READY

General Housekeeping

Submission Details

READY

- **Student**: Shih Yu Chang
- **Email Address**: sychang@berkeley.edu
- **Course**: 2016-0111 DATASCI W261: Machine Learning at Scale
- **Section**: Fall 2016, Section 2
- **Assignment**: Homework 11, Week 11
- **Submission Date**: Nov. 29, 2016

MathJax for LaTeX Notation

READY

```
%angular

<script type="text/javascript" async src="https://cdn.mathjax.org/mathjax/latest/Mat
<script type="text/x-mathjax-config">MathJax.Hub.Config({tex2jax: {inlineMath: [['$'
```

Pyspark Plots without SQL

FINISHED

```
%spark.pyspark

# Code adapted from the following places:
# https://github.com/bernhard-42/Zeppelin-Visualizations/blob/master/Code.md
# http://stackoverflow.com/questions/5314707/matplot-store-image-in-variable

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
import StringIO
import urllib, base64

def showMPL():
    img = StringIO.StringIO()
    plt.savefig(img, format='png', dpi=72)
    img.seek(0)
```

Zeppelin Notebook

MIDS-W261-2016-

Took 1 sec. Last updated by anonymous at November 11 2016, 10:34:38 AM.

anonymous

READER default

HW11.0 Broadcast Versus Caching In Spark

What is the difference between broadcasting and caching data in Spark?

FINISHED

A broadcasting makes sure that all partitions receive a materialized value in memory without having to bundle this value in a closure, and this value is shared by all partitions on the same node. A caching represents elements that have been materialized from an RDD on some partition, but the cache is subject to a least-recently-used (LRU) eviction policy.

Took 0 sec. Last updated by anonymous at November 11 2016, 10:30:08 AM. (outdated)

Give an example (in the context of machine learning) of each mechanism (at a high level).

FINISHED

One example in the context of machine learning would be the PageRank algorithm. In PageRank, you would want all partitions to recognize the total number of nodes in the graph and any mass associated with dangling nodes so that they can adopt the teleportation factor. In this case, you would want to broadcast the node count and the dangling node mass to all partitions. At the same time, you would want to avoid re-materializing the web graph in every iteration because the entire web graph would not fit in memory. Instead, one would opt for each partition by storing as much as possible (usually the specific values it materialized) through caching the data.

Took 0 sec. Last updated by anonymous at November 11 2016, 10:37:51 AM. (outdated)

Review the following Spark-notebook-based implementation of k-means. (Part 1)

FINISHED

```
%spark.pyspark

# Start with the sample code
# http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb

# Create the randomized data.
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

Took 0 sec. Last updated by anonymous at November 11 2016, 10:54:46 AM.

Perform data exploration of the generated k-means data.

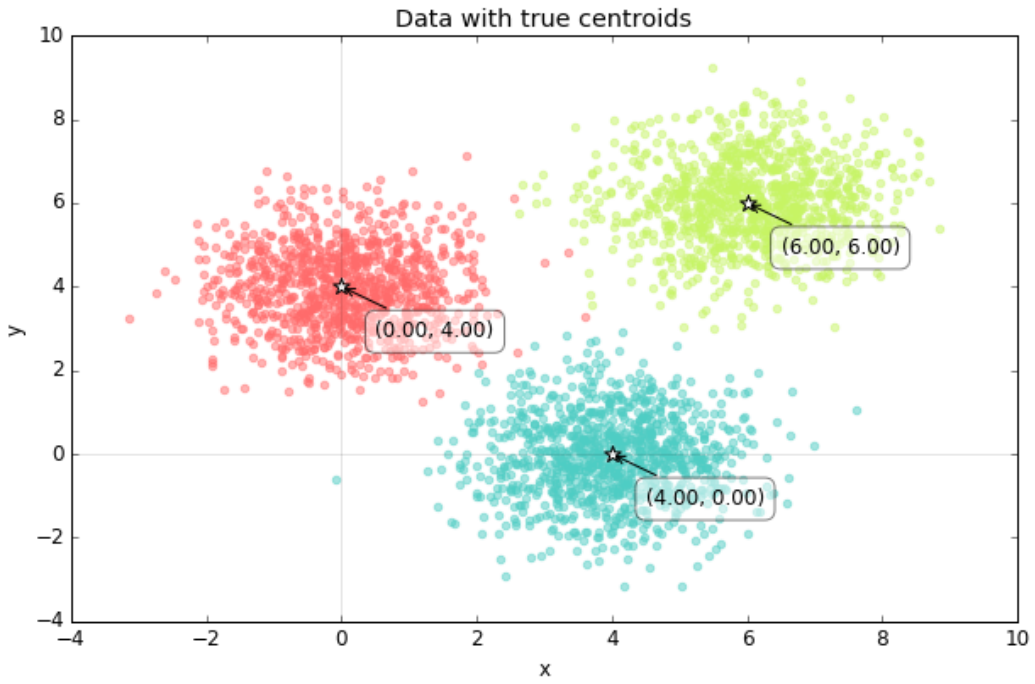
FINISHED

