

# CSE 474-INTRO TO MACHINE LEARNING

*Linear Models for Supervised Learning*

**Project Assignment 1**

**Christian Caballero**

**Clayton Dombrowsky**

**Ankit Sigroha**

**Group 34**

**03.06.2019**

## Report 1:

Calculate and report the RMSE for training and test data for two cases: first, without using an intercept (or bias) term, and second with using an intercept. Which one is better?

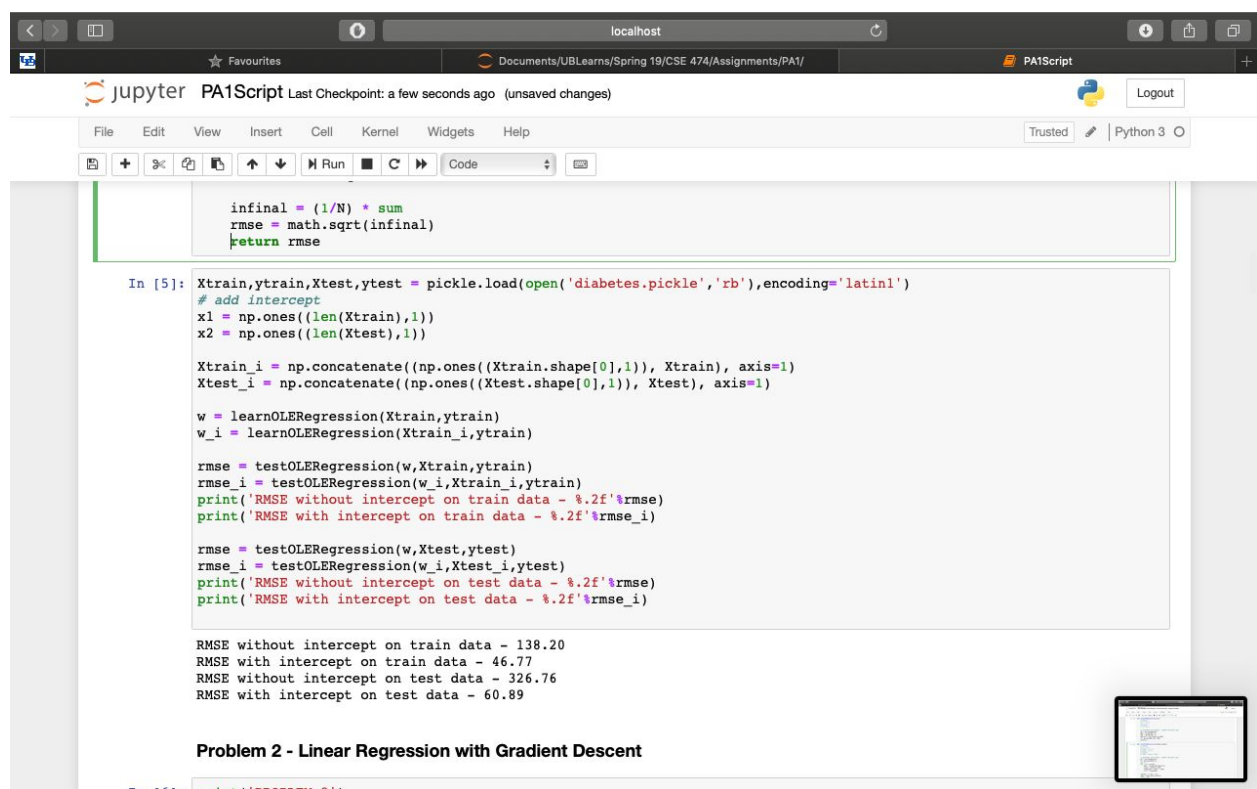
Ans: The RMSE without intercept on train data is: 138.20

