Table of Contents

In [1]:
```
%%javascript
/*************************************************************************************
**************
Known Mathjax Issue with Chrome - a rounding issue adds a border to the right of
mathjax markup
https://github.com/mathjax/MathJax/issues/1300
A quick hack to fix this based on stackoverflow discussions:
http://stackoverflow.com/questions/34277967/chrome-rendering-mathjax-equations-wi
th-a-trailing-vertical-line
*************************************************************************************
*************/

$('.math>span').css("border-left-color","transparent")
```

In [2]:
```
%reload_ext autoreload
%autoreload 2
```

MIDS - w261 Machine Learning At Scale

Course Lead: Dr James G. Shanahan (**email** Jimi via James.Shanahan *AT* gmail.com)

Assignment - HW11

Name: Nilesh Bhoyar **Class:** MIDS w261 (Section *Your Section Goes Here*, e.g., Fall 2016 Group 1)
Email: nilesh.bhoyar@iSchool.Berkeley.edu
StudentId 26302327 **End of StudentId**
Week: 11

NOTE: please replace `1234567` with your student id above

Due Time: HW is due the Wednesday of the following week by 8AM (West coast time). I.e., Wednesday, April 12, 2017 in the case of this homework.

Instructions

MIDS UC Berkeley, Machine Learning at Scale
DATSCIW261 ASSIGNMENT #11

Version 2017-3-16

IMPORTANT

This homework can be completed locally on your computer.

=== INSTRUCTIONS for SUBMISSIONS ===

Follow the instructions for submissions carefully.

Each student has a `HW-<user>` repository for all assignments.

Click this link to enable you to create a github repo within the MIDS261 Classroom:
https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8 (https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8)
and follow the instructions to create a HW repo.

Push the following to your HW github repo into the master branch:

- Your local HW6 directory. Your repo file structure should look like this:

```
HW-<user>
    --HW3
        |__MIDS-W261-HW-03-<Student_id>.ipynb
        |__MIDS-W261-HW-03-<Student_id>.pdf
        |__some other hw3 file
    --HW4
        |__MIDS-W261-HW-04-<Student_id>.ipynb
        |__MIDS-W261-HW-04-<Student_id>.pdf
        |__some other hw4 file
    etc..
```

2 Useful References

- Karau, Holden, Konwinski, Andy, Wendell, Patrick, & Zaharia, Matei. (2015). Learning Spark: Lightning-fast big data analysis. Sebastopol, CA: O'Reilly Publishers.
- Hastie, Trevor, Tibshirani, Robert, & Friedman, Jerome. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Stanford, CA: Springer Science+Business Media. (Download for free here (http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf))
- https://www.dropbox.com/s/ngomebw1koujs2d/classificationISLBook-Logistic-Regression-LDA-NaiveBayes.pdf?dl=0 (https://www.dropbox.com/s/ngomebw1koujs2d/classificationISLBook-Logistic-Regression-LDA-NaiveBayes.pdf?dl=0)

HW Problems

HW11.0: Broadcast versus Caching in Spark

Q: **What is the difference between broadcasting and caching data in Spark? Give an example (in the context of machine learning) of each mechanism (at a highlevel). Feel free to cut and paste code examples from the lectures to support your answer.**

Q: **Review the following Spark-notebook-based implementation of KMeans and use the broadcast pattern to make this implementation more efficient. Please describe your changes in English first, implement, comment your code and highlight your changes (write all your code in this notebook):**_

Notebook https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0 (https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0)

Notebook via NBViewer http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb (http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb)

Q: **What is the difference between broadcasting and caching data in Spark? Give an example (in the context of machine learning) of each mechanism (at a highlevel). Feel free to cut and paste code examples from the lectures to support your answer.**

Broadcast variables can be shared and accessed from across the cluster, similar to accumulators. But they're opposite from accumulators in that they can't be modified by executors. The driver creates a broadcast variable, and executors read it.

While on other hand caching, caches the RDD in spark clusture wide memory (not local one). It means 60% of data can be cached on one of the node while other 40% can be distributed. so not all nodes have complete RDD i.e. it is distributed.

We will use Broadcasting (it will put it in memory) when we want cache information which is

- Book keeping information for the learning algorithm that we wish to share with each executor
- This information changes after each iteration
- Need the entire weight vector.

In K-means, centroids changes every iteration but within mappers we want all mappers to read centroid same information. We will broadcast centroid after every update and read it in mappers.

In [2]:
```
%matplotlib inline
import numpy as np
import pylab
import json
size1 = size2 = size3 = 100
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('data.csv',data,delimiter = ',')
```

In [3]:
```
!hdfs dfs -rm -r HW11
!hdfs dfs -mkdir HW11
!hdfs dfs -rm -r HW11/HW110/
!hdfs dfs -mkdir HW11/HW110/
!hdfs dfs -copyFromLocal data.csv HW11/HW110/
```

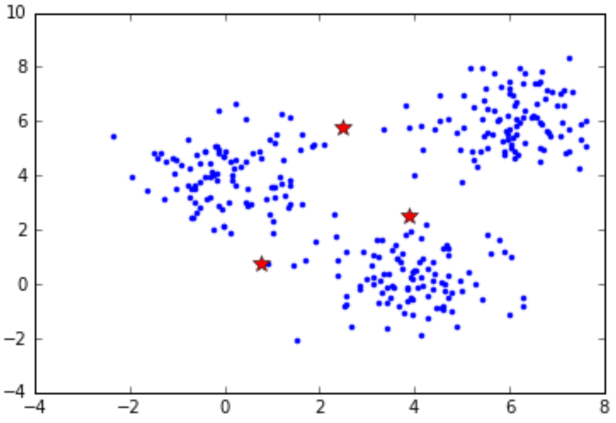
```
17/08/01 03:03:59 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele
tion interval = 5760 minutes, Emptier interval = 360 minutes.
Moved: 'hdfs://nn-ia.s3s.altiscale.com:8020/user/nileshbhoyar/HW11' to trash at:
hdfs://nn-ia.s3s.altiscale.com:8020/user/nileshbhoyar/.Trash/Current
rm: `HW11/HW110/': No such file or directory
```

