MIDS-W261-HW-11-TEMPLATE

December 3, 2016

```
In [51]: %%javascript
         /********************************************************************************************
         Known Mathjax Issue with Chrome - a rounding issue adds a border to the right of mathjax marku
         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-with-a-trailing-
         ********************************************************************************************

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

```
<IPython.core.display.Javascript object>
```

```
In [52]: %reload_ext autoreload
         %autoreload 2
```

```
In [2]: import os
        import sys

        #Change SPARK_HOME to point to the folder where you installed Spark
        spark_home = os.environ['SPARK_HOME'] = '/usr/local/spark'

        if not spark_home:
            raise ValueError('SPARK_HOME enviroment variable is not set')
        sys.path.insert(0,os.path.join(spark_home,'python'))
        sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.10.3-src.zip'))
        execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Python version 2.7.11 (default, Dec  6 2015 18:57:58)
SparkSession available as 'spark'.
```

1 MIDS - w261 Machine Learning At Scale

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

1.1 Assignment - HW11

Name: Anthony Spalvieri-Kruse **Class:** MIDS w261 (Section Fall 2016 Group 1)
Email: ask@iSchool.Berkeley.edu
Week: 11

2 Table of Contents

# 1 Instructions Back to Table of Contents
MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #11
Version 2016-07-27 (FINAL)
=== INSTRUCTIONS for SUBMISSIONS === Follow the instructions for submissions carefully.
https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form
=== IMPORTANT ===
TYPE-2 Fun option: Submit HW11 using a Zeppelin notebook (See Live slides for install instructions)
TYPE-1.5 Fun option: Complete HW11.8 only (no need to complete the rest of the questions)
HW11 can be completed locally on your computer ### Documents: * IPython Notebook, published and viewable online. * PDF export of IPython Notebook.
# 2 Useful References Back to Table of Contents

- 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)

# HW Problems Back to Table of Contents
HW11.0: Broadcast versus Caching in Spark
Back to Table of Contents
HW11.0
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.**

```
When you broadcast data, you're sending an immutable copy of that variable to each node rather than shi
```

```
In [ ]: def logisticRegressionGD(data, wInitial=None, learningRate=0.05, iterations=100):
            featureLen = len(data.take(1)[0].x)
            n = data.count()
            if wInitial is None:
                w = np.random.normal(size=featureLen)
            else:
                w = wInitial
            for i in range(iterations):
                wBroadcast = sc.broadcast(w)
                gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y*np.dot(wBroadcast.value, p.x)))-1) 
                            .reduce(lambda a, b: a + b)
                w = w - learningRate * gradient / n
            #w = w / np.linalg.norm(w) #normalization
            return w
```

In the above example we see that the weight vector is being broadcast to the regression job so that it

When you cache an RDD, this stores it in memory on each node for iterative use without having to retrie

```
In [ ]: data = sc.textFile('data.csv').map(readPoint).cache()
        logisticRegressionGD(data)
```

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:**

Notebook 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

The kmeans code in the notebook iteratively computes a centroids vector that is being shipped to every

```
In [ ]: import numpy as np

        #Calculate which class each data point belongs to
        def nearest_centroid(line):
            x = np.array([float(f) for f in line.split(',')])

            # LOOK HERE #
            #Changing the global variable 'centroids' to 'broadcastCentroids'
            closest_centroid_idx = np.sum((x - broadcastCentroids)**2, axis=1).argmin()
            return (closest_centroid_idx,(x,1))

        #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()

        K = 3
        # 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]])
```

```
D = sc.textFile("./data.csv").cache()
iter_num = 0
for i in range(10):

    # LOOK HERE #
    #Broadcasting the centroids to reduce overhead
    broadcastCentroids = sc.broadcast(centroids)
    # DID I MENTION THAT THIS IS MY CHANGE BECAUSE IT IS PLEASE ACKNOWLEDGE #

    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 cluster size
    if np.sum(np.absolute(centroids_new-centroids))<0.01:
        break
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    centroids = centroids_new
    print centroids
    plot_iteration(centroids)
print "Final Results:"
print centroids
```

HW11.1 Loss Functions

Back to Table of Contents

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.