The RMSE with intercept on train data is: 46.77

The RMSE without intercept on test data is: 326.76

The RMSE with intercept on test data is: 60.89

The RMSE with intercept is better since it is less and we eventually want to minimize the error in the learning on the real data or the test data which is low in this case. Adding an intercept lowers the RMSE error.



```
def calculate_rmse(Xtrain, ytrain, Xtest, ytest):
    infinal = (1/N) * sum
    rmse = math.sqrt(infinal)
    return rmse

In [5]: Xtrain, ytrain, Xtest, ytest = pickle.load(open('diabetes.pickle', 'rb'), encoding='latin1')
# add intercept
x1 = np.ones((len(Xtrain), 1))
x2 = np.ones((len(Xtest), 1))

Xtrain_i = np.concatenate((np.ones((Xtrain.shape[0], 1)), Xtrain), axis=1)
Xtest_i = np.concatenate((np.ones((Xtest.shape[0], 1)), Xtest), axis=1)

w = learnOLERegression(Xtrain, ytrain)
w_i = learnOLERegression(Xtrain_i, ytrain)

rmse = testOLERegression(w, Xtrain, ytrain)
rmse_i = testOLERegression(w_i, Xtrain_i, ytrain)
print('RMSE without intercept on train data - %.2f'%rmse)
print('RMSE with intercept on train data - %.2f'%rmse_i)

rmse = testOLERegression(w, Xtest, ytest)
rmse_i = testOLERegression(w_i, Xtest_i, ytest)
print('RMSE without intercept on test data - %.2f'%rmse)
print('RMSE with intercept on test data - %.2f'%rmse_i)

RMSE without intercept on train data - 138.20
RMSE with intercept on train data - 46.77
RMSE without intercept on test data - 326.76
RMSE with intercept on test data - 60.89
```

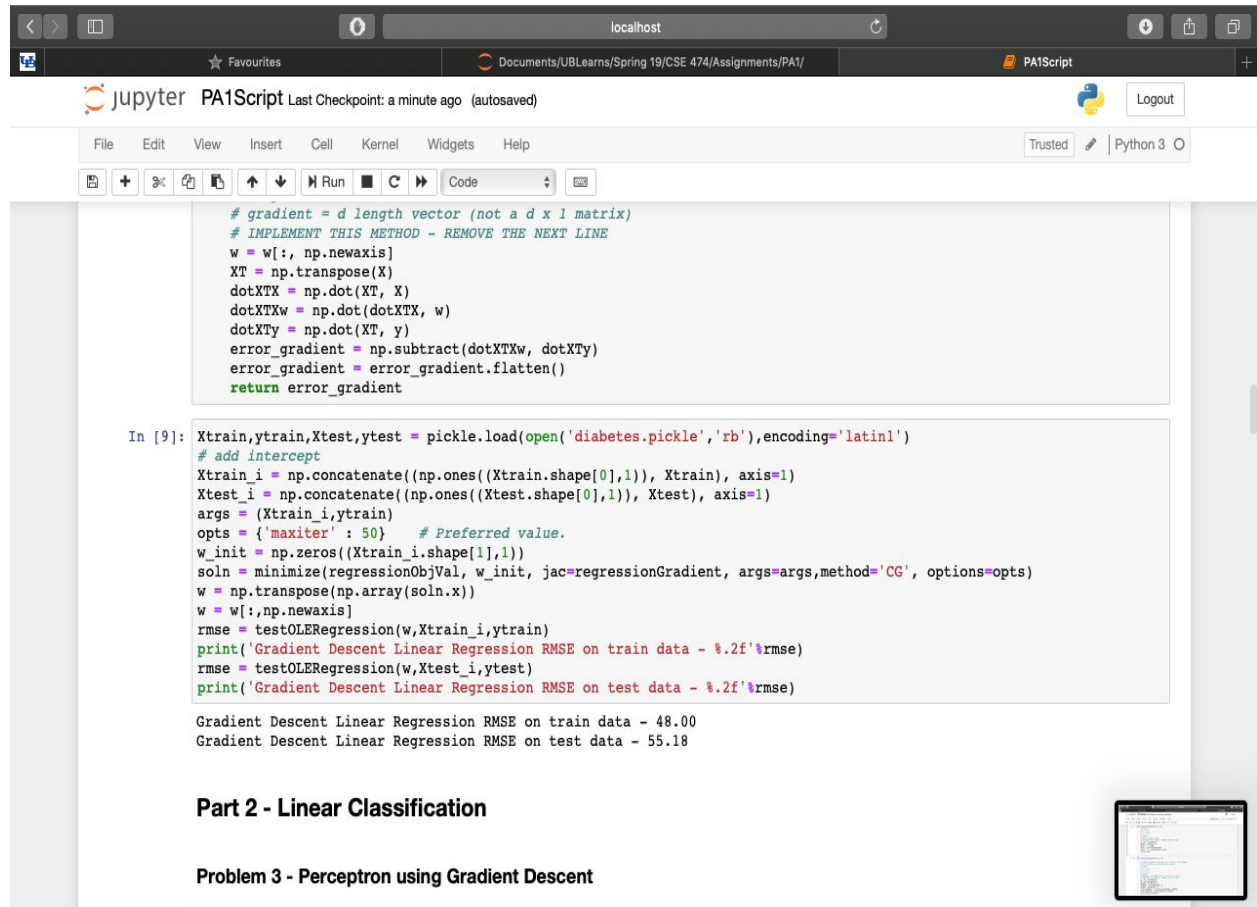
**Problem 2 - Linear Regression with Gradient Descent**

