Perceptron Implementation

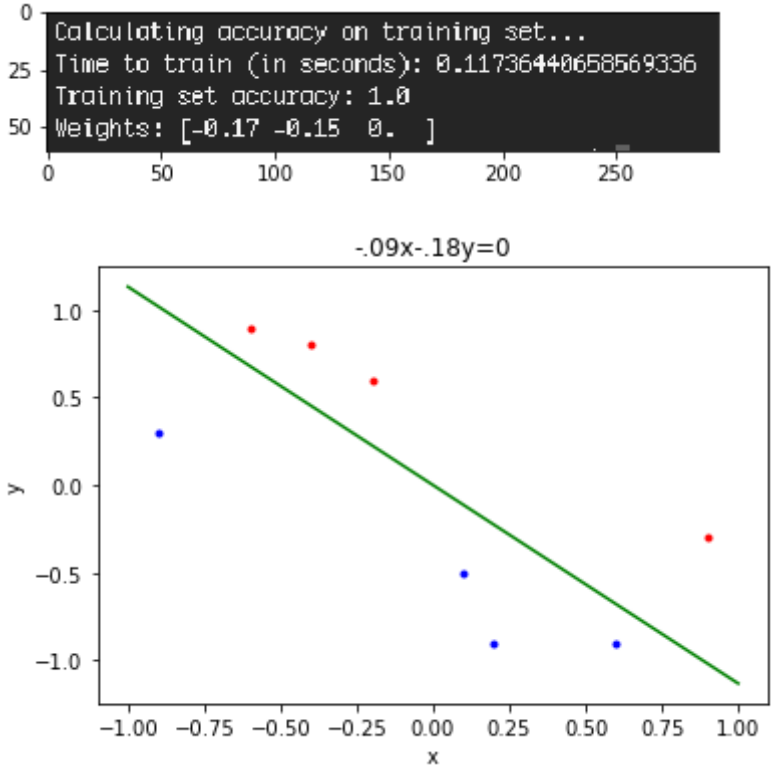
By Alexandra Hurst

In [93]:
```
import numpy as np
from matplotlib import pyplot as plt
import cv2
from IPython.display import HTML, display
import tabulate
import pandas as pd
```

Linearly Seperable Data

In [75]:
```
x_a=[.9,-.2,-.4,-.6]
x_b=[-.9,.2,.1,.6]
y_a=[-.3,.6,.8,.9]
y_b=[.3,-.9,-.5,-.9]
```

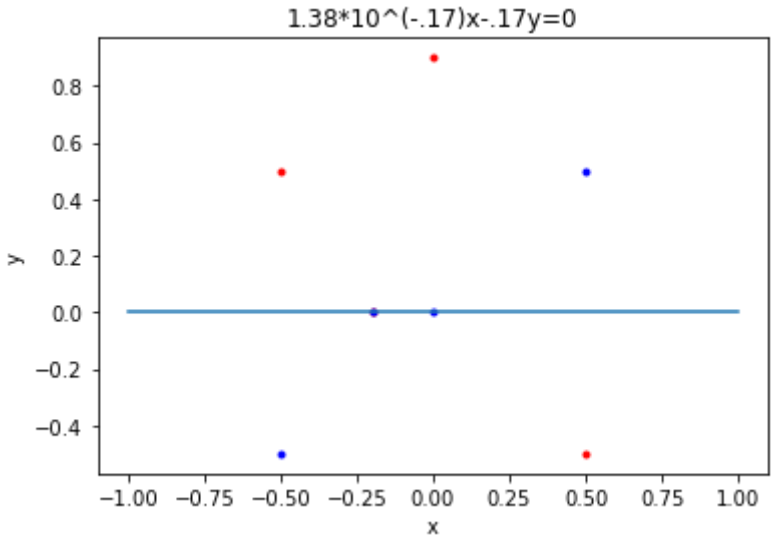
```
Calculating accuracy on training set...
Time to train (in seconds): 0.11736440658569336
Training set accuracy: 1.0
Weights: [-0.17 -0.15  0.  ]
```



Not Linearly Seperable Data

In [76]:
```
x_a=[-.5, .5, 0, -.2]
x_b=[-.5, .5, 0, -.2]
y_a=[.5, -.5, .9, 0]
y_b=[-.5, .5, -.0, 0]
```

```
Calculating accuracy on training set...
Time to train (in seconds): 0.11736440658569336
Training set accuracy: 1.0
Weights: [-0.17 -0.15  0.  ]
```



Stopping Criteria

The code either stops running after 1000 epochs or if there are 6 epochs in a row where the average value of old_w-updated_w is less than .001. This shows overall convergence over time before stopping rather than stopping when one iteration may progress slower. Calculating average change rather than total change avoids penalizing larger datasets

Learning

The number of epochs the code takes seems to be inversely related to the training rate. For learning rate = .1 most small datasets converge in 10-20 epochs. With learning rate = .001, the 8 points in my arff file still converged quickly, but larger data sets (like the voting dataset) 150-200 iterations to converge.

