CS/ECE/ME532 Activity 19

Estimated Time: 15 mins for P1, 20 mins for P2, 20 mins for P3

1. You have two feature vectors $\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\boldsymbol{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and corresponding labels $d_1 = -1, d_2 = 1$. The linear classifier is $\text{sign}\{\boldsymbol{x}_i^T \boldsymbol{w}\}$ where $\boldsymbol{w} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$.

a) Find the squared error loss for this classifier.

b) Find the hinge loss for this classifier.

2. You have four data points $x_1 = 2, x_2 = 1.5, x_3 = 1/2, x_4 = -1/2$ and corresponding labels $y_1 = 1, y_2 = 1, y_3 = -1, y_4 = -1$.

a) Find a maximum margin linear classifier for this data. *Hint:* Graph the data.

b) Use squared-error loss to train the classifier (with the help of Python). Does this classifier make any errors?

c) Find a classifier with zero hinge loss. *Hint:* Use what you've learned about hinge loss, not computation. Does this classifier make any errors?

d) Now suppose $x_4 = -5$. Use squared-error loss to find the classifier (with the help of Python). Does this classifier make any errors?

e) Can you still find a classifier with zero hinge loss when $x_4 = -5$? Does it make any errors?

3. Previously, we examined the performance of classifiers trained using the squared error loss function (i.e, trained using least squares). This problem uses an *off-the-shelf* Linear Support Vector Machine to train a binary linear classifier.

The data set is divided into training and test data sets. In order to represent a decision boundary that may not pass through the origin, we can consider the feature vector $\boldsymbol{x}^T = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix}$.

a) *Classifier using off the shelf SVM.* Code is provided to train a classifier using an off the shelf SVM with hinge loss. Run the code to find the linear classifier weights. Next, uses the weights to predict the class of the test data. How many classification errors occur?

b) Comment out the code that trains the classifier using the linear SVM, and uncomment the code that train the classifier using least squares (i.e, $\boldsymbol{w}_{opt} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$). How many errors occur on the test set?

c) Training a classifier using the *squared error as a loss function* can fail when correctly labeled data points lie far from the decision boundary. Linear SVMs trained with hinge loss are not susceptible to the same problem. A new dataset consisting of the first dataset, plus 1000 (correctly labeled) datapoints at $x_1 = 0, x_2 = 10$ is created. What happens to the decision boundary when these new data points are included in training the linear SVM?

d) How does this compare with the error rate of the linear classifier trained with the new data points? Why is the such a difference in performance?

DEVIN BRESSER

1.) $\underline{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (-1) $\underline{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ (1)

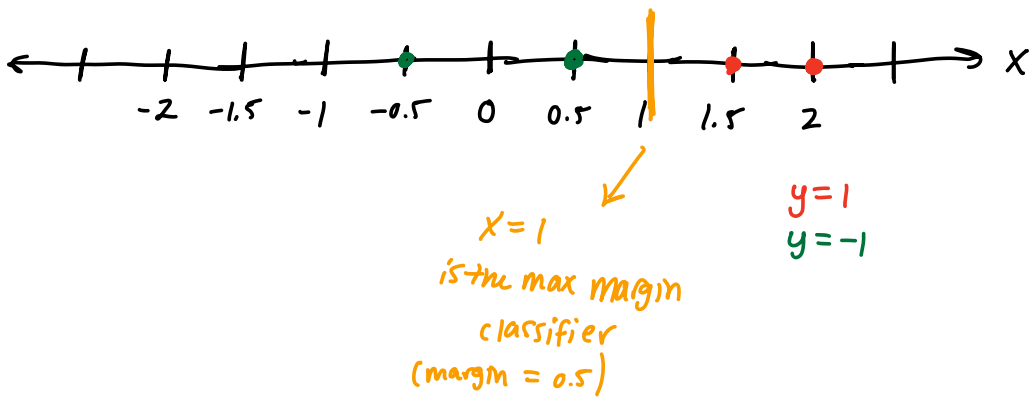
a.) $\sum_i (\underline{x}_i^T \underline{w} - d)^2 = \underbrace{\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0.5 \end{bmatrix} + 1 \right)^2}_{-0.5} + \underbrace{\left(\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0.5 \end{bmatrix} - 1 \right)^2}_{1.5}$

$= 0.25 + 0.25 = 0.5$

b.) $\sum_i \left(1 - d_i \underline{x}_i^T \underline{w}\right)_+ = \left(1 - \underbrace{(-1 \cdot -0.5)}_{0.5}\right)_+ + \left(1 - \underbrace{(1 \cdot 1.5)}_{1.5}\right)_+$

$= (0.5)_+ + (-1.5)_+ = 0.5$ $\quad \max(0, 1 - d_i \underline{x}_i^T \underline{w})$

2.)