```
%spark.pyspark

def plot_samples(title):
    plt.figure(figsize=(10,10/3*2))

    # Plot axis
    plt.axhline(0, color='black', alpha=0.1)
    plt.axvline(0, color='black', alpha=0.1)

    # Plot data points
    plt.scatter(samples1[:, 0], samples1[:, 1], color='#4ECDC4', alpha=0.5)
    plt.scatter(samples2[:, 0], samples2[:, 1], color='#C7F464', alpha=0.5)
    plt.scatter(samples3[:, 0], samples3[:, 1], color='#FF6B6B', alpha=0.5)

    # Label graph
    plt.title(title)
    plt.xlabel('x')
    plt.ylabel('y')

def plot_iteration(before, after):
    centroids_path = np.array([before, after])

    # Plot the transition from one centroid to another as a path
    for i in range(3):
        plt.plot(
            centroids_path[:,i,0], centroids_path[:,i,1], color='grey', alpha=0.9, m
            markerfacecolor='white', markeredgecolor='black', zorder=2)

def plot_centroids(means):
    # Plot centroids
    plt.scatter(means[0][0], means[0][1], marker='*', s=100, color='white', edgecolo
    plt.scatter(means[1][0], means[1][1], marker='*', s=100, color='white', edgecolo
    plt.scatter(means[2][0], means[2][1], marker='*', s=100, color='white', edgecolo

    # Annotate centroids
    for i in means:
        text = '({:.2f}'.format(i[0]) + ', ' + '{:.2f})'.format(i[1])
        plt.annotate(text,
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 1 sec. Last updated by anonymous at November 11 2016, 10:54:52 AM.

Make the created k-means data available in HDFS.

FINISHED

```
%sh

hdfs dfs -rm -f -skipTrash data.csv > /dev/null
hdfs dfs -copyFromLocal data.csv
```

Took 15 sec. Last updated by anonymous at November 11 2016, 11:00:08 AM. (outdated)

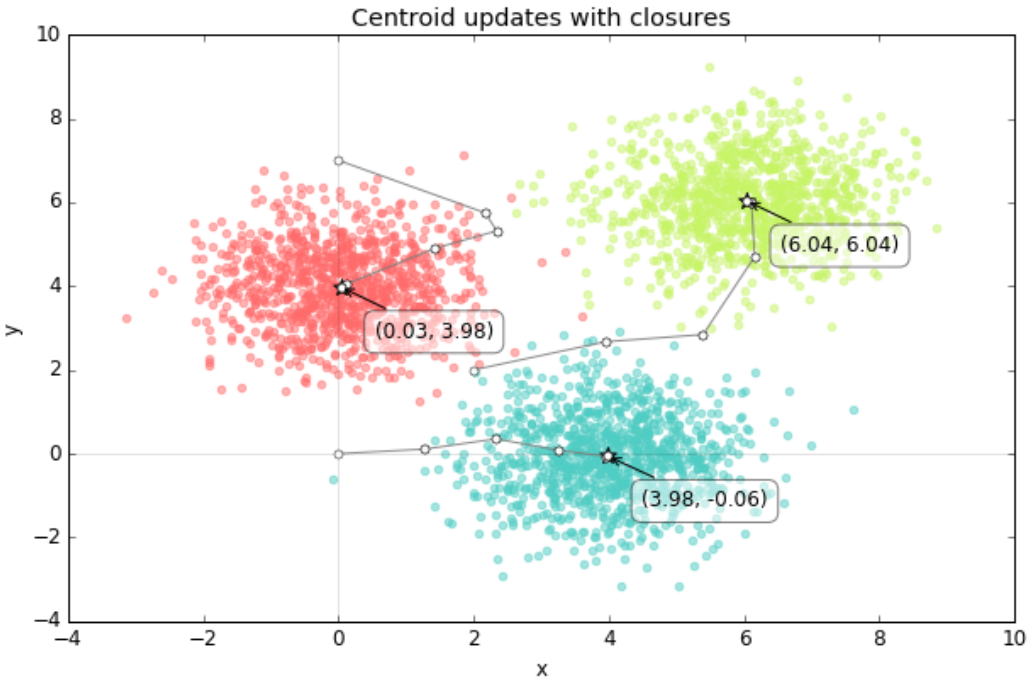
FINISHED

```
%spark.pyspark

# Calculate which class each data point belongs to

def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])
    closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin()
    return (closest_centroid_idx,(x,1))
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 9 sec. Last updated by anonymous at November 11 2016, 11:04:30 AM.

Use the broadcast pattern to make this implementation more efficient. Please describe your changes in English first.

FINISHED