In [4]:
```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt
```

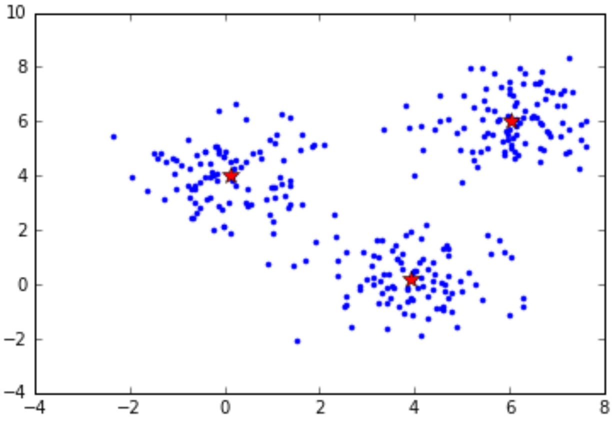
In [5]:
```
#plot centroids and data points for each iteration
def plot_iteration(means):
    pylab.plot(samples1[:, 0], samples1[:, 1], '.', color = 'blue')
    pylab.plot(samples2[:, 0], samples2[:, 1], '.', color = 'blue')
    pylab.plot(samples3[:, 0], samples3[:, 1],'.', color = 'blue')
    pylab.plot(means[0][0], means[0][1],'*',markersize =10,color = 'red')
    pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 'red')
    pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 'red')
    pylab.show()
```

In [6]:
```
# START STUDENT CODE 11.0
# (ADD CELLS AS NEEDED)
def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])
    wx = 1/abs(sqrt(x[0]**2 + x[1]**2))
    closest_centroid_idx = np.sum((x - wBroadcast.value)**2 * wx, axis=1).argmin(
)

    return (closest_centroid_idx,(x,1))

K = 3


D = sc.textFile("HW11/HW110/data.csv").cache()

def calcError(line):

    center = centroids[nearest_centroid(line)[0]]

    point = np.array([float(f) for f in line.split(',')])

    return sqrt(sum([x**2 for x in (point - center)]))
iter_num = 0
for k in [1, 10, 20, 30, 40, 50, 100]:
    # Initialization: initialization of parameter is fixed to show an example
    centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
    wBroadcast = sc.broadcast(centroids)
    for i in range(k):
        res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[
1])).collect()

        res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
        centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by clus
ter size
        centroids = centroids_new
        wBroadcast = sc.broadcast(centroids)

    print "Iteration" + str(k)
    WSSSE = D.map(calcError).reduce(lambda x, y: x + y)

    print("Within Set Sum of Squared Error = " + str(WSSSE))
    print centroids
    plot_iteration(centroids)
# END STUDENT CODE 11.0
```

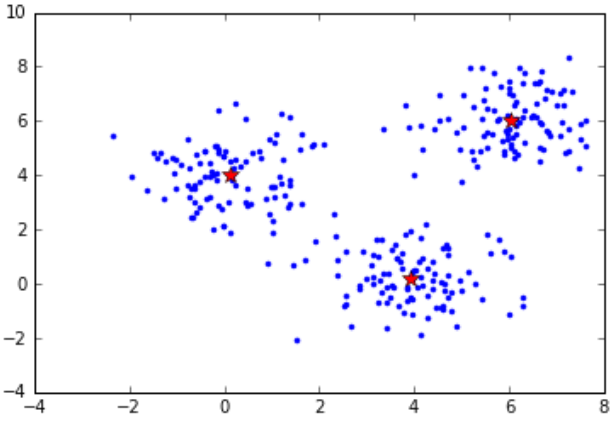
```
Iteration1
Within Set Sum of Squared Error = 847.682060861
[[ 0.76460264  0.77703177]
 [ 3.90201475  2.52204535]
 [ 2.47189914  5.79766002]]
```



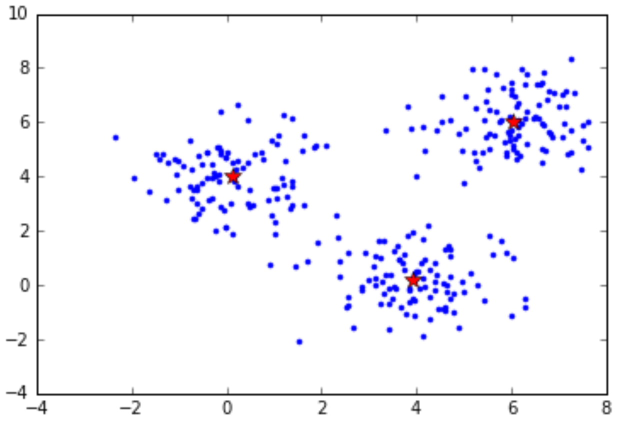
```
Iteration10
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



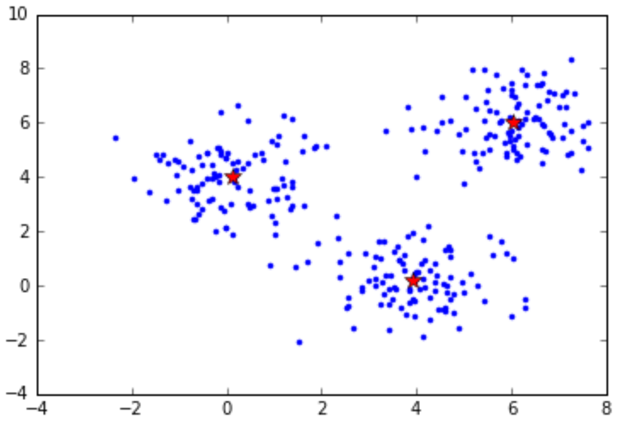
```
Iteration20
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