## Report 2:

Using testOLERegression, calculate and report the RMSE for training and test data after gradient descent based learning. Compare with the RMSE after direct minimization. Which one is better?

Ans: Gradient Descent Linear Regression RMSE on train data is: 48.00  
Gradient Descent Linear Regression RMSE on test data is: 55.18

The RMSE after the minimization using gradient descent is better as it lowers the error than RMSE with an intercept.



```
# gradient = d length vector (not a d x 1 matrix)
# IMPLEMENT THIS METHOD - REMOVE THE NEXT LINE
w = w[:, np.newaxis]
XT = np.transpose(X)
dotXTX = np.dot(XT, X)
dotXTXw = np.dot(dotXTX, w)
dotXTy = np.dot(XT, y)
error_gradient = np.subtract(dotXTXw, dotXTy)
error_gradient = error_gradient.flatten()
return error_gradient

In [9]: Xtrain,ytrain,Xtest,ytest = pickle.load(open('diabetes.pickle','rb'),encoding='latin1')
# add intercept
Xtrain_i = np.concatenate((np.ones((Xtrain.shape[0],1)), Xtrain), axis=1)
Xtest_i = np.concatenate((np.ones((Xtest.shape[0],1)), Xtest), axis=1)
args = (Xtrain_i,ytrain)
opts = {'maxiter' : 50} # Preferred value.
w_init = np.zeros((Xtrain_i.shape[1],1))
soln = minimize(regressionObjVal, w_init, jac=regressionGradient, args=args,method='CG', options=opts)
w = np.transpose(np.array(soln.x))
w = w[:,np.newaxis]
rmse = testOLERegression(w,Xtrain_i,ytrain)
print('Gradient Descent Linear Regression RMSE on train data - %.2f'%rmse)
rmse = testOLERegression(w,Xtest_i,ytest)
print('Gradient Descent Linear Regression RMSE on test data - %.2f'%rmse)

Gradient Descent Linear Regression RMSE on train data - 48.00
Gradient Descent Linear Regression RMSE on test data - 55.18

Part 2 - Linear Classification

Problem 3 - Perceptron using Gradient Descent
```

### **Report 3:**

Train the perceptron model by calling the `scipy.optimize.minimize` method and use the `evaluateLinearModel` to calculate and report the accuracy for the training and test data.