```
%md
```

Zeppelin Notebook

First, the centroids are transmitted to each partition through a closure, then we broadcast the centroids so that the centroids are transmitted once per node rather than once per partition.

MIDS-W261-2016-... anonymous

Took 0 sec. Last updated by anonymous at November 11 2016, 11:08:01 AM.

default

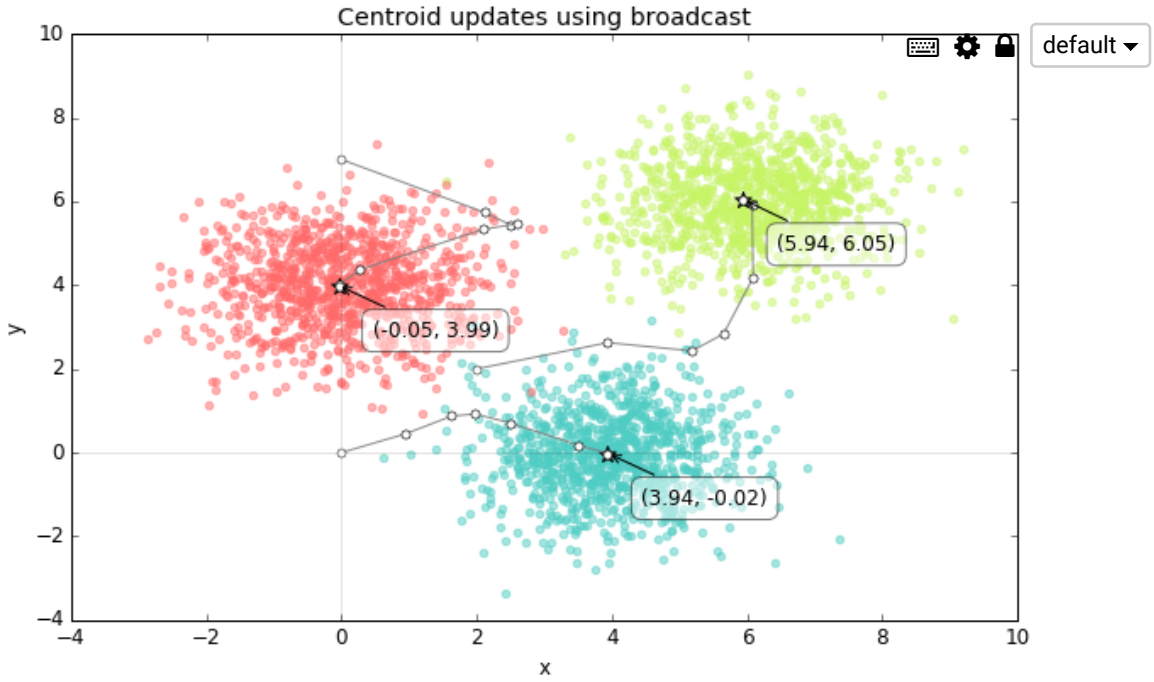
READY

Implement, comment your code and highlight your changes.