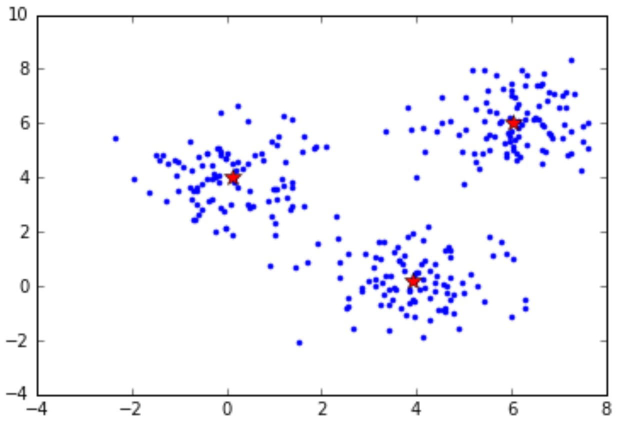
```
Iteration30
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



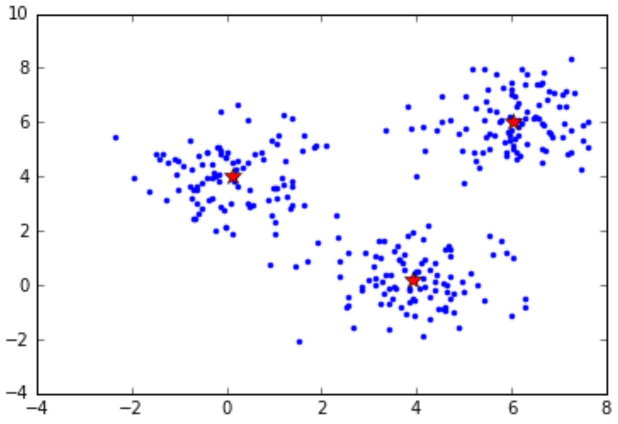
```
Iteration40
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



```
Iteration50
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



```
Iteration100
Within Set Sum of Squared Error = 371.452412819
[[ 0.11556139  4.01101192]
 [ 3.90583797  0.20422471]
 [ 6.02605209  6.02935042]]
```



HW11.1 Loss Functions

In the context of binary classification problems, does the linear SVM learning algorithm yield the same result as a L2 penalized logistic regesssion learning algorithm?

In your reponse, please discuss the loss functions, and the learnt models, and separating surfaces between the two classes.

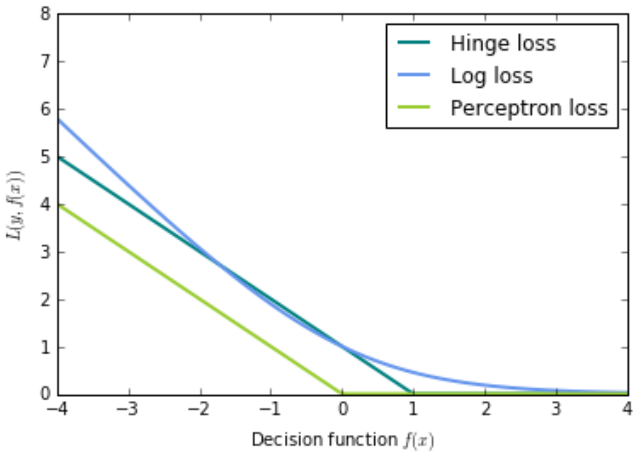
In the context of binary classification problems, does the linear SVM learning algorithm yield the same result as a perceptron learning algorithm?

[OPTIONAL]: generate an artifical binary classification dataset with 2 input features and plot the learnt separating surface for both a linear SVM and for logistic regression. Comment on the learnt surfaces. Please feel free to do this in Python (no need to use Spark).

In the context of binary classification problems, does the linear SVM learning algorithm yield the same result as a L2 penalized logistic regesssion learning algorithm?

Answer: Unlike logistic regression, Linear Support vector machines, primary intuition wasn't concretely probabilistic. The main intuition was maximizing the margin between two distributions.The basic intuition of Logistic Regression was modeling the conditional probability of one of the classes P(Y=1 | X= x) given and by minimizing the log loss function is equivalent to maximizing the likelihood. So we arrive to the logistic loss function. Afterwards all this add regularization term we arrive to a function that is very similar to the SVM function but with the log loss instead of the hinge loss.

In [7]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

xmin, xmax = -4, 4
xx = np.linspace(xmin, xmax, 100)
lw = 2

plt.plot(xx, np.where(xx < 1, 1 - xx, 0), color='teal', lw=lw,
         label="Hinge loss")
plt.plot(xx, np.log2(1 + np.exp(-xx)), color='cornflowerblue', lw=lw,
         label="Log loss")
plt.plot(xx, -np.minimum(xx, 0), color='yellowgreen', lw=lw,
         label="Perceptron loss")
plt.ylim((0, 8))
plt.legend(loc="upper right")
plt.xlabel(r"Decision function $f(x)$")
plt.ylabel("$L(y, f(x))$")
plt.show()
```



For the logistic loss, while training the classifier, even for correctly classified training examples there will be some error calculated (see the right part of the graph) . Which correlates with the initial intuition of logistic regression is modeling "probabilities". In other words Logistic regression is modeled to assume that the training data might contain some mistakes in annotation as well. While in the case of Hinge loss no error will be calculated

So they are different

In the context of binary classification problems, does the linear SVM learning algorithm yield the same result as a perceptron learning algorithm?

Answer:Perceptron algorithm does not attempt to minimize the margin;rather it is finished as soon as it find the weight vector for hyperplane that separates two classes.Perceptron does not penalize the correct guess while hinge loss does.

Also hyperplane in perceptron depends upon the sequence in which records are processed while in SVM it depends only on support vectors.

HW11.2 Gradient descent

In the context of logistic regression describe and define three flavors of penalized loss functions. Are these all supported in Spark MLLib (include online references to support your answers)?

Describe probabilistic interpretations of the L1 and L2 priors for penalized logistic regression (HINT: see synchronous slides for week 11 for details)

Response

First let put the equations for logistic regression, loss function and it's gradient

Logistic regression esitmated probability function;

$\hat{p} = h_\theta(X) = \sigma(\Theta^T * X)$

Where $\sigma(.)$ is a sigmoid function (i.e., S-shaped) that outputs a number between 0 and 1.

