# Homework 01 LaTeX Sheet

Cameron Brenner

2/13/2019

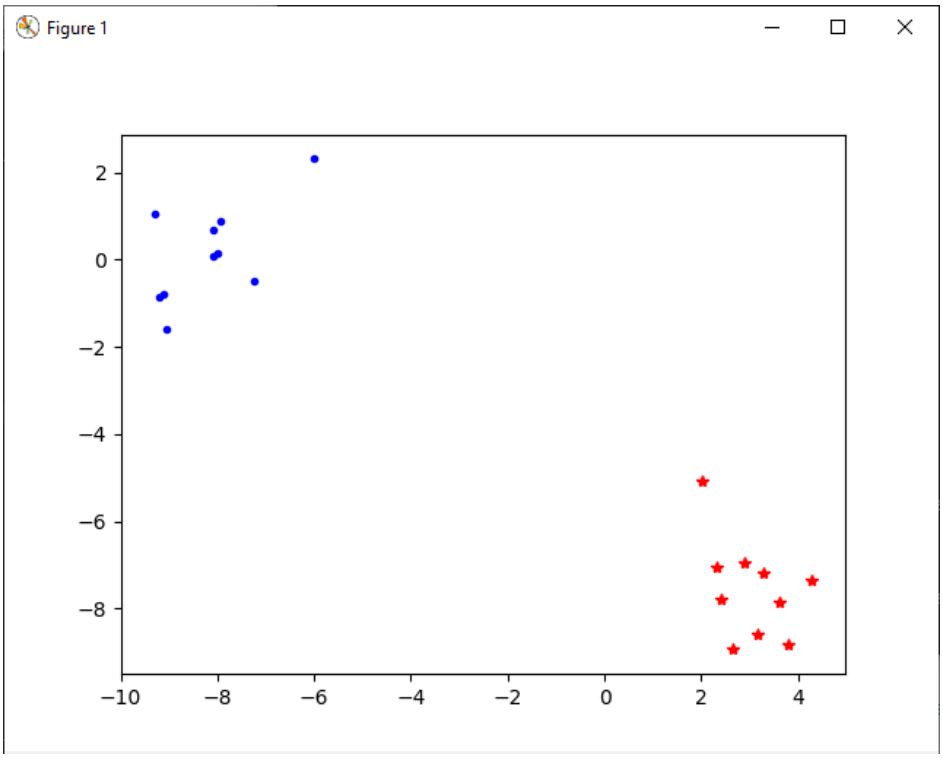
## 1 Problem 1.4

### 1.1 Part A.

In part A. of Exercise 1.4 we are asked to generate 20 random data points, to plot them. This is done with the make_blobs function in sklearn.

```
N = 20

# data
X, y = make_blobs(n_samples=N, centers=2, n_features=2)
y[y==0] = -1  # replace the zeros
X = np.append(np.ones((N,1)), X, 1)   # add a column of ones
```

N is the number of data points that we desire.Now to plot the data we use the plot function from matplotlib