```
%spark.pyspark

# Calculate which class each data point belongs to

def nearest_centroid(line):
    x = np.array([float(f) for f in line.split(',')])

    # Customization START - access broadcast value
    closest_centroid_idx = np.sum((x - centroids.value)**2, axis=1).argmin()
    # Customization END

    return (closest_centroid_idx,(x,1))

# Distributed KMeans with Spark

K = 3
# Initialization: initialization of parameter is fixed to show an example

# Customization START - broadcast centroids
centroids = sc.broadcast(np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]]))
# Customization END

D = sc.textFile("data.csv").cache()

fig = plt.gcf()
fig.set_size_inches(8, 6)

plot_samples('Centroid updates using broadcast')

# Run iteration 10 times

for i in range(10):
    res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).co
    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 siz

    # Customization START - access broadcast value
    if np.sum(np.absolute(centroids_new-centroids.value))<0.01:
        break
    # Customization END

    # Customization START - access broadcast value
    plot_iteration(centroids.value, centroids_new)
    # Customization END

    # Customization START - broadcast centroids
    centroids = sc.broadcast(centroids_new)
    # Customization END
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous



READY

HW11.1 Loss Functions

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

FINISHED

```
%md

Logistic regression and linear SVM are different classification concepts, where line
different boundary hyperplanes and thus different models. Athough it has been demons
classify objects are different.
```

Logistic regression and linear SVM are different classification concepts, where linear SVM is minimizing margin while logistic regression is minimizing log loss (more details below). Therefore, the two can end up with different boundary hyperplanes and thus different models. Athough it has been demonstrated that a logistic regression that has both an L1 and L2 penalty can be reduced to a linear SVM, their basic criterions to classify objects are different.

Took 0 sec. Last updated by anonymous at November 11 2016, 11:11:28 AM.

Zeppelin Notebook

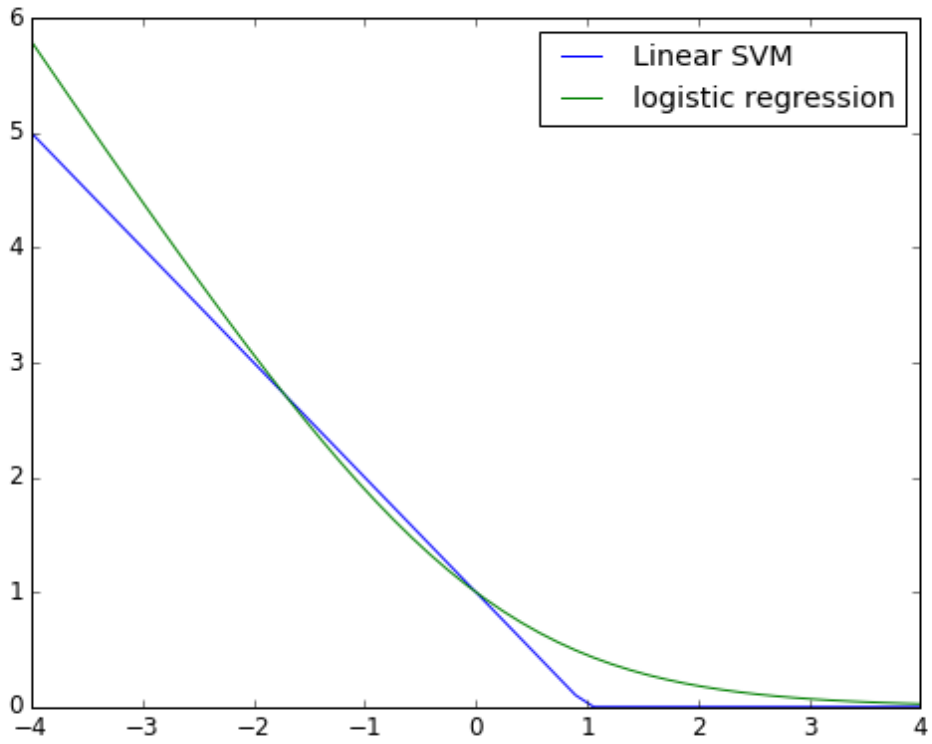
MIDS-W261-2016-...

Loss Function for Linear SVM vs. Logistic Regression

FINISHED anonymous

default

```
%spark.pyspark

from numpy import exp, log

# Linear SVM is hinge loss, logistic regression is log loss

x = np.linspace(-4, 4)
hinge_loss = [max(0, 1-n) for n in x]
log_loss = log(1 + exp(-x)) / log(2)

# Plot the loss functions on the same plot

fig = plt.gcf()
fig.set_size_inches(8, 6)

plt.plot(x, hinge_loss, label = 'Linear SVM')
plt.plot(x, log_loss, label = 'logistic regression')
plt.legend()

showMPL()
```



Took 0 sec. Last updated by anonymous at November 11 2016, 11:12:30 AM.

Zeppelin Notebook

MIDS-W261-2016-...