Logitic regression closed form solution is found using log-loss which can be be calculated as

$J(\mathbf{\Theta}) = -\frac{1}{m} * \sum_{n=1}^{m} [y^i \log \overset{\wedge}{p^i} + (1 - y^i)(1 - \log \overset{\wedge}{p^i})]$

But this does not have closed form solution so we will have to go for gradient way And the equation for the gradients:

$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \dfrac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)}\right) \mathbf{x}^{(i)}$

We can add L1/L2 loss penalty as explained on slide #360 https://docs.google.com/presentation/d/1qTBzcXAQ6Y-njiUqv9sHNMk3zBX-Pm8BidWrSwtrUP8/edit#slide=id.p520 (https://docs.google.com/presentation/d/1qTBzcXAQ6Y-njiUqv9sHNMk3zBX-Pm8BidWrSwtrUP8/edit#slide=id.p520)

L1

$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \dfrac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)}\right) \mathbf{x}^{(i)} + \lambda\theta^T$

L2 form

$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \dfrac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)}\right) \mathbf{x}^{(i)} + \lambda * sign(\theta^T)$

Elastic Net

$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \dfrac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)}\right) \mathbf{x}^{(i)} + (\alpha * \lambda * sign(\theta^T) + (1 - \alpha) * \lambda * sign(\theta^T)$

As per the current documentation https://spark.apache.org/docs/latest/mllib-linear-methods.html#regularizers (https://spark.apache.org/docs/latest/mllib-linear-methods.html#regularizers)

above mentioned 3 forms are supported.

The key idea is that you increase the weights as much as necessary to reach the global minimum.

you are doing MAP estimation if you can interpret the regularization term R(θ)

as the log of your prior.

Describe probabilistic interpretations of the L1 and L2 priors for penalized logistic regression (HINT: see synchronous slides for week 11 for details)

The first term in equation is the log of the (conditional) likelihood. Thus, we are maximizing the posterior estimation and regularization term R(θ) as the log of your prior.

The L2 regularizer can be interpreted as the log of a Gaussian prior, and the L1 regularizer can be interpreted as the log of a Laplace prior.

Reference https://www.quora.com/What-is-the-probabilistic-interpretation-of-regularized-logistic-regression (https://www.quora.com/What-is-the-probabilistic-interpretation-of-regularized-logistic-regression)

HW11.3 Logistic Regression

Generate 2 sets of linearly separable data with 100 data points each using the data generation code provided below and plot each in separate plots. Call one the training set and the other the testing set.

```
def generateData(n):
 """
  generates a 2D linearly separable dataset with n samples.
  The third element of the sample is the label
 """
 xb = (rand(n)*2-1)/2-0.5
 yb = (rand(n)*2-1)/2+0.5
 xr = (rand(n)*2-1)/2+0.5
 yr = (rand(n)*2-1)/2-0.5
 inputs = []
 for i in range(len(xb)):
  inputs.append([xb[i],yb[i],1])
  inputs.append([xr[i],yr[i],-1])
 return inputs
```

Modify this data generation code to generating non-linearly separable training and testing datasets (with approximately 10% of the data falling on the wrong side of the separating hyperplane. Plot the resulting datasets.

NOTE: For the remainder of this problem please use the non-linearly separable training and testing datasets.

Using MLLib train up a LASSO logistic regression model with the training dataset and evaluate with the testing set. What a good number of iterations for training the logistic regression model? Justify with plots and words.

Derive and implement in Spark a weighted LASSO logistic regression. Implement a convergence test of your choice to check for termination within your training algorithm .

Weight the above training dataset as follows: Weight each example using the inverse vector length (Euclidean norm):

> weight(X)= 1/||X||,
>
> where ||X|| = SQRT(X.X)= SQRT(X1^2 + X2^2)
>
> Here X is vector made up of X1 and X2.

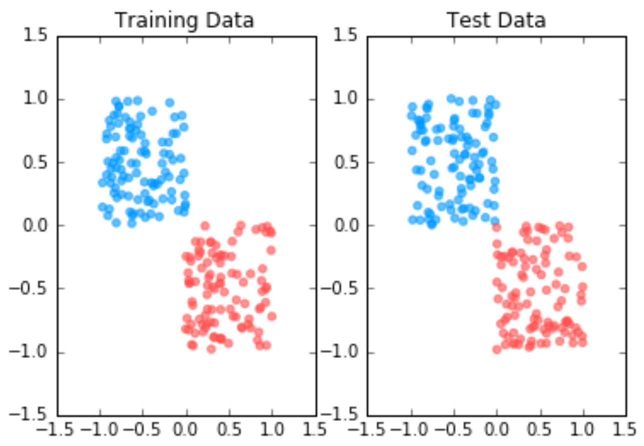
Evaluate your homegrown weighted LASSO logistic regression on the test dataset. Report misclassification error (1 - Accuracy) and how many iterations does it took to converge.

Does Spark MLLib have a weighted LASSO logistic regression implementation. If so use it and report your findings on the weighted training set and test set.

In [8]:
```
import numpy as np
def generateData(n):
 """
  generates a 2D linearly separable dataset with n samples.
  The third element of the sample is the label
 """
 xb = (np.random.rand(n)*2-1)/2-0.5
 yb = (np.random.rand(n)*2-1)/2+0.5
 xr = (np.random.rand(n)*2-1)/2+0.5
 yr = (np.random.rand(n)*2-1)/2-0.5
 inputs = []
 for i in range(len(xb)):
  inputs.append([xb[i],yb[i],1])
  inputs.append([xr[i],yr[i],-1])
 return inputs
```

In [9]:
```
train_data = np.array(generateData(100))
test_data = np.array(generateData(100))
```

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
def plot_data(data):
    data_colors = ['#0099ff' if x==1 else '#ff5050' for x in data[:,2]]
    plt.scatter(data[:,0],data[:,1],color=data_colors,alpha=0.6)
plt.subplot(1,2,1)
plt.title("Training Data")
plot_data(train_data )
plt.subplot(1,2,2)
plt.title("Test Data")
plot_data(test_data )
```

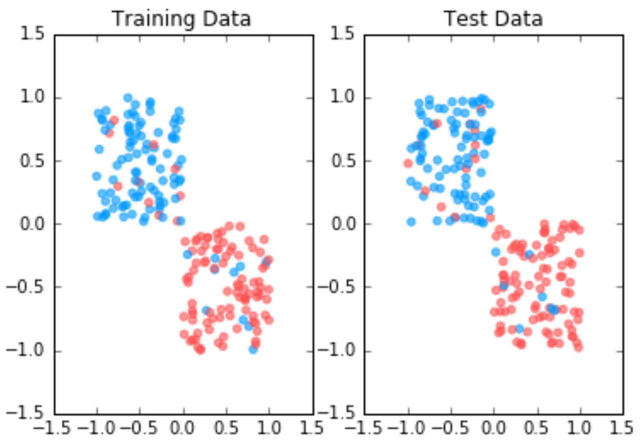


In [11]:
```
import csv
def data_save(X,fileName):

    data = zip(X[:,-1], X[:,-3], X[:,-2])
    with open(fileName,'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