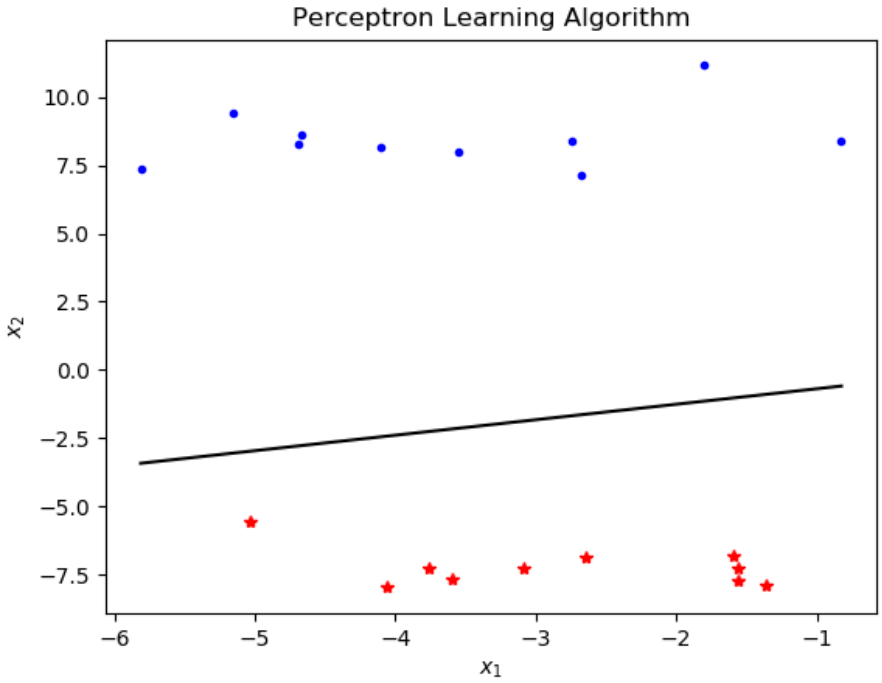
## 1.2 Part B.

In part B. we are asked to run the perceptron learning algorithm on the data set and report the number of iterations, as well as the final plot. This is the code that uses the PLA.

```
# coding: utf-8

# In[45]:


# Some attempt to do the PLA

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    plt.show()


# In[57]:


def main():
    itlst = []
    #for x in range(100):
    N = 20

        # data
    X, y = make_blobs(n_samples=N, centers=2, n_features=2)
    y[y==0] = -1  # replace the zeros
    X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

    # initialize the weigths to zeros
    w = np.zeros(3)
    it = 0
    pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
    while classification_error(w, X, y) != 0:
        it += 1
```

```
        # Pick random misclassified point
        x, s = choose_miscl_point(w, X, y)
        # Update weights
        w += s*x
    pltPer(X,y,w)
    print("Total iterations: " + str(it))
    itlst.append(it)
    #print(itlst)

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

This is the final plot that shows $g$

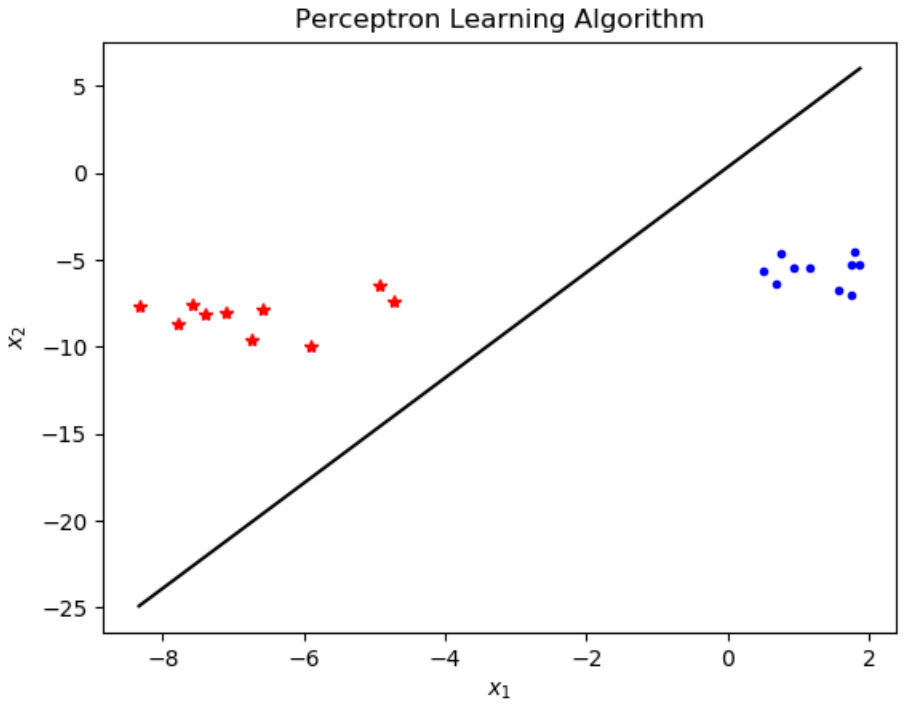



This is the number of iterations it took.