Ans: Perceptron accuracy on train data is: 0.84

Perceptron accuracy on test data is: 0.84

The screenshot shows a Jupyter Notebook interface with a dark theme. The browser address bar shows 'localhost'. The Jupyter interface includes a top bar with 'Favourites', 'Documents/UBLeads/Spring 19/CSE 474/Assignments/PA1/', and 'PA1Script'. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. A toolbar contains icons for file operations and a 'Run' button. The main area displays two code cells. The first cell contains a function to calculate accuracy. The second cell, labeled 'In [13]:', contains code to load data, concatenate features, train a Perceptron model using 'regressionObjVal' and 'regressionGradient' methods, and print the accuracy on both training and test data. The output shows 0.84 for both. Below this is a section titled 'Problem 4 - Logistic Regression Using Newton's Method'. The third cell, labeled 'In [14]:', contains code to print 'PROBLEM 4' and a separator line. The output shows 'PROBLEM 4' followed by a dashed line. A small thumbnail of the notebook is visible in the bottom right corner.

```
def evaluateLinearModel(w, Xtrain, ytrain, Xtest, ytest):
    acc_counter = 0
    for i in range(Xtest.shape[0]):
        if ytest[i] == ypred[i]:
            acc_counter += 1
    acc = (acc_counter / len(Xtest))
    return acc

In [13]: Xtrain, ytrain, Xtest, ytest = np.load(open('sample.pickle', 'rb'))
# add intercept
Xtrain_i = np.concatenate((np.ones((Xtrain.shape[0], 1)), Xtrain), axis=1)
Xtest_i = np.concatenate((np.ones((Xtest.shape[0], 1)), Xtest), axis=1)

args = (Xtrain_i, ytrain)
opts = {'maxiter': 50} # Preferred value.
w_init = np.zeros((Xtrain_i.shape[1], 1))
soln = minimize(regressionObjVal, w_init, jac=regressionGradient, args=args, method='CG', options=opts)
w = np.transpose(np.array(soln.x))
w = w[:, np.newaxis]
acc = evaluateLinearModel(w, Xtrain_i, ytrain)