In [12]:
```
def generate_non_separable(n):
    xb = (np.random.rand(n)*2-1)/2-0.5
    yb = (np.random.rand(n)*2-1)/2+0.5
    xr = (np.random.rand(n)*2-1)/2+0.5
    yr = (np.random.rand(n)*2-1)/2-0.5
    inputs = []
    for i in range(len(xb)):
      inputs.append([xb[i],yb[i],1])
      inputs.append([xr[i],yr[i],-1])
    inputs = np.array(inputs)
    wrongidx = np.random.choice(range(2*n),np.round(0.1*2*n),replace=False)
    inputs[wrongidx,-1]=-inputs[wrongidx,-1]
    return inputs
```

In [13]:
```
train_data = generate_non_separable(100)
test_data = generate_non_separable(100)
plt.subplot(1,2,1)
plt.title("Training Data")
plot_data(train_data )
plt.subplot(1,2,2)
plt.title("Test Data")
plot_data(test_data )
y_test = test_data[:,-1]
y_test[y_test == -1 ] = 0
```

```
/mnt/ephemeral1/jupyter/anaconda2/envs/py27/lib/python2.7/site-packages/ipykerne
l/__main__.py:11: DeprecationWarning: using a non-integer number instead of an i
nteger will result in an error in the future
```



In [14]:
```
data_save(train_data,"data_logistic_train.txt")
data_save(test_data,"data_logistic_test.txt")
!hdfs dfs -rm -r HW11/HW113/
!hdfs dfs -mkdir HW11/HW113/
!hdfs dfs -copyFromLocal data_logistic_train.txt HW11/HW113/
!hdfs dfs -copyFromLocal data_logistic_test.txt HW11/HW113/
```

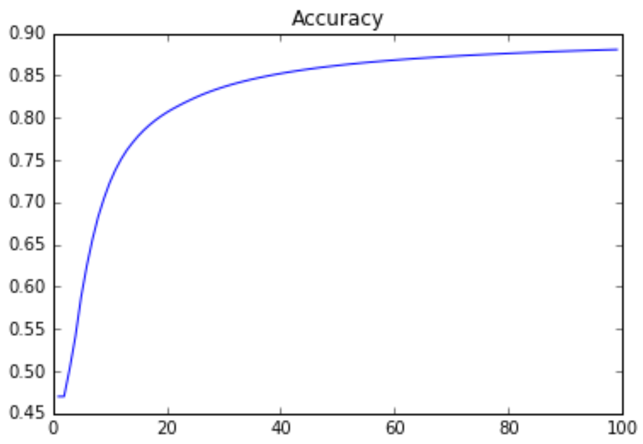
```
rm: `HW11/HW113/': No such file or directory
```