```
C:\Users\Cameron\Python\Python37-32>python hw01partb.py
hw01partb.py:21: RuntimeWarning: invalid value encountered in double_scalars
  m, b = -W[1]/W[2], -W[0]/W[2]
20
10
2
10
0
Total iterations: 4
```

## 1.3 Part C.

In part C. we are asked to run it on another random set of size 20. The same code as above is being used as it generates a new set every time it is run.




This is how many times the algorithm was iterated.

```
C:\Users\Cameron\Python\Python37-32>python hw01partb.py
hw01partb.py:21: RuntimeWarning: invalid value encountered in double_scalars
  m, b = -W[1]/W[2], -W[0]/W[2]
20
10
5
0
Total iterations: 3
```

## 1.4 Part D.

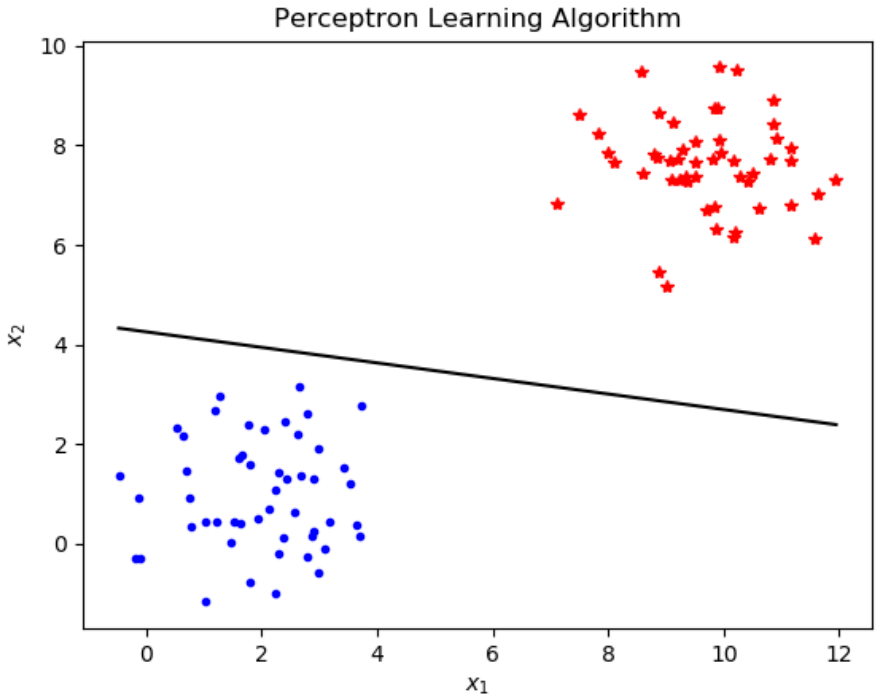
In part D. we are asked to change the sample size from $N = 20$ to $N = 100$, meaning we will be using a sample of 100 random points. Note in the code we change $N = 20$ to $N = 100$

```
# coding: utf-8

# In[45]:


# Some attempt to do the PLA

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
```

```
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    plt.show()


# In[57]:


def main():
    itlst = []
    #for x in range(100):
    N = 100

        # data
    X, y = make_blobs(n_samples=N, centers=2, n_features=2)
    y[y==0] = -1  # replace the zeros
    X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

    # initialize the weigths to zeros
    w = np.zeros(3)
    it = 0
    pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
    while classification_error(w, X, y) != 0:
        it += 1
        # Pick random misclassified point
        x, s = choose_miscl_point(w, X, y)
        # Update weights
        w += s*x
    pltPer(X,y,w)
    print("Total iterations: " + str(it))
    itlst.append(it)
    #print(itlst)

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt
```

```
def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

Here is the final plot of the PLA on this data set.