```
Both logistic regression and linear svm find a separating hyperplane, but the approach to doing this is

SVM Loss: Li=∑(j≠yi)max(0,wTj·xi − wT·yi·xi+Δ)

Logistic Regression Loss: L = 1/N * ∑ ln(1 + e^(−yn·wt·xn)
```

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

```
SVM algorithm and perceptron also have similar loss functions that find separating hyper planes, but the
```

HW11.2 Gradient descent

Back to Table of Contents

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)?

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

```
All of the referenced loss penalties are available in mllib.  Probabalistically, we can interpret the
```

HW11.3 Logistic Regression

Back to Table of Contents

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.

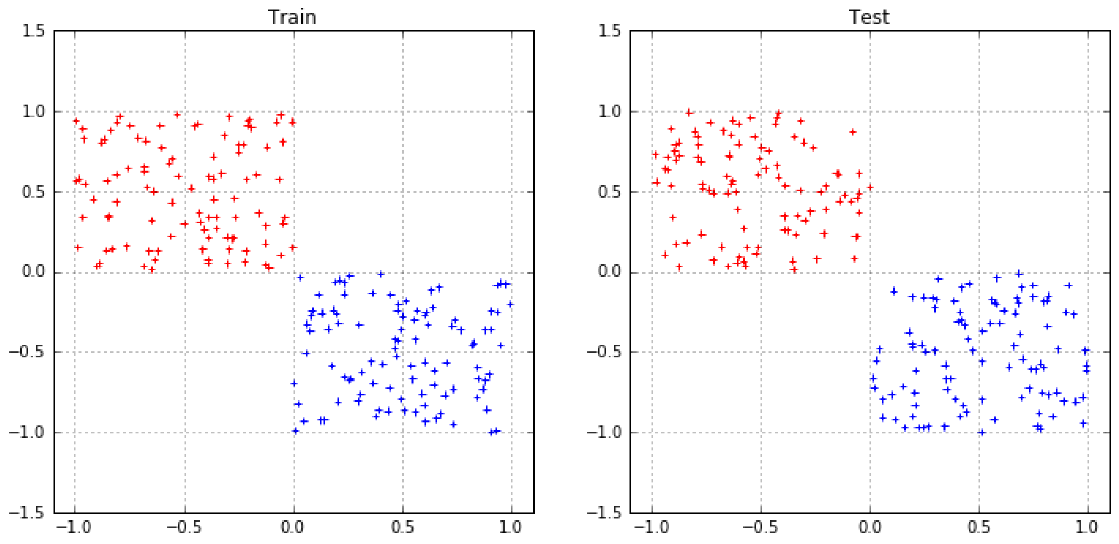
```
def generateData(n):
 """
  generates a 2D linearly separable dataset with n samples.
  The third element of the sample is the label
 """
 xb = (rand(n)*2-1)/2-0.5
 yb = (rand(n)*2-1)/2+0.5
 xr = (rand(n)*2-1)/2+0.5
 yr = (rand(n)*2-1)/2-0.5
 inputs = []
 for i in range(len(xb)):
  inputs.append([xb[i],yb[i],1])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

```
In [49]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt
         import random

         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 # rand(n) - 1/2 +/- 1/2
             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

         train = generateData(100)
         test = generateData(100)

         # plot with various axes scales
         plt.figure(figsize=[12,12])

         colors = [{1:'r',-1:'b'}[i[2]] for i in train]
         plt.subplot(221)
         plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
         plt.grid(True)
         plt.ylim(-1.5,1.5)
         plt.xlim(-1.1,1.1)
         plt.title("Train")


         colors = [{1:'r',-1:'b'}[i[2]] for i in test]
```

```
plt.subplot(222)
plt.scatter([i[0] for i in test], [i[1] for i in test], marker='+', c=colors)
plt.grid(True)
plt.ylim(-1.5,1.5)
plt.xlim(-1.1,1.1)
plt.title("Test")
```