Separating Surfaces for Linear SVM vs. Logistic Regression

In binary classification, a linear SVM and logistic regression both have a hyperplane as a decision surface. Logistic regression fits the data with a continuous sigmoid function, where the separating hyperplane surface occurs where the probability of one of the classes is greater than some predetermined threshold. Linear SVM fits a hyperplane that attempts to separate the data into two classes that lie in either side of the hyperplane, and it chooses the hyperplane that maximizes the margin between two sets of objects.

default

Took 0 sec. Last updated by anonymous at November 11 2016, 11:18:03 AM. (outdated)

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

FINISHED

- If the two classes are not linearly separable and two models do not converges, they will have the same result although the two may diverge to entirely different weight vectors.
- If the two classes are linearly separable, the two may yield different models, because the perceptron learning algorithm does not attempt to minimize the margin; rather, it is finished as soon as it finds a weight vector corresponding to any of the hyperplanes that are able to separate the two classes. While the perceptron loss function is similar to the hinge loss function used by linear SVM, there is no penalty for a correct guess, regardless of the distance from the separating hyperplane. In such case, the classification results will be different.

Took 0 sec. Last updated by anonymous at November 11 2016, 11:33:06 AM. (outdated)

READY

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)?

READY

As noted in the Apache Spark documentation (reference (http://spark.apache.org/docs/latest/mllib-linear-methods.html#regularizers)), in the context of logistic regression, there are three flavors of penalized loss functions supported by Spark MLLib.

- **L1 penalty (lasso)**: The model is penalized by the L1 norm of the weight vector, $\|\mathbf{w}\|_1$ (reference (http://scikit-learn.org/stable/modules/generated

/sklearn.linear_model.Lasso.html))
- **L2 penalty (ridge)**: The model is penalized by the L2 norm of the weight vector, $\frac{1}{2}\|\mathbf{w}\|_2^2$ (reference (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html))
- **L1 and L2 penalty (elastic net)**: The model is penalized by a linear combination of the L1 norm and the L2 norm of the weight vector, $\alpha\|\mathbf{w}\|_1 + (1-\alpha)\frac{1}{2}\|\mathbf{w}\|_2^2$ (reference (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html))

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

FINISHED

When applying L1 penalties, we knwo that the value that minimizes the L1 norm is the median, which is equivalent to the result for the MLE estimator for a Laplace distribution. When applying L2 penalties, we know that the value that minimizes the L2 norm is the sample mean, which is equivalent to the result for the MLE estimator for a Gaussian distribution. Therefore, when we look at probabilistic interpretations of the L1 and L2 priors, they can be interpreted as a Laplacian prior and a Gaussian prior, respectively.

Took 0 sec. Last updated by anonymous at November 11 2016, 12:20:58 PM. (outdated)

READY

HW11.3 Logistic Regression

Generate 2 sets of linearly separable data with 100 data points each using the data generation code provided below. Call one the Training Set and the other the Testing Set. Plot each in separate plots.

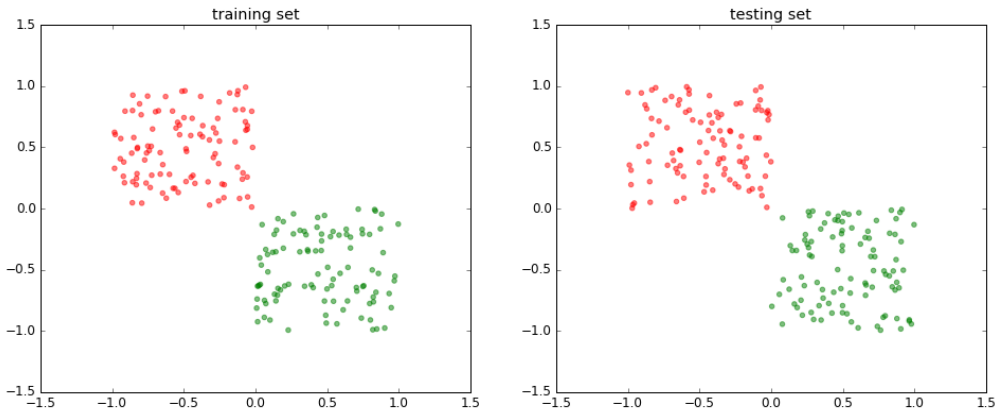
FINISHED

```
%spark.pyspark

# Provided data generation code

from numpy.random import rand

def generateData(n):
    """
    generates a 2D linearly separable dataset with n samples and
    an overlap of d percent where 0 <= d <= 100
    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
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 1 sec. Last updated by anonymous at November 11 2016, 12:30:23 PM. (outdated)

FINISHED

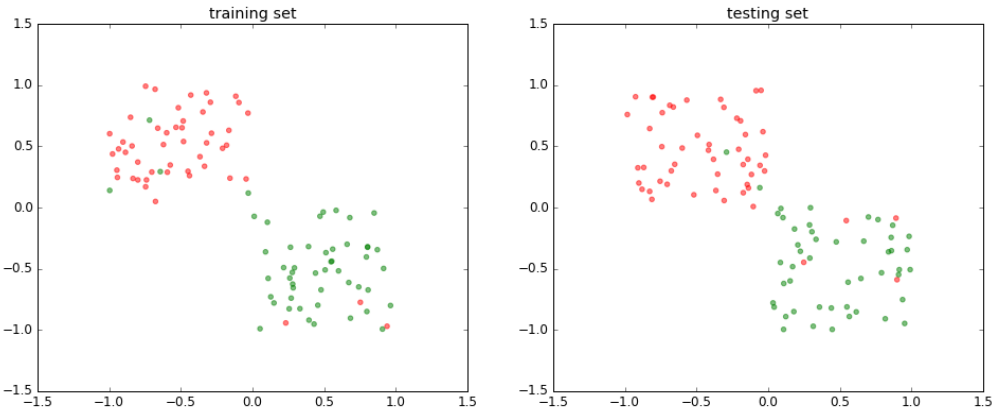
Modify this data generation code to generate non-linearly separable training and testing datasets (with approximately 10% of the data falling on the wrong side of the separating hyperplane). Plot the resulting data sets.