Here is the number of iterations that the PLA went through


## 1.5 Part E.

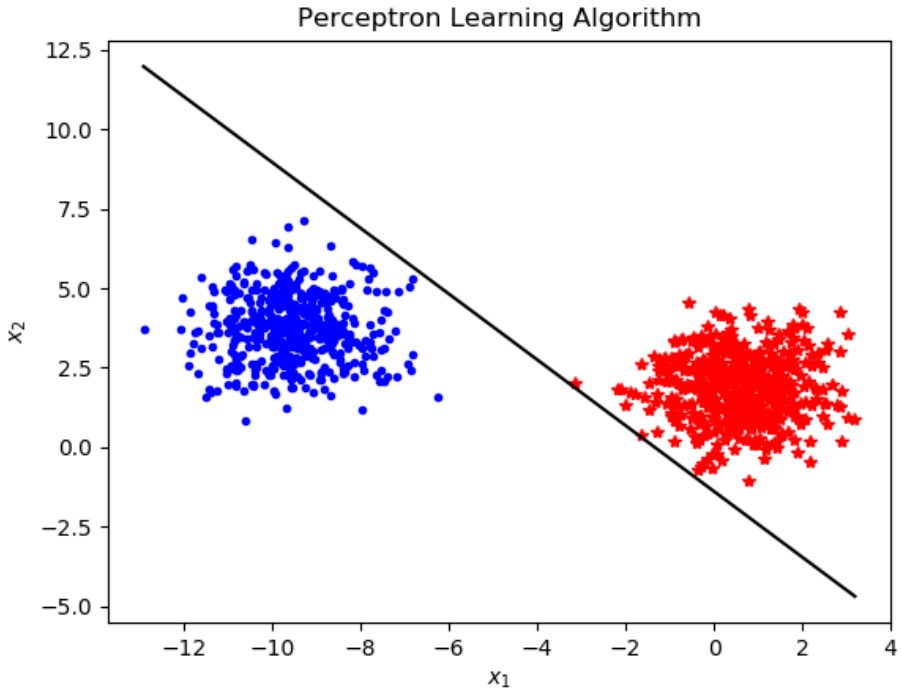
In Part E. we are asked to change the sample size to 1,000. This means changing $N = 100$ to $N = 1000$ in the code.

```
# coding: utf-8
```

```
# In[45]:


# Some attempt to do the PLA

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    plt.show()


# In[57]:


def main():
    itlst = []
    #for x in range(100):
    N = 1000

        # data
    X, y = make_blobs(n_samples=N, centers=2, n_features=2)
    y[y==0] = -1  # replace the zeros
    X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

    # initialize the weigths to zeros
    w = np.zeros(3)
    it = 0
    pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
    while classification_error(w, X, y) != 0:
        it += 1
        # Pick random misclassified point
        x, s = choose_miscl_point(w, X, y)
        # Update weights
        w += s*x
    pltPer(X,y,w)
    print("Total iterations: " + str(it))
```

```
    itlst.append(it)
    #print(itlst)

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

Here is the final plot of the PLA on this data set.




Here is the number of iterations that the PLA went through


This is odd, because if there is a larger data set, one would expect the iteration count to be higher. I believe that this is a result from the numbers generating far apart from each other. I ran the code a few times just to make sure and I would get either very low numbers, when they would generate far apart from each other, or it would iterate infinitely because there was some overlap in the points.

### 1.6 Part F.

In Part F. we are asked to change the algorithm such that $X_n \in \mathbb{R}^{10}$. This means that we have to change the number of dimensions we are in from $d = 2$ to $d = 10$ when we are generating data points. Obviously we cannot plot in ten dimensions so those parts are commented out. Notice *n_features* is now 10

```
# coding: utf-8

# In[45]:


# Some attempt to do the PLA

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    plt.show()


# In[57]:


def main():
    itlst = []
    #for x in range(100):
    N = 1000

        # data
    X, y = make_blobs(n_samples=N, centers=2, n_features=10)
```

```
    y[y==0] = -1  # replace the zeros
    X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