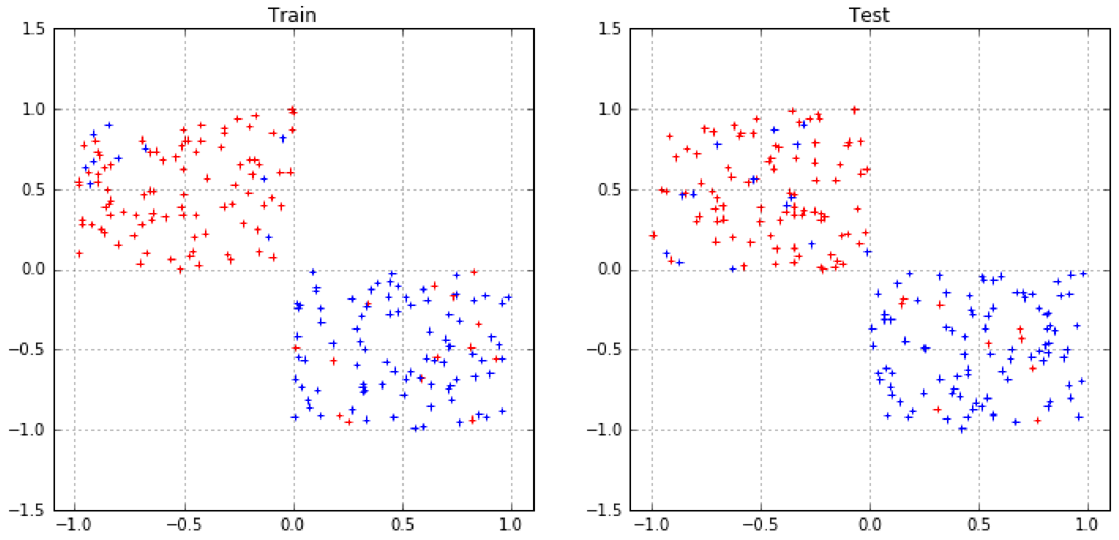
```
Out[49]: <matplotlib.text.Text at 0x1125f7f90>
```



```
In [30]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt
         import random
         from numpy.random import choice

         def generateMixedData(n, mixProp):
             """
             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 # rand(n) - 1/2 +/- 1/2
             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)):
                 draw1 = choice([1,0], 1, p=[1-mixProp, mixProp])[0]
                 draw2 = choice([0,1], 1, p=[1-mixProp, mixProp])[0]
                 inputs.append([xb[i],yb[i],draw1])
                 inputs.append([xr[i],yr[i],draw2])
             return inputs

         train = generateMixedData(100, .1)
         test = generateMixedData(100, .1)
```

```
# plot with various axes scales
plt.figure(figsize=[12,12])

colors = [{1:'r',0:'b'}[i[2]] for i in train]
plt.subplot(221)
plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
plt.grid(True)
plt.ylim(-1.5,1.5)
plt.xlim(-1.1,1.1)
plt.title("Train")


colors = [{1:'r',0:'b'}[i[2]] for i in test]
plt.subplot(222)
plt.scatter([i[0] for i in test], [i[1] for i in test], marker='+', c=colors)
plt.grid(True)
plt.ylim(-1.5,1.5)
plt.xlim(-1.1,1.1)
plt.title("Test")
```

```
Out[30]: <matplotlib.text.Text at 0x10e082890>
```



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.

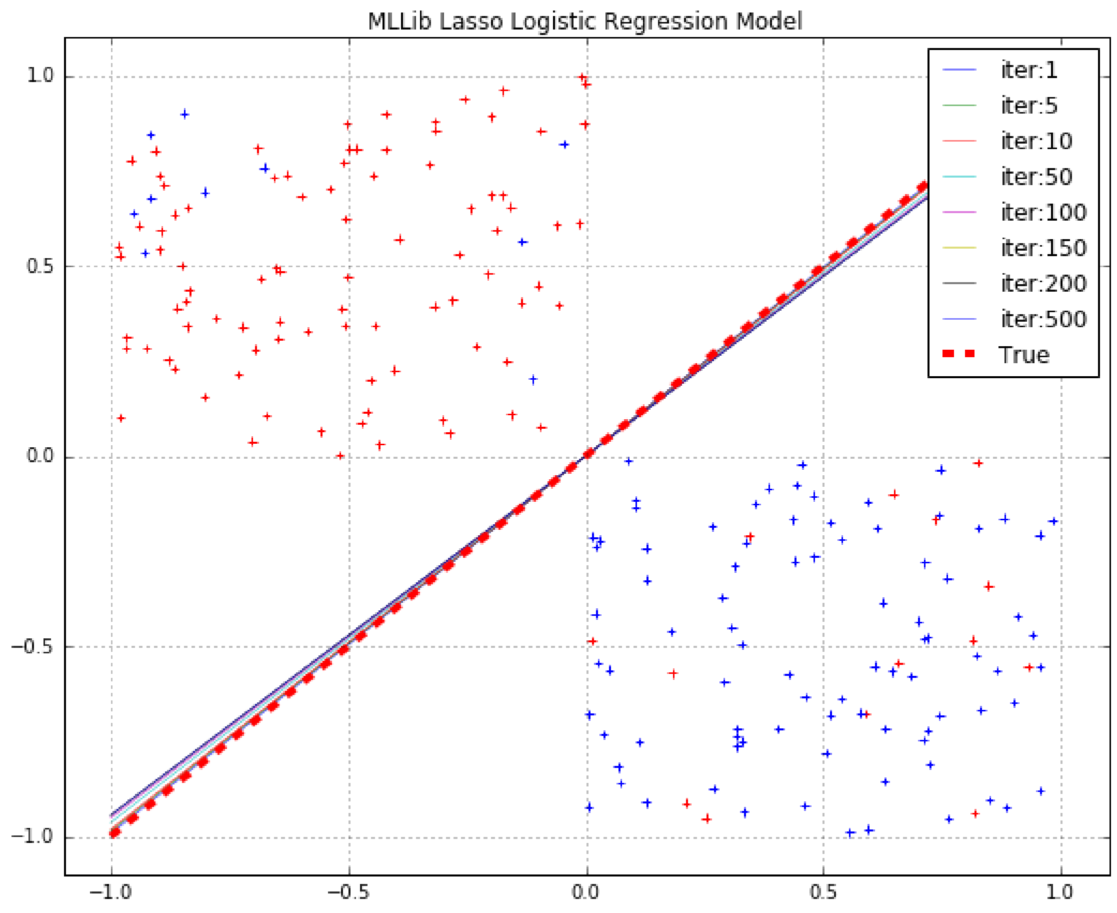
```
At around 150 iterations we see that the weight and intercept values stop varying, and we see that the
```

```
In [78]: from pyspark.mllib.regression import LabeledPoint
         from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel,
         import numpy as np
         iterations = [1,5,10,50,100,150, 200,500]
```

```
# Load and parse the data
def parsePoint(line):
    return LabeledPoint(line[2], line[0:2])

train = np.array(train)