In [15]:
```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticReg
ressionModel,LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(point):
    label = 1 if point[-1] ==1 else 0
    return LabeledPoint(label, point[:-1])  #the input shoul be {0,1} instead of
{-1,1}
# Load and parse the data
def parsePoint_pred(point):

    return  point[:-1]
x_axis,y_axis =[],[]
correct, total = 0, 0
train_rdd = sc.parallelize(train_data).map(parsePoint).cache()
test_rdd = sc.parallelize(test_data).map(parsePoint_pred).cache()
for k in range(1,100):
# Build the model
    model_lasso = LogisticRegressionWithSGD.train(train_rdd, regType="l1", interc
ept=True,iterations=k)
    print model_lasso
    #parsedtest = sc.textFile("HW11/HW113//data_logistic_test.txt").map(parsePoin
t_pred)
    Z = model_lasso.predict(test_rdd).collect()

    for pred, label in zip(Z,y_test):
            if pred == label: correct += 1
            total += 1

    x_axis.append(k)
    y_axis.append(1.0*correct/total)
plt.plot(x_axis,y_axis)
plt.title("Accuracy")
```

```
/opt/alti-spark-2.0.2/python/pyspark/mllib/classification.py:313: UserWarning: D
eprecated in 2.0.0. Use ml.classification.LogisticRegression or LogisticRegressi
onWithLBFGS.
  "Deprecated in 2.0.0. Use ml.classification.LogisticRegression or "
```

```
(weights=[-0.191432942435,0.194879926459], intercept=0.758941421369995)
(weights=[-0.309409664784,0.315161312566], intercept=0.6260390295842806)
(weights=[-0.39611358357,0.403699492161], intercept=0.5363544206633)
(weights=[-0.464963664357,0.474112840287], intercept=0.47016841797621556)
(weights=[-0.522144655525,0.53267578653], intercept=0.4186872694103101)
(weights=[-0.57106269442 9,0.58284371528], intercept=0.3772057406416989)
(weights=[-0.613806894442,0.626736203733], intercept=0.342914709485614)
(weights=[-0.651755788893,0.66575218112], intercept=0.314007832415797)
(weights=[-0.685868839702,0.700865555049], intercept=0.2892590089981045)
(weights=[-0.716841639673,0.732782588988], intercept=0.26780079908344395)
(weights=[-0.745195045497,0.762032283028], intercept=0.24899900845465164)
(weights=[-0.771329460953,0.78902141963], intercept=0.2323773891179758)
(weights=[-0.79555948613,0.814069702861], intercept=0.2175702437179935)
(weights=[-0.818136903453,0.837433072367], intercept=0.20429141607835122)
(weights=[-0.839266426449,0.859319679141], intercept=0.19231333479260088)
(weights=[-0.859116790099,0.879901137518], intercept=0.18145245524876147)
(weights=[-0.877828747546,0.89932063979], intercept=0.17155890359039108)
(weights=[-0.895520956137,0.917698930064], intercept=0.16250895579417468)
(weights=[-0.912294389104,0.935138782545], intercept=0.15419947536106826)
(weights=[-0.92823569573,0.951728413026], intercept=0.14654373255966363)
(weights=[-0.943419797638,0.967544115298], intercept=0.13946821637422696)
(weights=[-0.957911920878,0.982652324991], intercept=0.13291017166235958)
(weights=[-0.971769205086,0.997111254158], intercept=0.1268156740662761)
(weights=[-0.9850419913,1.01097219966], intercept=0.12113810909826261)
(weights=[-0.997774862604,1.02428060059], intercept=0.11583695875973699)
(weights=[-1.01000749254,1.03707690055], intercept=0.11087682480561283)
(weights=[-1.02177534242,1.04939725641], intercept=0.10622663599451841)
(weights=[-1.03311023884,1.06127412539], intercept=0.10185899974999262)
(weights=[-1.04404085521,1.07273675476], intercept=0.09774966817104624)
(weights=[-1.05459311598,1.0838115928], intercept=0.09387709532955732)
(weights=[-1.06479053792,1.09452263596], intercept=0.09022206799807304)
(weights=[-1.07465451993,1.10489172365], intercept=0.08676739586366058)
(weights=[-1.08420459041,1.11493878992], intercept=0.08349765025113853)
(weights=[-1.09345861951,1.12468207935], intercept=0.08039894265032968)
(weights=[-1.10243300198,1.13413833321], intercept=0.0774587360947784)
(weights=[-1.11114281547,1.14332295053], intercept=0.07466568380260025)
(weights=[-1.11960195815,1.15225012813], intercept=0.07200949055814002)
(weights=[-1.12782326869,1.16093298289], intercept=0.06948079315562006)
(weights=[-1.13581863144,1.16938365867], intercept=0.06707105689490706)
(weights=[-1.14359906875,1.17761342048], intercept=0.06477248565393706)
(weights=[-1.15117482247,1.18563273733], intercept=0.06257794349176167)
(weights=[-1.15855542592,1.19345135566], intercept=0.06048088608314441)
(weights=[-1.16574976784,1.20107836435], intercept=0.05847530056745036)
(weights=[-1.17276614919,1.20852225271], intercept=0.056555652624608634)
(weights=[-1.17961233387,1.21579096206], intercept=0.05471683977959186)
(weights=[-1.18629559406,1.22289193197], intercept=0.05295415009228922)
(weights=[-1.19282275085,1.22983214166], intercept=0.05126322551825725)
(weights=[-1.19920021082,1.23661814727], intercept=0.049640029332682994)
(weights=[-1.20543399886,1.24325611532], intercept=0.048080817099018854)
(weights=[-1.21152978795,1.24975185307], intercept=0.046582110738368133)
(weights=[-1.21749292595,1.25611083583], intercept=0.045140675318401845)
(weights=[-1.22332845999,1.26233823179], intercept=0.04375349823345739)
(weights=[-1.22904115855,1.26843892462], intercept=0.04241777049219722)
(weights=[-1.23463553158,1.27441753388], intercept=0.0411308698671671)
(weights=[-1.2401158488,1.28027843378], intercept=0.03989034569291044)
(weights=[-1.24548615648,1.28602577023], intercept=0.038693905126888424)
(weights=[-1.25075029268,1.2916634764], intercept=0.03753940071108389)
(weights=[-1.2559119013,1.29719528708], intercept=0.03642481909245313)
(weights=[-1.26097444489,1.30262475174], intercept=0.03534827077785603)
(weights=[-1.2659412165,1.30795524657], intercept=0.034307980814168124)
(weights=[-1.27081535051,1.31318998561], intercept=0.033302280297321296)
(weights=[-1.27559983264,1.31833203089], intercept=0.032329598625331434)
(weights=[-1.28029750915,1.32338430191], intercept=0.03138845642020391)
(weights=[-1.28491109542,1.32834958429], intercept=0.030477459052175304)
(weights=[-1.28944318378,1.33323053783], intercept=0.02959529070722 9)
```

```
Out[15]: <matplotlib.text.Text at 0x7f8324047150>
```



Based on accuracy scores we can see improvement stop after 90 iterations. so that could be good number of iterations with MLlib API. Also we can see there is little variation in weights after 90 iterations.

In [16]:
```
def logisticRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50
, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0])-1
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is
large
    else:
        w = wInitial

    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda d: (1 / (1 + np.exp(-d[0]*np.dot(wBroadcast.va
lue, d[1:])))-1)*(1/np.linalg.norm(np.array(d[1:]))) * d[0] * np.array(d[1:]))\
                     .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in re
gularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in re
gularization
            wReg = np.sign(wReg)
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg  #gradient:  GD of Sqaured Error+ G
D of regularized term
        w = w - learningRate * gradient / n



    return w
```

In [17]:
```
data = sc.textFile('HW11/HW113/data_logistic_train.txt').map(lambda line: [float(
v) for v in line.split(',')]+[1.0]).cache()
model = logisticRegressionGDReg(data,regType="Lasso")
```

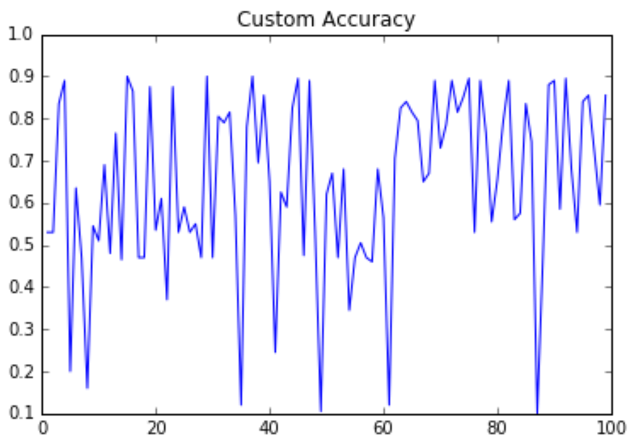
In [18]:
```
def predict(i):

    if i > 0:
        prob = 1 / (1 + exp(-i))
    else:
        exp_margin = exp(i)
        prob = exp_margin / (1 + exp_margin)
    if prob > 0.5:
         return 1
    else:
        return 0
```

In [19]:
```
def stop_here(prior, new):
    changes = np.array(prior) - np.array(new)
    return np.linalg.norm(changes)
```

In [20]:
```
from math import exp
data = sc.textFile('HW11/HW113/data_logistic_train.txt').map(lambda line: [float(
v) for v in line.split(',')]+[1.0]).cache()
test_data = sc.textFile('HW11/HW113/data_logistic_test.txt').map(lambda line: [fl
oat(v) for v in line.split(',')]).cache()

x_axis,y_axis =[],[]
last_model = []
for k in range(1,100):
    model = logisticRegressionGDReg(data,regType="Lasso",iterations=k)

    margin = np.array(test_data.collect())[:,1:3].dot(model[0:2]) + model[2:3]
    preds = np.array(map(lambda x: predict(x),margin))
    accuracy_score = np.mean(preds == y_test)
    x_axis.append(k)
    y_axis.append(accuracy_score)
    if len(last_model) > 0:
        if stop_here(last_model, model) < 0.02:

            break #Stoping criterian
        else:
            #print "Model ",model
            #print "Mode change Score %2.4f"%(stop_here(last_model, model))
            last_model = model

    last_model = model
print "Run stopped after %d Iterations"%(k  )
plt.plot(x_axis,y_axis)
plt.title("Custom Accuracy")
```

```
Run stopped after 99 Iterations
```

Out[20]:
```
<matplotlib.text.Text at 0x7f831ed9e990>
```



Response

HW11.4 SVMs

Use the non-linearly separable training and testing datasets from HW11.3 in this problem.