    # initialize the weigths to zeros
    w = np.zeros(11)
    it = 0
    #pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
    while classification_error(w, X, y) != 0:
        it += 1
        # Pick random misclassified point
        x, s = choose_miscl_point(w, X, y)
        # Update weights
        w += s*x
    #pltPer(X,y,w)
    print("Total iterations: " + str(it))
    itlst.append(it)
    #print(itlst)

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

Here is the number of iterations

```
C:\Users\Cameron\Python\Python37-32>python hw01partf.py
1000
484
0
Total iterations: 2
```

This is also odd, and I think we are running into a similar problem as the previous part where make_blobs is generating the numbers too far apart from each other, causing the PLA to go through a small number of iterations, usually one or two, even with such a large data set.

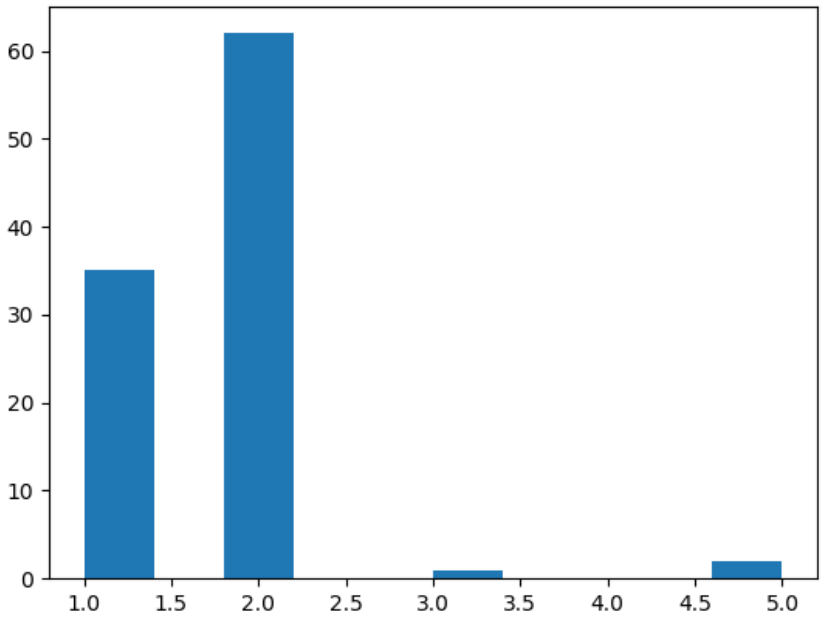
## 1.7 Part G.

In Part G. we are asked to repeat the experiment from the previous part 100 times. This means putting the PLA, as well as the data generator, in a loop and having it repeat 100 times. It also asks for a histogram of the data.

```
# coding: utf-8

# In[45]:


# Some attempt to do the PLA

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

def pltPer(X, y, W):
    f = plt.figure()
    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")


# In[57]:


def main():
    itlst = []
    for x in range(100):
        N = 100

        # data
        X, y = make_blobs(n_samples=N, centers=2, n_features=10)
        y[y==0] = -1  # replace the zeros
        X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

        # initialize the weigths to zeros
        w = np.zeros(11)
        it = 0
        #pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
        while classification_error(w, X, y) != 0:
            it += 1
```

```
            # Pick random misclassified point
            x, s = choose_miscl_point(w, X, y)
            # Update weights
            w += s*x
        #pltPer(X,y,w)
        #print("Total iterations: " + str(it))
        itlst.append(it)
    print(itlst)
    plt.hist(itlst)
    plt.show()


def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    #print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

main()
```

Because of the issues involving make_blobs, I predict the histogram is not going to be varied greatly.




## 1.8 Part H

We are asked to summarize our conclusions with respect to accuracy and running time as a function of $N$ and $d$. I believe that accuracy and running time increase as the number of data points increase, as was shown from the jump from $N = 20$ to $N = 100$, but that was not the case when $N = 1000$ or when $d$ was changed from $d = 2$ to $d = 10$, and I think this is an issue with the make_blobs function.