test = np.array(test)
trainData = spark.sparkContext.parallelize(train).map(parsePoint).cache()
testData = spark.sparkContext.parallelize(test).map(parsePoint).cache()
plt.figure(figsize=(10, 8))
colors = [{1:'r',0:'b'}[i[2]] for i in train]
plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
plt.grid(True)
plt.ylim(-1.1,1.1)
plt.xlim(-1.1,1.1)
print "{0: <10} | {1: <35} | {2: <10} | {3}".format("Iteration", "Weights", "Intercept", "Mean
for iteration in iterations:
    model = LogisticRegressionWithSGD.train(trainData, iterations=iteration, regType='l1', ini
    # Evaluate the model on training data
    valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))
    MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAnd
    x2 = [- i * model.weights[0] / model.weights[1] for i in [-1,1]]
    plt.plot([-1,1], x2, label='iter:' + str(iteration), linewidth=0.5)
    accuracy = valuesAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())
    print "{0: <10} | {1: <35} | {2: <10} | {3}".format(iteration, model.weights, model.interc
plt.plot([-1, 1], [-1, 1], 'r--', label='True', linewidth=4.0)
plt.title("MLLib Lasso Logistic Regression Model")
plt.legend()
```

```
Iteration  | Weights                             | Intercept  | Mean Squared Error
1          | [-0.174434084155,0.175692994186]    | 0.0        | 0.115
5          | [-0.467130127517,0.473554748155]    | 0.0        | 0.115
10         | [-0.639061974904,0.650955194959]    | 0.0        | 0.115
50         | [-1.06182401207,1.10165771661]      | 0.0        | 0.115
100        | [-1.21511987396,1.27653477398]      | 0.0        | 0.115
150        | [-1.27969568109,1.35509810109]      | 0.0        | 0.115
200        | [-1.27969568109,1.35509810109]      | 0.0        | 0.115
500        | [-1.27969568109,1.35509810109]      | 0.0        | 0.115
```

```
Out[78]: <matplotlib.legend.Legend at 0x110e8b050>
```