Using MLLib train up a soft SVM model with the training dataset and evaluate with the testing set. What is a good number of iterations for training the SVM model? Justify with plots and words.

HW11.4.1 [Optional] Derive and Implement in Spark a weighted hard linear svm classification learning algorithm. Feel free to use the following notebook as a starting point

> SVM Notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/dm2l73iznde7y4f/SVM-Notebook-Linear-Kernel-2015-06-19.ipynb).

Evaluate your homegrown weighted linear svm classification learning algorithm on the weighted training dataset and test dataset from HW11.3 (linearly separable dataset). Report misclassification error (1 - Accuracy) and how many iterations does it took to converge? How many support vectors do you end up with?

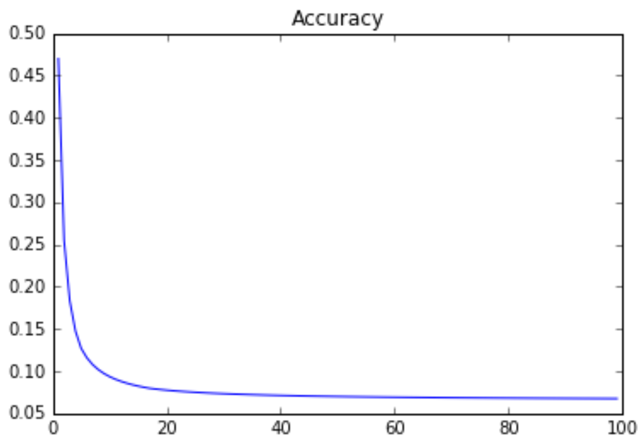
Does Spark MLLib have a weighted soft SVM learner. If so use it and report your findings on the weighted training set and test set.

HW11.4.2 [Optional] Repeat HW11.4.2 using a soft SVM and a nonlinearly separable datasets. Compare the error rates that you get here with the error rates you achieve using MLLib's soft SVM. Report the number of support vectors in both cases (may not be available the MLLib implementation).

In [23]:
```
from pyspark.mllib.classification import SVMModel, SVMWithSGD
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(point):
    label = 1 if point[-1] ==1 else 0
    return LabeledPoint(label, point[:-1])  #the input shoul be {0,1} instead of
{-1,1}
# Load and parse the data
def parsePoint_pred(point):

    return  point[:-1]
x_axis,y_axis =[],[]
correct, total = 0, 0
train_rdd = sc.parallelize(train_data).map(parsePoint).cache()
test_rdd = sc.parallelize(test_data.collect()).map(parsePoint_pred).cache()
last_model = []
for k in range(1,100):
# Build the model
    model_SVM = SVMWithSGD.train(train_rdd, regType="l1", intercept=True,iteratio
ns=k)
    print model_SVM
    #parsedtest = sc.textFile("HW11/HW113//data_logistic_test.txt").map(parsePoin
t_pred)
    Z = model_SVM.predict(test_rdd).collect()

    for pred, label in zip(Z,y_test):
            if pred == label: correct += 1
            total += 1
    if len(last_model) > 0:
        if stop_here(last_model, list(model_SVM.weights) )< 0.005 and 1.0*correct
/total > 80.00:

            break #Stoping criterian
        else:
            #print "Model ",model
            #print "Mode change Score %2.4f"%(stop_here(last_model, model))
            last_model = list(model_SVM.weights)

    last_model = list(model_SVM.weights)

    x_axis.append(k)
    y_axis.append(1.0*correct/total)
print "Run stopped after %d Iterations"%(k  )
plt.plot(x_axis,y_axis)
plt.title("Accuracy")
```

```
(weights=[-0.194909052426,0.201202350388], intercept=0.49)
(weights=[-0.46848367972,0.476193452946], intercept=0.48292893218813454)
(weights=[-0.629580330468,0.633936792616], intercept=0.38477938642589815)
(weights=[-0.758184092405,0.763154757913], intercept=0.2822793864258981)
(weights=[-0.856264981664,0.869138151421], intercept=0.18836453137090697)
(weights=[-0.923833779232,0.947901196897], intercept=0.1291685292536468)
(weights=[-0.963954770948,0.998447585817], intercept=0.10271101614300089)
(weights=[-0.996093081918,1.03564539532], intercept=0.08503334661333721)
(weights=[-1.01988639411,1.06319897022], intercept=0.07503334661333722)
(weights=[-1.03955030682,1.08400048081], intercept=0.07345220778325302)
(weights=[-1.05700028417,1.10226294294], intercept=0.0719446510603642)
(weights=[-1.07314842742,1.1178275358], intercept=0.0676145240414201)
(weights=[-1.08704172463,1.13183200349], intercept=0.06345427256975279)
(weights=[-1.10042964081,1.14532704623], intercept=0.059445353941066426)
(weights=[-1.11336359662,1.15836449613], intercept=0.05557237059485901)
(weights=[-1.12579461602,1.17005680159], intercept=0.05057237059485901)
(weights=[-1.13785447626,1.18140000403], intercept=0.045721658094132346)
(weights=[-1.1495537945,1.19139385564], intercept=0.04218612418819961)
(weights=[-1.15917893982,1.19980324339], intercept=0.04218612418819961)
(weights=[-1.16624078492,1.20665372992], intercept=0.037713988233200026)
(weights=[-1.17313244006,1.21333912022], intercept=0.03334963042848018)
(weights=[-1.17986564517,1.21987080284], intercept=0.02908561610136797)
(weights=[-1.18645084968,1.22625891447], intercept=0.024915327820226478)
(weights=[-1.19289740278,1.23251252447], intercept=0.020832844915587845)
(weights=[-1.19921370905,1.23863978589], intercept=0.01683284491558784)
(weights=[-1.20529210323,1.2439499989], intercept=0.01389110288851508)
(weights=[-1.21125687255,1.24916094684], intercept=0.011004351542566952)
(weights=[-1.21665317015,1.25397609404], intercept=0.0072247068124746796)
(weights=[-1.22195561193,1.25870749316], intercept=0.0035108000489336424)
(weights=[-1.22716893069,1.2633593673], intercept=0.0)
(weights=[-1.2321974351,1.26731796065], intercept=-0.0)
(weights=[-1.23714674551,1.27121421004], intercept=-0.0)
(weights=[-1.24191988539,1.2744638852], intercept=-0.0)
(weights=[-1.24662230817,1.27766541435], intercept=-0.0)
(weights=[-1.25125706666,1.28082087594], intercept=-0.0)
(weights=[-1.25582700017,1.28393220303], intercept=-0.0)
(weights=[-1.26033475481,1.28700119716], intercept=-0.0)
(weights=[-1.26425850091,1.28991066173], intercept=-0.0008111071056538125)
(weights=[-1.2681316159,1.29278258325], intercept=-0.0016117478746792478)
(weights=[-1.27195601052,1.29561837854], intercept=-0.0024023172897213426)
(weights=[-1.27536829635,1.29817570125], intercept=-0.0024023172897213426)
(weights=[-1.278739715,1.30070239623], intercept=-0.0024023172897213426)
(weights=[-1.2820717005,1.30319953821], intercept=-0.002402317289721343)
(weights=[-1.28536560491,1.30566814049], intercept=-0.0024023172897213426)
(weights=[-1.28862270477,1.30810915976], intercept=-0.0024023172897213426)
(weights=[-1.29184420684,1.31052350041], intercept=-0.0024023172897213426)
(weights=[-1.29462699211,1.31275407798], intercept=-0.00167299233223187)
(weights=[-1.29738063747,1.31496129808], intercept=-0.0009513044957448376)
(weights=[-1.30010603957,1.31714587942], intercept=-0.0002370187814591232)
(weights=[-1.30280404999,1.31930850461], intercept=-0.0)
(weights=[-1.30547547839,1.32144982263], intercept=-0.0)
(weights=[-1.30812109527,1.3235704511], intercept=-0.0)
(weights=[-1.31074163465,1.32567097838], intercept=-0.0)
(weights=[-1.3133377964,1.32775196544], intercept=-0.0)
(weights=[-1.31518625395,1.32951361945], intercept=0.0)
(weights=[-1.31701813307,1.33125947356], intercept=0.0)
(weights=[-1.31878046239,1.33254474118], intercept=0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
(weights=[-1.32016765486,1.33368198215], intercept=-0.0)
```

```
Out[23]: <matplotlib.text.Text at 0x7f831ed3ead0>
```



