues):
    for C in CValues:
        svmLin = SVC(kernel='linear', C=C)
        accuracyTrain, accuracyTest = SVMTrainAndTest(svmLin, XTrain,
yTrain, XTest, yTest)
        accuracyTrainLinear.append(accuracyTrain)
        accuracyTestLinear.append(accuracyTest)
        print("Train accuracy for SVM with linear kernel for C = " +
str(C) + " = " + str(accuracyTrain) + "%")
        print("Test accuracy for SVM with linear kernel for C = " +
str(C) + " = " + str(accuracyTest) + "%")
        for sigma in SigmaValues:
            svmRBF = SVC(kernel='rbf', C=C, gamma=1.0/(2 * sigma **
2))
            accuracyTrain, accuracyTest = SVMTrainAndTest(svmRBF,
XTrain, yTrain, XTest, yTest)
            print("Train accuracy for SVM with RBF kernel for C = " +
str(C) + " and sigma = " + str(sigma) + " = " + str(accuracyTrain) +
"%")
            print("Test accuracy for SVM with RBF kernel for C = " +
str(C) + " and sigma = " + str(sigma) + " = " + str(accuracyTest) +
"%")
```

```
SVMParameterSearch(XTrain, yTrain, XTest, yTest, CValues,
SigmaValues)

plt.plot(CValues, accuracyTrainLinear, color='red', label='Train')
plt.plot(CValues, accuracyTestLinear, color='blue', label='Test')
plt.xlabel("C")
plt.ylabel("Accuracy for linear kernel (%)")
plt.legend()
plt.show()
```