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(X1ˆ2 + X2ˆ2)

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.

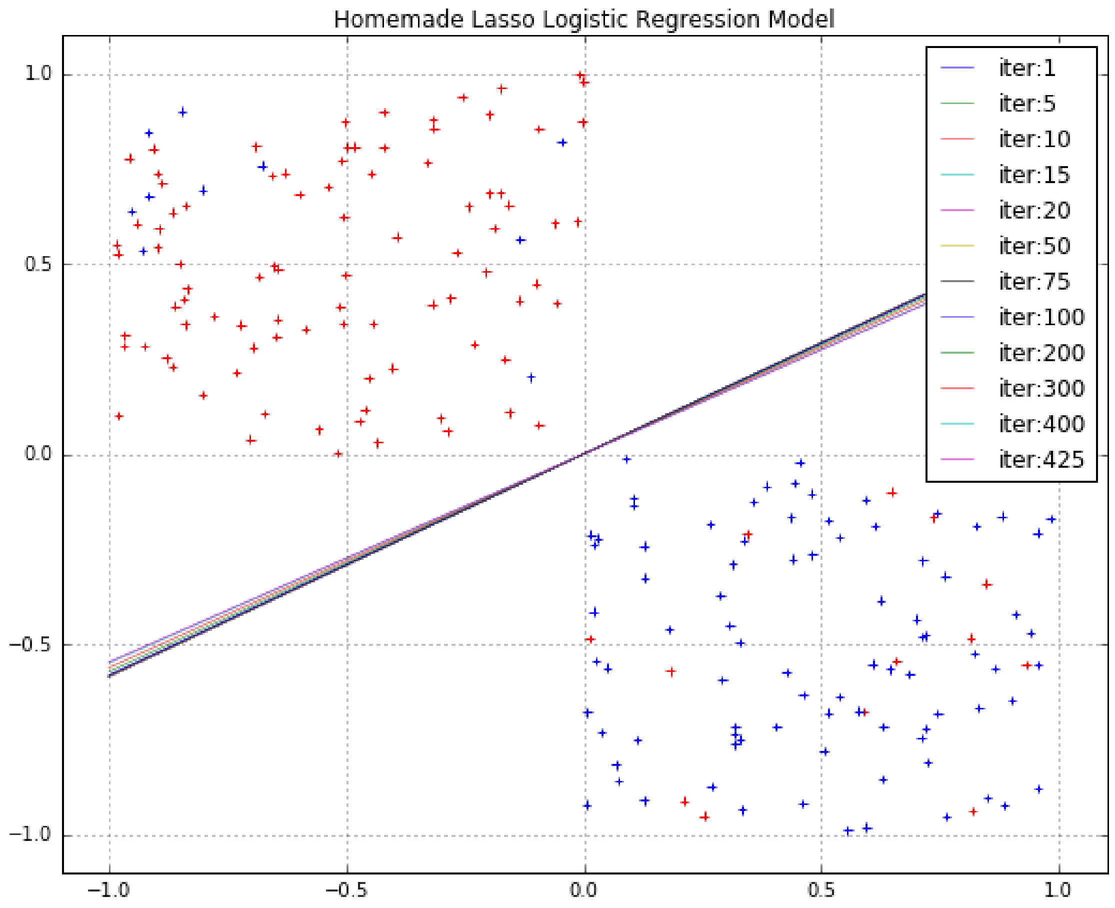
Does Spark MLLib have a weighted LASSO logistic regression implementation. If so use it and report your findings on the weighted training set and test set.

```
In [131]: #Riffing off of notebook from class: http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/r20ff7
          def logisticRegressionGDReg(data, test=None, wInitial=None, learningRate=0.05, iterations=50,
              featureLen = len(data.take(1)[0].features)
              n = data.count()
              delta = 0
              converged = False

              if test!= None:
```

```
        predicted_total = test.count()

    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial

    plt.figure(figsize=(10, 8))
    colors = [{1:'r',0:'b'}[i[2]] for i in train]
    plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
    plt.grid(True)
    plt.ylim(-1.1,1.1)
    plt.xlim(-1.1,1.1)
    print "{0: <10} | {1: <25} | {2: <20} | {3}".format("Iteration", "Weights", "Weight Diff"

    for i in range(iterations):
        wBroadcast = sc.broadcast(w)

        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.label*np.dot(wBroadcast.value, p.fea
                       .reduce(lambda x, y: x + y)/n

        if regType == "Ridge":
            wReg = wBroadcast.value * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = wBroadcast.value * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(wBroadcast.value.shape[0])

        wdelta = learningRate * (gradient + regParam * wReg)
        gradient = gradient + regParam * wReg

        # Stop condition when weight change is less than threshold
        if np.sum(np.abs(wdelta)) <= tolerance:
            break
        else:
            delta = np.sum(np.abs(wdelta)) - tolerance*np.sum(np.abs(wBroadcast.value))

        w = wBroadcast.value - wdelta

        if test != None:
            predicted_correct = test.map(lambda p: np.dot(w, p.features) - p.label).filter(la
            accuracy = predicted_correct / float(predicted_total)
            errors = 1 - accuracy
        if i in [1, 5, 10, 15, 20,50, 75, 100, 200, 300, 400, 500, 1000, 1500]:
            print "{0: <10} | {1: <25} | {2: <20} | {3}".format(i, w, delta, errors)
            x2 = [- j * w[0] / w[1] for j in [-1,1]]
            plt.plot([-1,1], x2, label='iter:' + str(i), linewidth=0.5)

    print "{0: <10} | {1: <25} | {2: <20} | {3}".format(i, w, delta, errors)
    x2 = [- j * w[0] / w[1] for j in [-1,1]]
    plt.plot([-1,1], x2, label='iter:' + str(i), linewidth=0.5)
```

```
        plt.legend()
        plt.title("Homemade Lasso Logistic Regression Model")
        print "Num Iterations: ", i
        print "End Weights: ",  w

    logisticRegressionGDReg(trainData, testData, regType="Lasso", iterations=1500, tolerance=0.00
```

```
Iteration   | Weights                     | Weight Diff           | Misclassification Error
1           | [-1.02979104  1.75745246] | -0.000370770797763    | 0.365
5           | [-1.03324144  1.76360022] | -0.000403583264725    | 0.365
10          | [-1.0374895   1.77122062] | -0.000444042716905    | 0.365
15          | [-1.04166655  1.77877062] | -0.000483896092793    | 0.365
20          | [-1.04577378  1.78625131] | -0.000523155765455    | 0.365
50          | [-1.06901415  1.82973906] | -0.000746898982279    | 0.37
75          | [-1.08665989  1.86425826] | -0.000919058771277    | 0.38
100         | [-1.10286965  1.89733312] | -0.00107950716833     | 0.38
200         | [-1.15537109  2.01707184] | -0.00162354660625     | 0.39
300         | [-1.19198725  2.12030338] | -0.00204627337785     | 0.4
400         | [-1.21659574  2.21064194] | -0.00238089865561     | 0.405
425         | [-1.22104288  2.23071072] | -0.00245074323906     | 0.405
Num Iterations:  425
End Weights:  [-1.22104288  2.23071072]
```