2a.)

```
In [1]: import numpy as np

In [25]: # Problem 2 - Devin Bresser

         X = np.array([[2,1],[1.5,1],[0.5,1],[-0.5,1]])
         y = np.array([[1],[1],[-1],[-1]])

In [29]: w_min = np.linalg.inv((X.T @ X)) @ X.T @ y
         np.sign(X @ w_min)

Out[29]: array([[ 1.],
                [ 1.],
                [-1.],
                [-1.]])

In [30]: # Problem 2a comment: no errors
```

2c.)

$$\begin{bmatrix} 2 \\ 1.5 \\ 0.5 \\ -0.5 \end{bmatrix} \begin{bmatrix} w_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

$$(1 - 2w_1)_+ = 0$$
$$(1 - 1.5w_1)_+ = 0$$
$$(1 + 0.5w_1)_+ = 0$$
$$(1 - 0.5w_1)_+ = 0$$

There is clearly no 1d classifier that satisfies these constraints due to the 3rd term

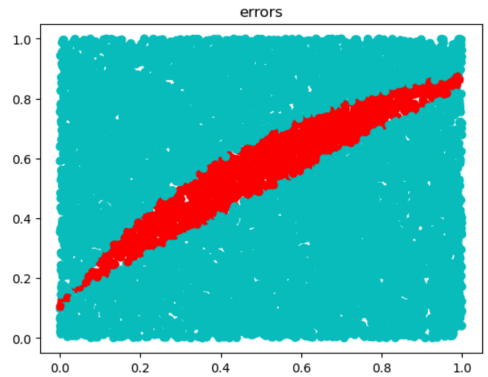
2d.)

```
In [1]: import numpy as np

In [31]: # Problem 2 - Devin Bresser

         X = np.array([[2,1],[1.5,1],[0.5,1],[-5,1]])
         y = np.array([[1],[1],[-1],[-1]])

In [32]: w_min = np.linalg.inv((X.T @ X)) @ X.T @ y
         np.sign(X @ w_min)

Out[32]: array([[ 1.],
                [ 1.],
                [ 1.],
                [-1.]])

In [30]: # Problem 2d comment: Now point 3 is misclassified

In [ ]:
```

2e.) The 3rd term still has the opposite sign in the $(\cdot)_+$ expression so nothing has changed.

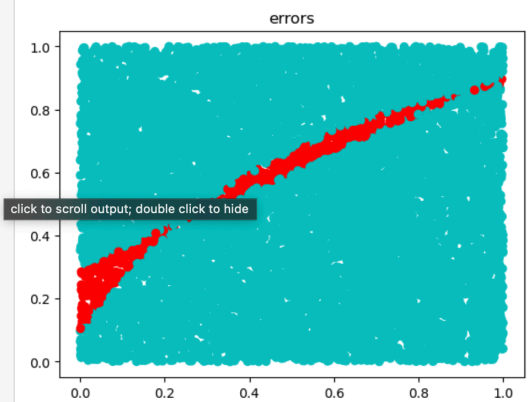
3a.)

```
rrors: '+ str(sum(error_vec)))
```




```
Errors: 1213
```

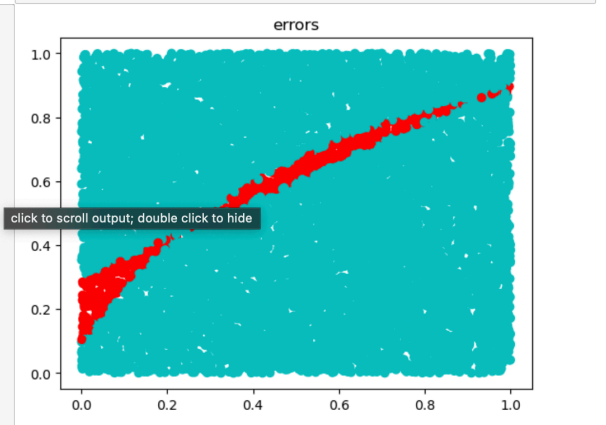
3b.)




```
Errors: 495
```

3c.)

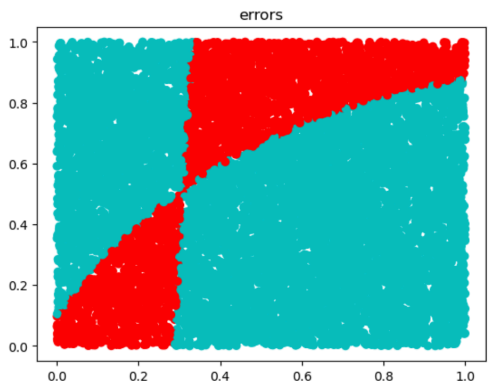
Nothing changes b/c hinge loss is not susceptible to correctly labeled outliers




```
Errors: 495
```

3d.) Sq. error binary classification is influenced by "easy to classify" points (large $|\underline{x}_i^T \underline{w}|$, but correctly classified)

In [11]:
```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_outlier, y
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i i
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec)))
```



Errors: 2668

In [ ]:

So the performance decreases.