print('Perceptron Accuracy on train data - %.2f' % acc)
acc = evaluateLinearModel(w, Xtest_i, ytest)
print('Perceptron Accuracy on test data - %.2f' % acc)

Perceptron Accuracy on train data - 0.84
Perceptron Accuracy on test data - 0.84

Problem 4 - Logistic Regression Using Newton's Method

In [14]: print('PROBLEM 4')
print('-----')

PROBLEM 4
-----
```

## **Report 4:**

Train the logistic regression model by calling the `scipy.optimize.minimize` method, and use the `evaluateLinearModel` to calculate and report the accuracy for the training and test data.

Ans: Logistic Regression accuracy on train data is: 0.84

Logistic Regression accuracy on test data is: 0.86

```

hessian = 0
for i in range(N):
    wTx = np.dot(wT, X[i])
    ywTx = np.dot(y[i], wTx)
    top = np.exp(ywTx)
    bot = (1 + top) * (1 + top)
    xxT = np.prod(X[i])
    quotient = np.divide(top, bot)
    hessian += np.dot(quotient, xxT)

hessian = (1/N) * hessian
return hessian

In [18]: rain,ytrain, Xtest, ytest = np.load(open('sample.pickle','rb'))
add intercept
rain_i = np.concatenate((np.ones((Xtrain.shape[0],1)), Xtrain), axis=1)
est_i = np.concatenate((np.ones((Xtest.shape[0],1)), Xtest), axis=1)

gs = (Xtrain_i,ytrain)
ts = {'maxiter': 50} # Preferred value.
init = np.zeros((Xtrain_i.shape[1],1))
ln = minimize(logisticObjVal, w_init, jac=logisticGradient, hess=logisticHessian, args=args,method='Newton-CG', options
= np.transpose(np.array(soln.x))
= np.reshape(w,[len(w),1])
c = evaluateLinearModel(w,Xtrain_i,ytrain)
int('Logistic Regression Accuracy on train data - %.2f'%acc)
c = evaluateLinearModel(w,Xtest_i,ytest)
int('Logistic Regression Accuracy on test data - %.2f'%acc)

Logistic Regression Accuracy on train data - 0.84
Logistic Regression Accuracy on test data - 0.86

Problem 5 - Support Vector Machines Using Gradient Descent

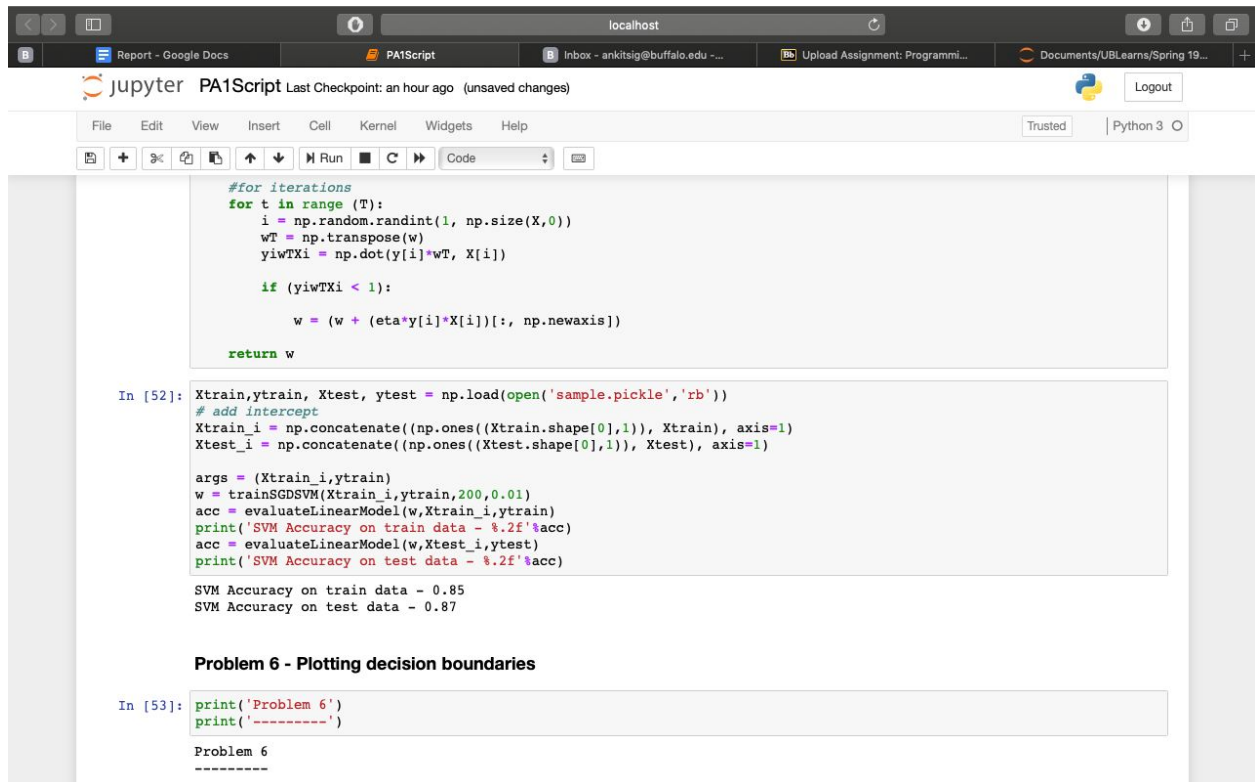
In [19]: print('PROBLEM 5')
```