2.0.1 I was unable to find an MLLib weighted Lasso logistic regression model to use out of the box.

HW11.4 SVMs

Back to Table of Contents

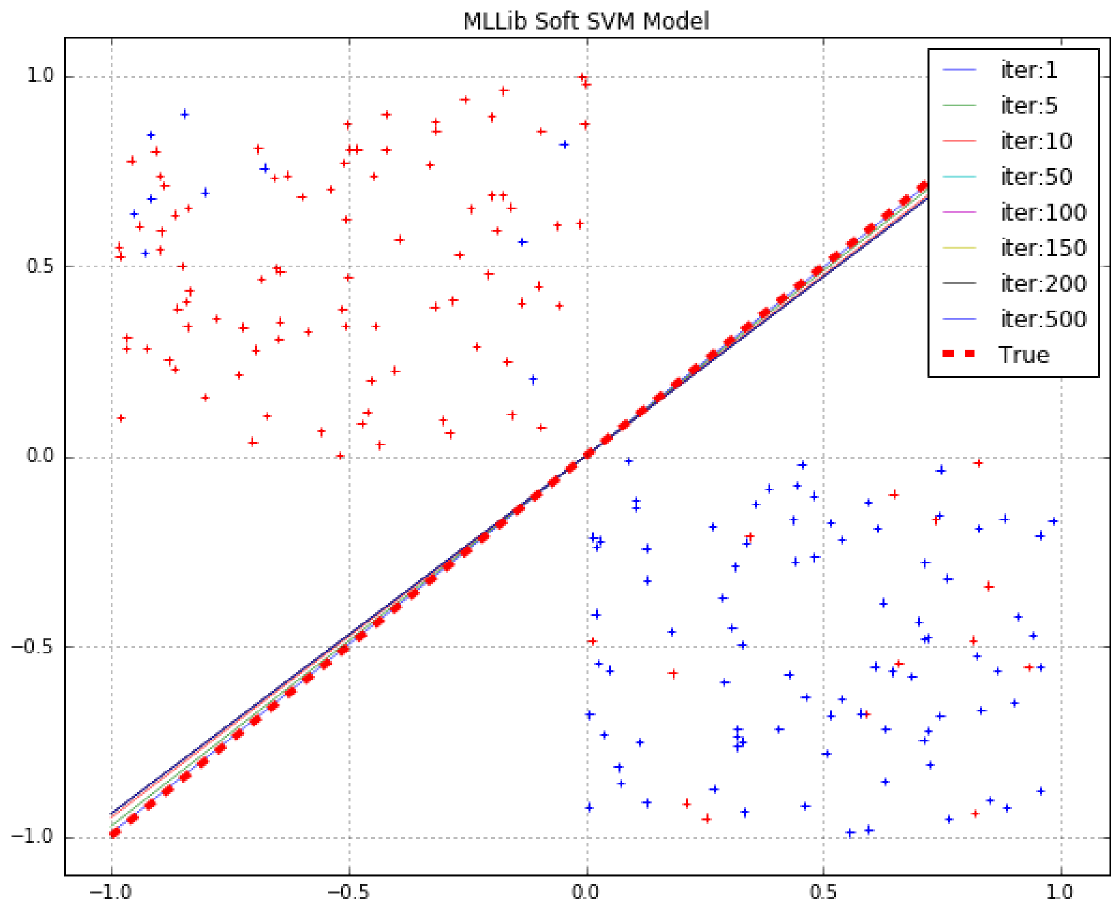
Use the non-linearly separable training and testing datasets from HW11.3 in this problem.