```
%spark.pyspark
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 1 sec. Last updated by anonymous at November 11 2016, 12:37:25 PM.

Using MLLib train up a LASSO logistic regression model with the training dataset.

FINISHED

```
%spark.pyspark

import functools

from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionModel, LogisticRegression

# Use the last element as the label, remaining other elements as the features
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

Took 46 sec. Last updated by anonymous at November 11 2016, 1:03:32 PM.

Evaluate With The Testing Set.

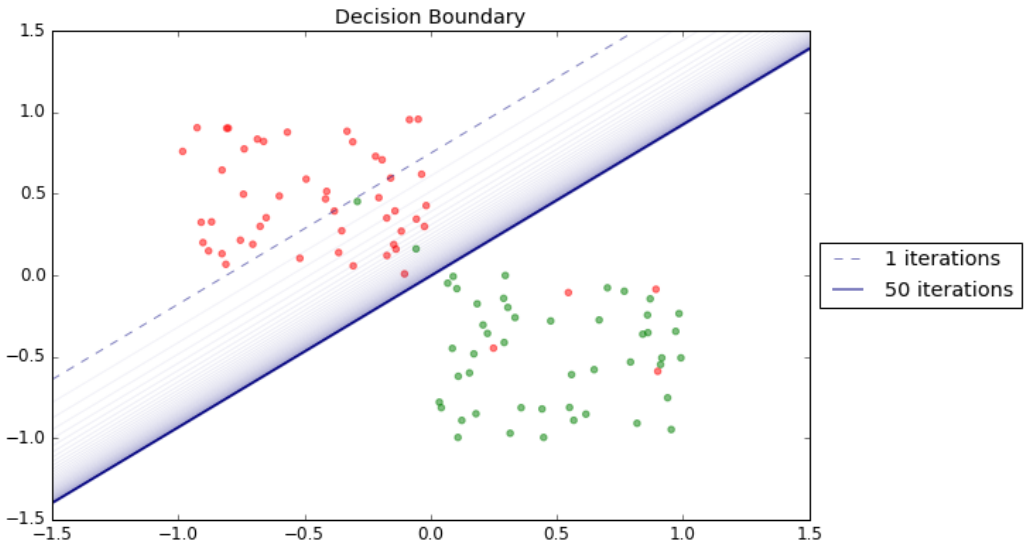
FINISHED

```
%spark.pyspark

# Plot the weights for an iteration

def plot_weights(iteration_id, model_weights, xlim, ylim, line_style, line_alpha, li
    slope = model_weights[0] / (-1 * model_weights[1])
    intercept = model_weights[2]

    x_values = xlim
    y_values = [x * slope + intercept for x in x_values]
    zip_values = np.array(zip(x_values, y_values))

    if iteration_id is not None:
        plot_label = str(iteration_id) + ' iterations'
    else:
        plot_label = None

    print plot_label
    plt.plot(zip_values[:, 0], zip_values[:, 1], linestyle=line_style, linewidth=lin

# Plot the weights forall iterations
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 1 sec. Last updated by anonymous at November 11 2016, 1:09:19 PM.