Response

HW11.5 [OPTIONAL] Distributed Perceptron algorithm.

Back to Table of Contents

Using the following papers as background: http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/36266.pdf (http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/36266.pdf)

https://www.dropbox.com/s/a5pdcp0r8ptudgj/gesmundo-tomeh-eacl-2012.pdf?dl=0 (https://www.dropbox.com/s/a5pdcp0r8ptudgj/gesmundo-tomeh-eacl-2012.pdf?dl=0)

http://www.slideshare.net/matsubaray/distributed-perceptron (http://www.slideshare.net/matsubaray/distributed-perceptron)

Implement each of the following flavors of perceptron learning algorithm:

1. Serial (All Data): This is the classifier returned if trained serially on all the available data. On a single computer for example (Mistake driven)
2. Serial (Sub Sampling): Shard the data, select one shard randomly and train serially.
3. Parallel (Parameter Mix): Learn a perceptron locally on each shard: Once learning is complete combine each learnt percepton using a uniform weighting
4. Parallel (Iterative Parameter Mix) as described in the above papers.

```
In [102]: # START STUDENT CODE 11.5
          # (ADD CELLS AS NEEDED)

          # END STUDENT CODE 11.5
Out[102]: 4000
```

HW11.6 [OPTIONAL: consider doing this in a group] Evalution of perceptron algorihtms on PennTreeBank POS corpus

Back to Table of Contents

Reproduce the experiments reported in the following paper:

Prediction with MapReduce - Andrea Gesmundo and Nadi Tomeh

http://www.aclweb.org/anthology/E12-2020 (http://www.aclweb.org/anthology/E12-2020)

These experiments focus on the prediction accuracy on a part-of-speech (POS) task using the PennTreeBank corpus. They use sections 0-18 of the Wall Street Journal for training, and sections 22-24 for testing.

In [ ]:
```
# START STUDENT CODE 11.6
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 11.6
```

HW11.7 [OPTIONAL: consider doing this in a group] Kernel Adatron

Implement the Kernal Adatron in Spark (contact Jimi for details)

In [ ]:
```
# START STUDENT CODE 11.7
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 11.7
```

HW11.8 [OPTIONAL] Create an animation of gradient descent for the Perceptron learning or for the logistic regression

Learning with the following 3 training examples. Present the progress in terms of the 2 dimensional input space in terms of a contour plot and also in terms of the 3D surface plot. See Live slides for an example.

Here is a sample training dataset that can be used:

```
-2, 3, +1
-1, -1, -1
2, -3, 1
```

Please feel free to use

- R (yes R!)
- d3
- https://plot.ly/python/ (https://plot.ly/python/)
- Matplotlib

I am happy for folks to collaborate on HW11.8 also.

It would be great to get the 3D surface and contours lines (with solution region and label normalized data) all in the same graph

In [ ]:
```
# START STUDENT CODE 11.8
# (ADD CELLS AS NEEDED)

# END STUDENT CODE 11.8
```

------- ALTERNATIVE HOWEWORK --------

- Implement a scaleable softmax classifier (aka multinomial logistic regression) in Spark (regularized and non-regularized)
- Run experiments on MNIST dataset and for CIFAR dataset
- And compare to MLLib implementations (accuracy, CPU times)

In [ ]:
```
```