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.

```
In [132]: from pyspark.mllib.regression import LabeledPoint
          from pyspark.mllib.classification import SVMWithSGD, SVMModel
          import numpy as np
          iterations = [1,5,10,50,100,150, 200,500]

          # Load and parse the data
          def parsePoint(line):
              return LabeledPoint(line[2], line[0:2])

          plt.figure(figsize=(10, 8))
          colors = [{1:'r',0:'b'}[i[2]] for i in train]
          plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
          plt.grid(True)
          plt.ylim(-1.1,1.1)
          plt.xlim(-1.1,1.1)
          print "{0: <10} | {1: <35} | {2: <10} | {3}".format("Iteration", "Weights", "Intercept", "Mea
          for iteration in iterations:
              model = SVMWithSGD.train(trainData, iterations=iteration, initialWeights=[0,0], regParam=
              # Evaluate the model on training data
              valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))
              MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAn
              x2 = [- i * model.weights[0] / model.weights[1] for i in [-1,1]]
              plt.plot([-1,1], x2, label='iter:' + str(iteration), linewidth=0.5)
              accuracy = valuesAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())
              print "{0: <10} | {1: <35} | {2: <10} | {3}".format(iteration, model.weights, model.inter
          plt.plot([-1, 1], [-1, 1], 'r--', label='True', linewidth=4.0)
          plt.title("MLLib Soft SVM Model")
          plt.legend()
```

```
Iteration  | Weights                             | Intercept  | Mean Squared Error
1          | [-0.368868168309,0.371385988372]    | 0.0        | 0.115
5          | [-0.940817939735,0.965766461144]    | 0.0        | 0.115
10         | [-1.03120826787,1.0830381422]       | 0.0        | 0.115
50         | [-1.10549532686,1.17378773468]      | 0.0        | 0.115
100        | [-1.10549532686,1.17378773468]      | 0.0        | 0.115
150        | [-1.10549532686,1.17378773468]      | 0.0        | 0.115
200        | [-1.10549532686,1.17378773468]      | 0.0        | 0.115
500        | [-1.10549532686,1.17378773468]      | 0.0        | 0.115
```

```
Out[132]: <matplotlib.legend.Legend at 0x111939e90>
```



The MLLib SVM model converges very quickly; at around 50 iterations the model weights cease to change appreciably. Not to mention, the initial separating line is quite close to actual, and the variation between each iteration is quite minor visually. **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.

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.