Voting Dataset

In [82]:
```
table = [["Epochs",22,22,11,9,3],
         ["Training Time",.380285,.55384,.33049,.20448,.00992],
         ["Training Accuracy",.96784,.9254,.97826,.97985,.95031],
         ["Testing Accuracy",.928057,.94964,.956788,.96602,.96237]]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

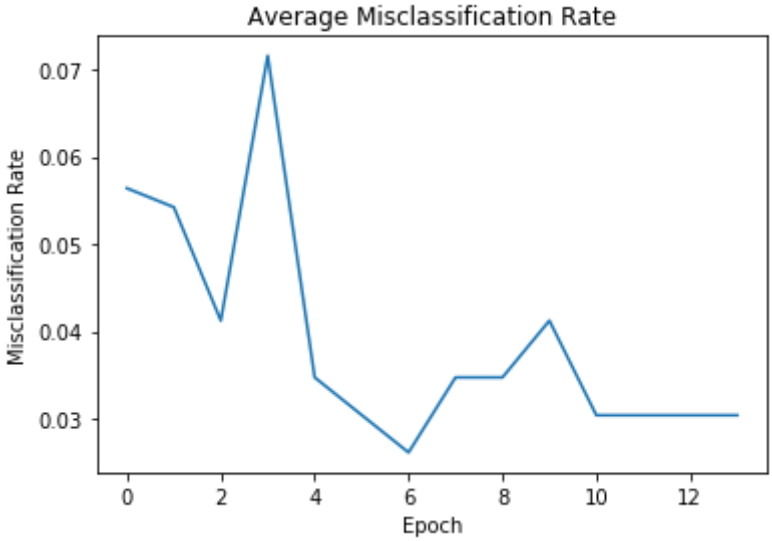
Epochs	22	22	11	9	3
Training Time	0.380285	0.55384	0.33049	0.20448	0.00992
Training Accuracy	0.96784	0.9254	0.97826	0.97985	0.95031
Testing Accuracy	0.928057	0.94964	0.956788	0.96602	0.96237

In [85]:
```
table = [["Avg Epochs",13.4],
         ["Avg Training Time",.295803],
         ["Avg Training Accuracy",.960332],
         ["Avg Testing Accuracy",.952575]]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

Avg Epochs	13.4
Avg Training Time	0.295803
Avg Training Accuracy	0.960332
Avg Testing Accuracy	0.952575

The testing showed "physician-fee-freeze" to be the most relevant to whether someone is Republican or Democrat. Crime, education-spending, handicapped-infants, and right-to-sue had the least effect.

In [90]:
```
accuracy=[.9436,.94577,.95878,.9284,.96529,.9696,.9739,.96529,.96529,\
          .95878,.96963,.96963,.96963,.96963]
plt.plot(np.ones_like(accuracy)-accuracy)
plt.title("Average Misclassification Rate")
plt.xlabel("Epoch")
plt.ylabel("Misclassification Rate")
plt.show()
```



Wholesale Customer Dataset

This dataset describes annual monetary spending on different categories. Each row refers to one client of a wholesale distributor in Portugul. Each customer is either from the restaurant industry or the retail industry, which is what I will be predicting. Training on this set yields the following results:

In [98]:
```
table = [["Number of Instances",440],
         ["Number of Attributes", 8],
         ["Training Time",2.49157],
         ["Training Accuracy",.452267]]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

Number of Instances	440
Number of Attributes	8
Training Time	2.49157
Training Accuracy	0.452267

Since this isn't exactly great accuracy, I will instead look at predicting the region of each customer. There are 3 options, I will compare all from region 1 to all not from region one by modifying from which column I load the target values and then changing all 2s and 3s to 0s in my perceptron algorithm.

In [99]:
```
table = [["Number of Instances",440],
         ["Number of Attributes", 8],
         ["Training Time",.1328],
         ["Training Accuracy",.4491]]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

Number of Instances	440
Number of Attributes	8
Training Time	0.1328
Training Accuracy	0.4491

This trained a lot faster than the other target, but the accuracy still isn't great. My conclusion is that despite having multiple columns, nothing in this dataset is linearly separable, and hence not very well compatible with perceptron.

In [ ]:
```
```