Zeppelin Notebook

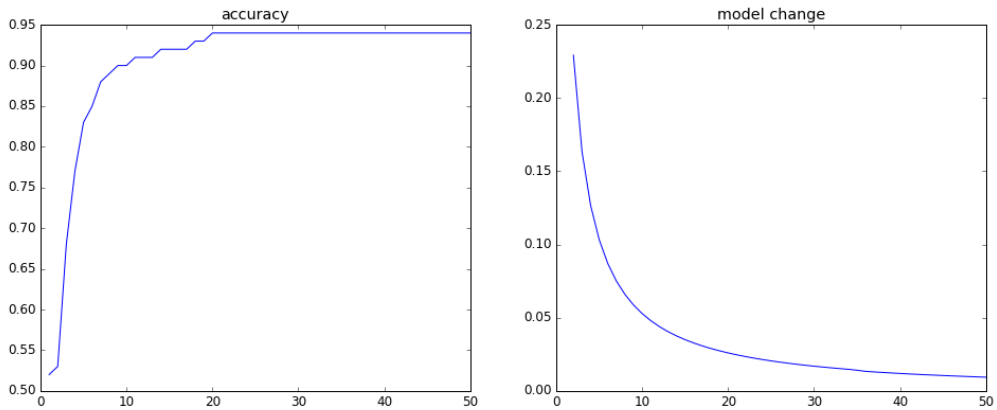
MIDS-W261-2016-...

anonymous

%spark.pyspark

default

```
# Initialize our plot

fig = plt.gcf()
fig.set_size_inches(16, 6)

# Get accuracy with the given set of weights using the testing set

def get_accuracy(model_weights):
    model = LogisticRegressionModel(model_weights[:-1], model_weights[-1], 2, 2)

    # Compare the predictions the actual result

    predicted_labels_rdd = model.predict(testing_features_rdd)

    return predicted_labels_rdd.zip(actual_labels_rdd).map(lambda x: 1.0 if x[0] ==

# Plot accuracies

mllib_accuracies = [get_accuracy(model_weights) for model_weights in mllib_weights]
plt.subplot(1, 2, 1)
plt.title('accuracy')
plt.plot(mllib_iterations, mllib_accuracies)

# Returns how much has changed at model

def model_changes(prev_weights, next_weights):
    weight_change = np.array(prev_weights) - np.array(next_weights)
    return np.linalg.norm(weight_change)

# Plot our model changes

mllib_model_changes = [model_changes(prev_weights, next_weights) for prev_weights, n
plt.subplot(1, 2, 2)
plt.title('model change')
plt.plot(mllib_iterations[1:], mllib_model_changes)

showMPL()
```



Zeppelin Notebook

Last updated by anonymous at November 11 2016, 4:05:03 PM.

MIDS-W261-2016-...

What is a good number of iterations for training the logistic regression model? Justify with words. (Part 1)

FINISHED anonymous

```
%spark.pyspark

def show_iteration_row(iteration, iteration_weights, accuracy, model_change):
    print str(iteration) + '\t' + str(accuracy) + '\t' + str(iteration_weights[:-1])

def show_iteration_table(iterations, weights, accuracies, model_changes):
    print '%table iterations\taccuracy\tweights\tintercept\tmodel change\tmisclassif

    indices = np.arange(9, len(iterations), 5)

    show_iteration_row(iterations[0], weights[0], accuracies[0], None)

    for i in indices:
        show_iteration_row(iterations[i], weights[i], accuracies[i], model_changes[i

    if (len(iterations) - 1) not in indices:
        show_iteration_row(iterations[-1], weights[-1], accuracies[-1], model_change

show_iteration_table(mllib_iterations, mllib_weights, mllib_accuracies, mllib_model_
```

default

iterations	accuracy	weights
1	0.52	[-0.20153735905318643, 0.21751684051067574]
10	0.9	[-0.75058142331531208, 0.80645715879161772]
15	0.92	[-0.87769491653208154, 0.94256901197058873]
20	0.94	[-0.96993734337855309, 1.0415333062224583]
25	0.94	[-1.0419619398707063, 1.1190143842198956]
30	0.94	[-1.1007644391207883, 1.1824669010221627]
35	0.94	[-1.1502425382742614, 1.236034472159691]
40	0.94	[-1.1927673939178214, 1.2822711528870046]
45	0.94	[-1.2298957850494976, 1.3228545849233726]
50	0.94	[-1.2627381596331819, 1.3589382500221843]

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

Took 0 sec. Last updated by anonymous at November 11 2016, 4:14:18 PM.