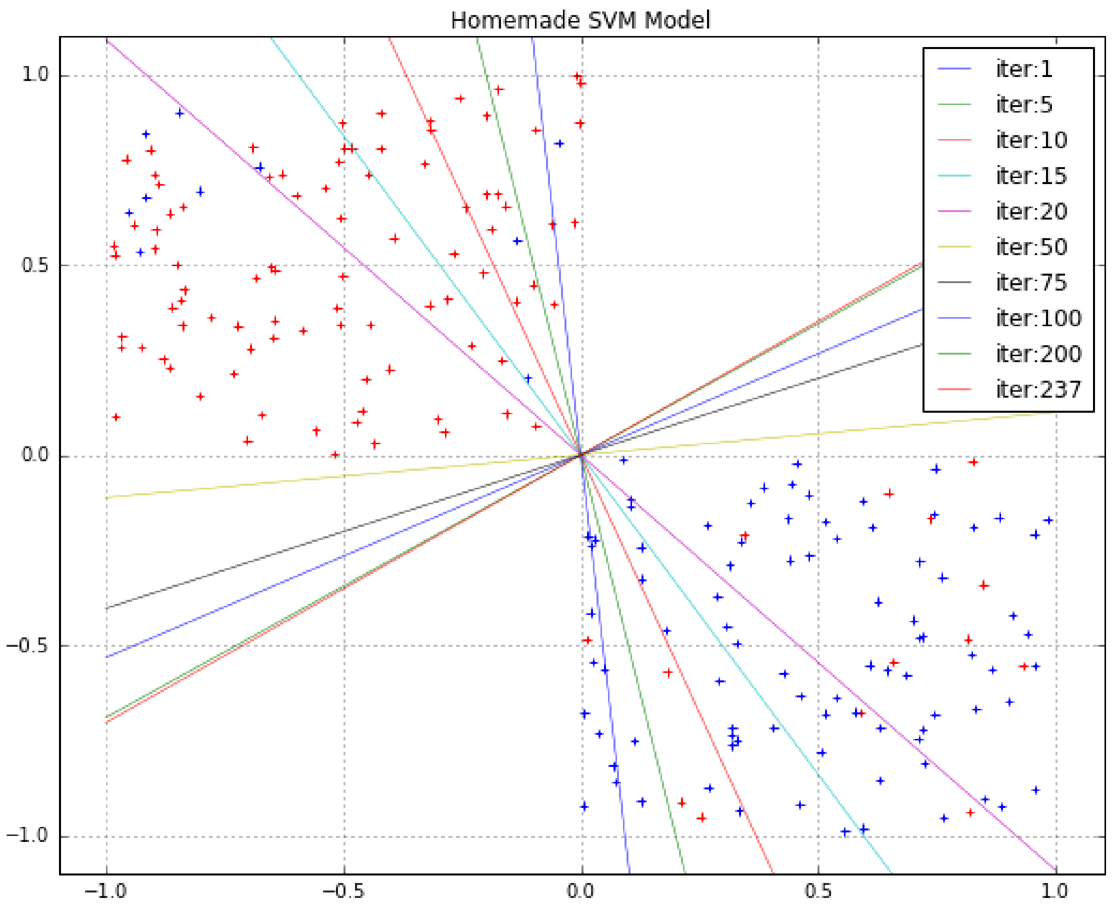
HW11.4.2 [Optional] Repeat HW11.4.2 using a soft SVM and a nonlinearly separable datasets. Compare the error rates that you get here with the error rates you achieve using MLLib's soft SVM. Report the number of support vectors in both cases (may not be available the MLLib implementation).

```
In [155]: def SVM_GD_SPARK(data, test=None,w=None,eta=0.05,iter_num=1000,regPara=0.01,stopCriteria=0.00
              #eta learning rate
              #regPara
              converged = False

              if test!= None:
                  predicted_total = test.count()

              plt.figure(figsize=(10, 8))
              colors = [{1:'r',0:'b'}[i[2]] for i in train]
              plt.scatter([i[0] for i in train], [i[1] for i in train], marker='+', c=colors)
              plt.grid(True)
              plt.ylim(-1.1,1.1)
              plt.xlim(-1.1,1.1)
              print "{0: <10} | {1: <25} | {2: <20} | {3}".format("Iteration", "Weights", "Weight Diff"

              if w==None:
                  w = np.random.normal(size=len(data.take(1)[0].features))
              for i in range(iter_num):       #label * margin
                  wBroadcast = spark.sparkContext.broadcast(w)
                  sv = data.filter(lambda p: p.label * np.dot(wBroadcast.value, p.features)<1)

                  if sv.isEmpty(): # converged as no more updates possible
                      break       # hinge loss compoent of gradient y*x and sum up
                  g = -1*sv.map(lambda x : x.label*x.features*np.reciprocal(np.sqrt(np.sum(np.square(x.
                  wreg = wBroadcast.value*1   #temp copy of weight vector
                  wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
                  wdelta = eta*(g+regPara*wreg)  #gradient: hinge loss + regularized term

                  if sum(abs(np.array(wdelta)))<=stopCriteria*sum(abs(wBroadcast.value)): # converged a
                      break
                  else:
                      delta = np.sum(np.abs(wdelta)) - stopCriteria*np.sum(np.abs(wBroadcast.value))
                  w = w - wdelta

                  if test != None:
                      predicted_correct = test.map(lambda p: np.dot(w, p.features) - p.label).filter(lam
                      accuracy = predicted_correct / float(predicted_total)
                      errors = 1 - accuracy
                  if i in [1, 5, 10, 15, 20,50, 75, 100, 200, 300, 400, 500, 1000, 1500]:
                      print "{0: <10} | {1: <25} | {2: <20} | {3}".format(i, w, delta, errors)
                      x2 = [- j * w[0] / w[1] for j in [-1,1]]
                      plt.plot([-1,1], x2, label='iter:' + str(i), linewidth=0.5)

              print "{0: <10} | {1: <25} | {2: <20} | {3}".format(i, w, delta, errors)
              x2 = [- j * w[0] / w[1] for j in [-1,1]]
              plt.plot([-1,1], x2, label='iter:' + str(i), linewidth=0.5)
              plt.legend()
              plt.title("Homemade SVM Model")
              print "Num Iterations: ", i
              print "End Weights: ",  w
              print "Num Support Vectors: ", sv.count()

              return w

          SVM_GD_SPARK(trainData, testData)
```

```
Iteration  | Weights                   | Weight Diff          | Misclassification Error
1          | [ 0.54955959  0.0510473 ] | 0.0243319804778      | 0.43
5          | [ 0.49825225  0.09940914] | 0.0243092851442      | 0.405
```

```
10          | [ 0.43426225  0.15986143] | 0.0242808438564      | 0.375
15          | [ 0.37043206  0.22031373] | 0.024252322621       | 0.33
20          | [ 0.30676128  0.28076603] | 0.0242237216377      | 0.295
50          | [-0.07193785  0.6434798 ] | 0.0239316403947      | 0.07
75          | [-0.38101863  0.94371616] | 0.0214151074722      | 0.13
100         | [-0.60086661  1.12718838] | 0.0113313260635      | 0.21
200         | [-0.97702249  1.41420235] | 0.000468116221091    | 0.32
237         | [-1.02737231  1.45890911] | 2.02730045571e-07    | 0.325
Num Iterations:  237
End Weights:  [-1.02737231  1.45890911]
Num Support Vectors:  134
```

```
Out[155]: array([-1.02737231,  1.45890911])
```



The misclassification error is all over the place from iteration to iteration, but it's cycling between .07 and .43. Final error was .325 and it took 237 iterations to converge, with 134 support vectors at the end (really? This seems wrong, that's too many support vectors). As in the previous question i couldn't find an MLLib weighted model for SVM.

```
In [ ]:
```