Problem 1

The answers to the stated questions:

1. **Q:** 1.a) Run the function several times - what do you notice for the Perceptron as compared to NCC or LDA? In one sentence, explain the behaviour of the perceptron.

   A: The Perceptron has a threshold so it cannot perfectly separate the data. Also the accuracy of the Perceptron varies a lot with the data set. In general, the best performance is achieved by the **LDA**, the second by the **PER** and the third **NCC**. But the results varies.

2. **Q:** 1.b) Have a look in the code how the toy data is generated - is LDA optimal for this type of data?

   A: The data is generated by two multivariate_normal distributions with the same covariance matrix. The LDA should by optimal for such a task.

3. **Q:** 1.c) How would you have to change the data generation such that NCC and LDA yield the same result?

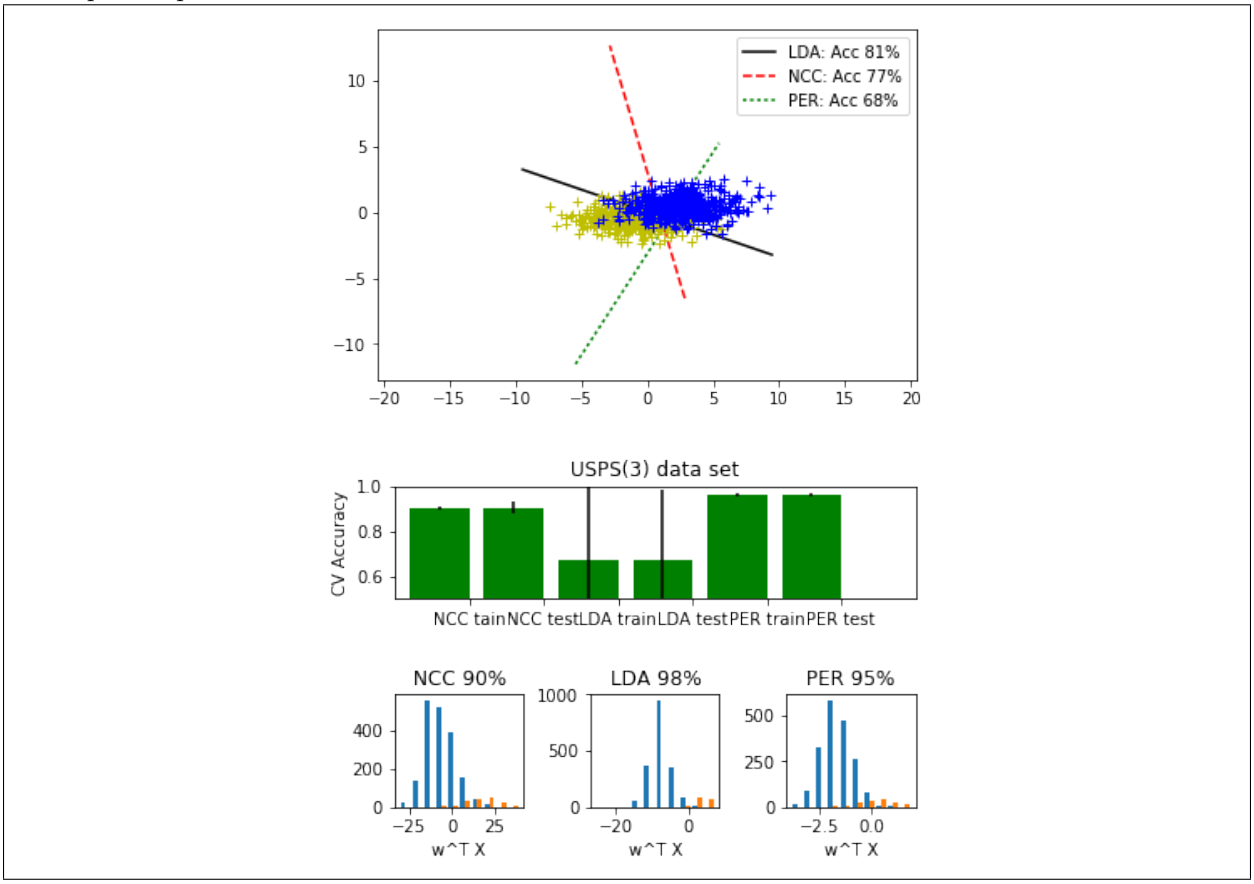
   A: You would have to send the means of the generated data more far away.

4. **Q:** 3. Examine the function **crossvalidate**. Briefly explain in your own words what this function does. When we want to compare the performance of different classifiers, which values should we look at - the train or the test accuracies?

   A: The function divides the data into chunks (the number of them is given as an argument. by default in 5) and then applies the Cross Validation algorithm for each chunk as the test data and the others as training data. When we compare we should look at the test accuracies.

Problem 2

The requested plots:



Problem 3

The code for the function *train_lda* is the flowing:

```
def train_lda(X,Y):
    ''' Trains a linear discriminant analysis
    Definition:  w, b   = train_lda(X,Y)
    Input:       X       -  DxN array of N data points with D features
                 Y       -  1D array of length N of class labels {-1, 1}
    Output:      w       -  1D array of length D, weight vector
                 b       -  bias term for linear classification
    '''
    # your code here
    # hint: use the scipy/numpy function sp.cov
    w_p = X.T[Y==1].mean(axis=0)
    w_m = X.T[Y==-1].mean(axis=0)
    n = len(w_p)
    S_B = (w_p - w_m).reshape(n,1).dot((w_p - w_m).reshape(1,n))
    S_W = np.cov(X.T[Y==1], rowvar = False, bias = 1)
    S_W += np.cov(X.T[Y==-1], rowvar = False, bias = 1)
    return np.dot(np.linalg.inv(S_W), (w_p - w_m)), 0
```