## Report 5:

Train the SVM model by calling the trainSGDSVM method for 200 iterations (set learning rate parameter  $\eta$  to 0.01). Use the evaluateLinearModel to calculate and report the accuracy for the training and test data.

Ans: The Accuracy for  $T = 200$  and  $\eta = 0.01$  is computed below:-

Ans: SVM accuracy on train data is: 0.85



The image shows a Jupyter Notebook interface with a browser window at the top. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The current cell is a code cell containing the following Python code:

```
#for iterations
for t in range(T):
    i = np.random.randint(1, np.size(X,0))
    wT = np.transpose(w)
    yiwTXi = np.dot(y[i]*wT, X[i])

    if (yiwTXi < 1):
        w = (w + (eta*y[i]*X[i]))[:, np.newaxis]

return w
```

The output of the code cell is displayed below it:

```
In [52]: Xtrain,ytrain, Xtest, ytest = np.load(open('sample.pickle','rb'))
# add intercept
Xtrain_i = np.concatenate((np.ones((Xtrain.shape[0],1)), Xtrain), axis=1)
Xtest_i = np.concatenate((np.ones((Xtest.shape[0],1)), Xtest), axis=1)

args = (Xtrain_i,ytrain)
w = trainSGDSVM(Xtrain_i,ytrain,200,0.01)
acc = evaluateLinearModel(w,Xtrain_i,ytrain)
print('SVM Accuracy on train data - %.2f'%acc)
acc = evaluateLinearModel(w,Xtest_i,ytest)
print('SVM Accuracy on test data - %.2f'%acc)

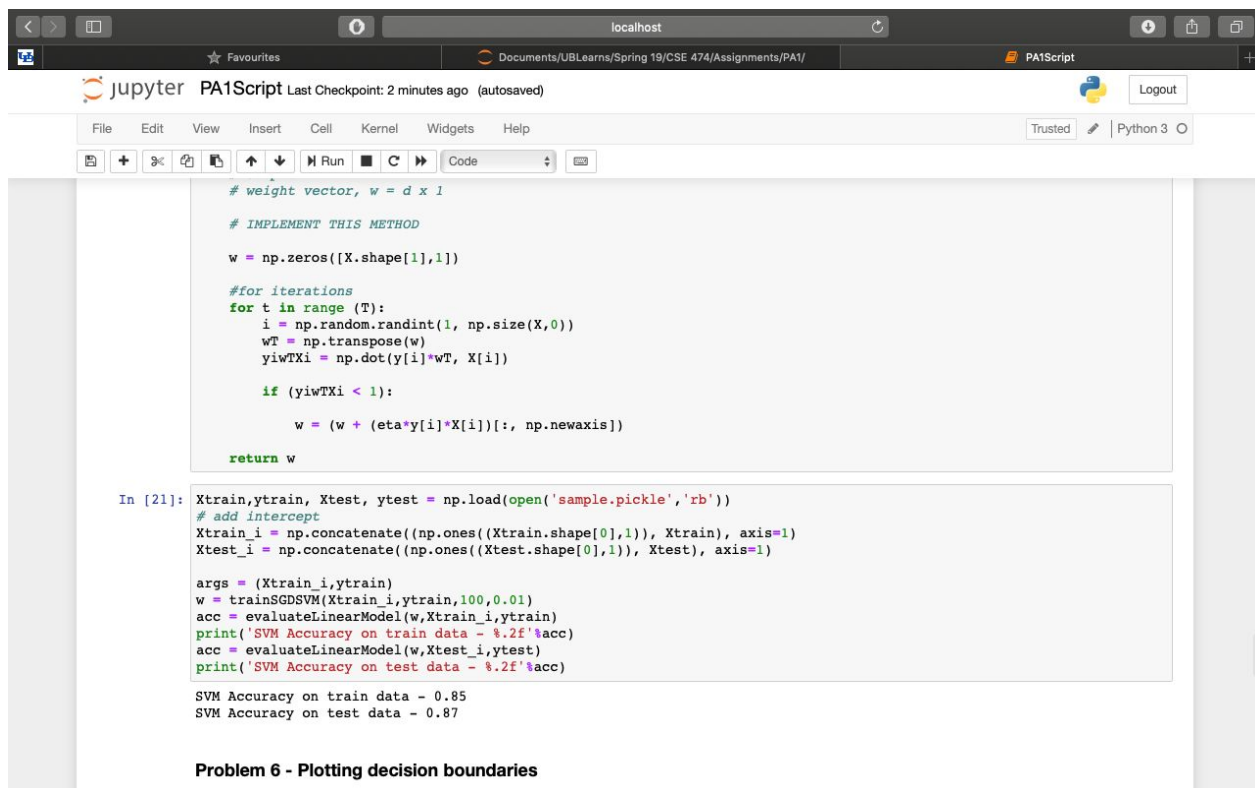
SVM Accuracy on train data - 0.85
SVM Accuracy on test data - 0.87
```

Below the output, there is a section titled "Problem 6 - Plotting decision boundaries". The code cell for this problem is:

```
In [53]: print('Problem 6')
print('-----')

Problem 6
-----
```

SVM accuracy on test data is: 0.87



The image shows a Jupyter Notebook interface with a browser window at the top. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The current cell is a code cell containing the following Python code:

```
# weight vector, w = d x 1
# IMPLEMENT THIS METHOD

w = np.zeros([X.shape[1],1])

#for iterations
for t in range(T):
    i = np.random.randint(1, np.size(X,0))
    wT = np.transpose(w)
    yiwTXi = np.dot(y[i]*wT, X[i])

    if (yiwTXi < 1):
        w = (w + (eta*y[i]*X[i]))[:, np.newaxis]

return w
```

The output of the code cell is displayed below it:

```
In [21]: Xtrain,ytrain, Xtest, ytest = np.load(open('sample.pickle','rb'))
# add intercept
Xtrain_i = np.concatenate((np.ones((Xtrain.shape[0],1)), Xtrain), axis=1)
Xtest_i = np.concatenate((np.ones((Xtest.shape[0],1)), Xtest), axis=1)

args = (Xtrain_i,ytrain)
w = trainSGDSVM(Xtrain_i,ytrain,100,0.01)
acc = evaluateLinearModel(w,Xtrain_i,ytrain)
print('SVM Accuracy on train data - %.2f'%acc)
acc = evaluateLinearModel(w,Xtest_i,ytest)
print('SVM Accuracy on test data - %.2f'%acc)

SVM Accuracy on train data - 0.85
SVM Accuracy on test data - 0.87
```

Below the output, there is a section titled "Problem 6 - Plotting decision boundaries".

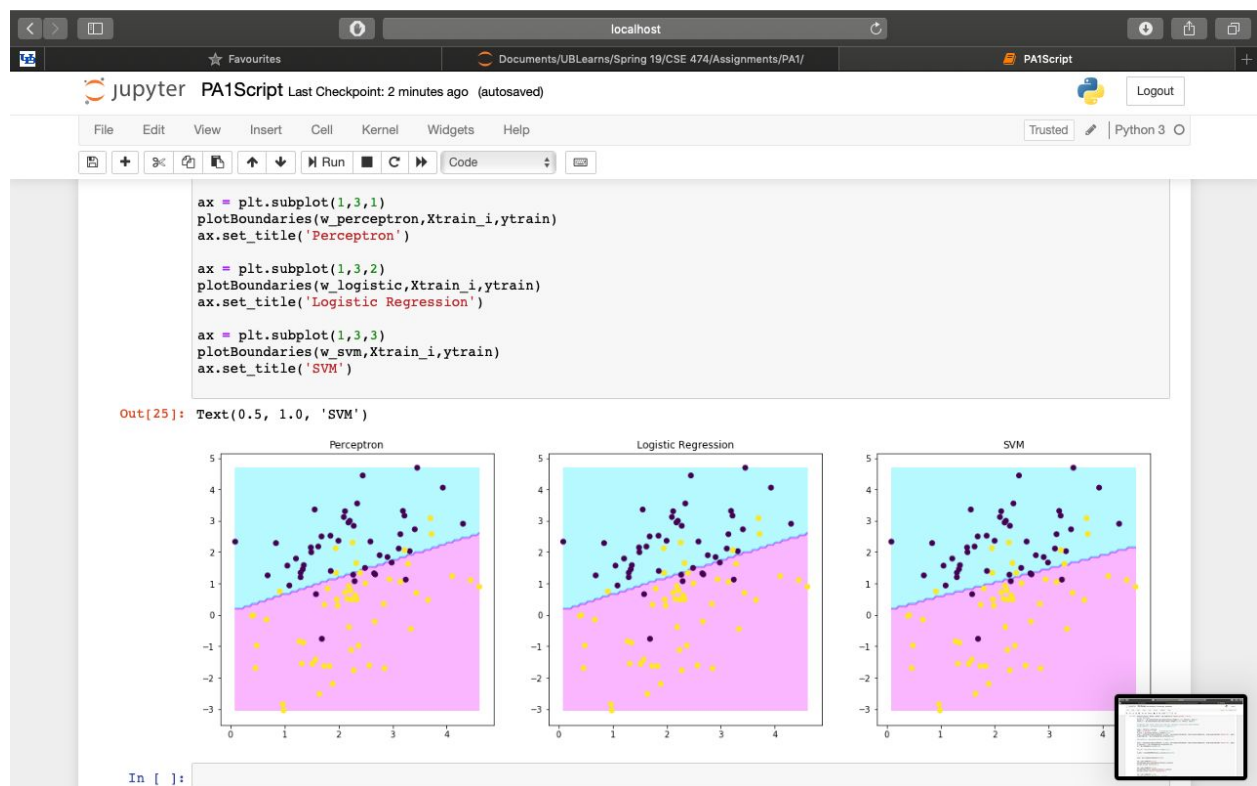
## Report 6:

1. Use the results for test data to determine which classifier is the most accurate?

Ans: Out of the three classifiers the SVM is most accurate of all. The Linear regression is second most accurate followed by the perceptron.

2. Plot the decision boundaries learnt by each classifier using the provided plotDecisionBoundary function which takes the learnt weight vector,  $w$  as one of the parameters. Study the three boundaries and provide your insights.

Ans: The three decision boundaries are:



Although the accuracy differs by a little but the SVM decision boundary is efficient as it provides a less slack for the points which are quite not separable. The homogeneous classification with the less risk of error makes it most accurate whereas the Perceptron and Logistic Regression classification seems to differ by a little and the decision boundary is flexible to make some errors or to take risk on classifying the data.