READY

What is a good number of iterations for training the logistic regression model? Justify with words. (Part 2)

From an accuracy perspective, we can stop after 10 iterations. The model change drops to the 0.01 threshold at around 50 iterations. If we were to use the change in the model as our measure of what would be a good number of iterations, we would stop at 50 iterations.

READY

Derive and implement in Spark a weighted LASSO logistic regression. Weight each example using the inverse vector length (Euclidean norm).

$$weight(X) = \frac{1}{\|X\|} \text{ where } \|X\| = \sqrt{X \cdot X} = \sqrt{X_1^2 + X_2^2}$$

FINISHED

```
%spark.pyspark

# Add bias term and transform points into weighted points

from collections import namedtuple

Point = namedtuple('Point', 'x y w')

def readPoint(labeledPoint):
    x = labeledPoint.features
    x = np.append(x, 1.0)
    return Point(x, labeledPoint.features[-1], 1 / np.linalg.norm(x))

weighted_training_rdd = training_rdd.map(readPoint).cache()
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

Took 0 sec. Last updated by anonymous at November 11 2016, 4:18:14 PM.

Implement a convergence test of your choice to check for termination within your training algorithm. Report how many iterations It took to converge.

FINISHED

```
%spark.pyspark

# Initialize the weights and the tracking of the iterations

previous_weights = None

normed_iterations = []
normed_weights = []

converged = False

# Run iterations until we have converged. We'll assume convergence is when the chang
# in the model is below 0.01.

iteration_id = 0

while not converged and iteration_id < 500:
    iteration_id += 1
    next_weights = logistic_regression_weight(previous_weights)

    normed_iterations.append(iteration_id)
    normed_weights.append(next_weights)

    if iteration_id > 1
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

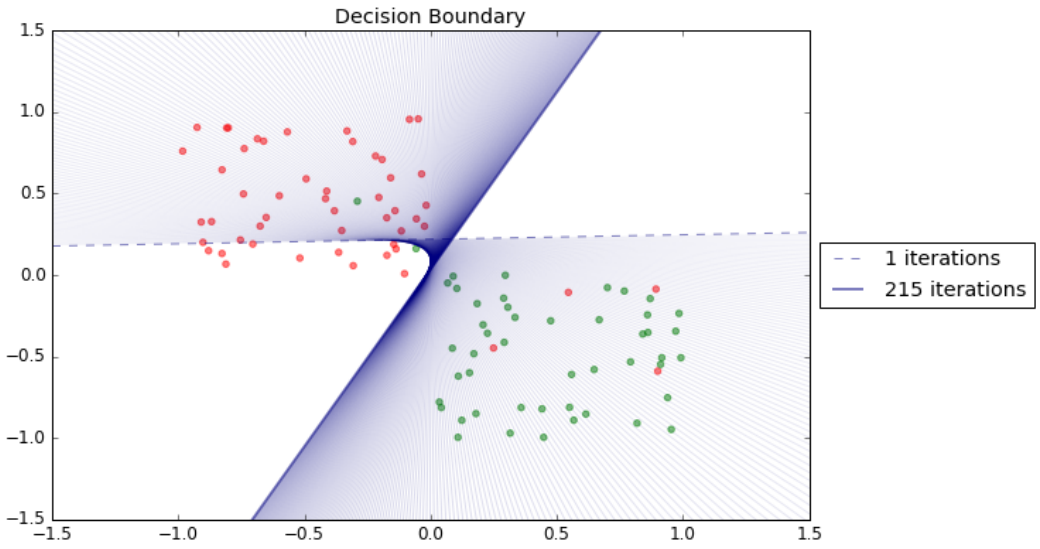
```
Converged in 215 iterations
```

Took 55 sec. Last updated by anonymous at November 11 2016, 4:21:59 PM.

Evaluate your homegrown weighted LASSO logistic regression on the test data set.

FINISHED

```
%spark.pyspark

evaluate_model_progress(normed_iterations, normed_weights, 10)
```



Took 3 sec. Last updated by anonymous at November 11 2016, 4:22:46 PM. (outdated)

Report misclassification error (1 - Accuracy). (Part 1)

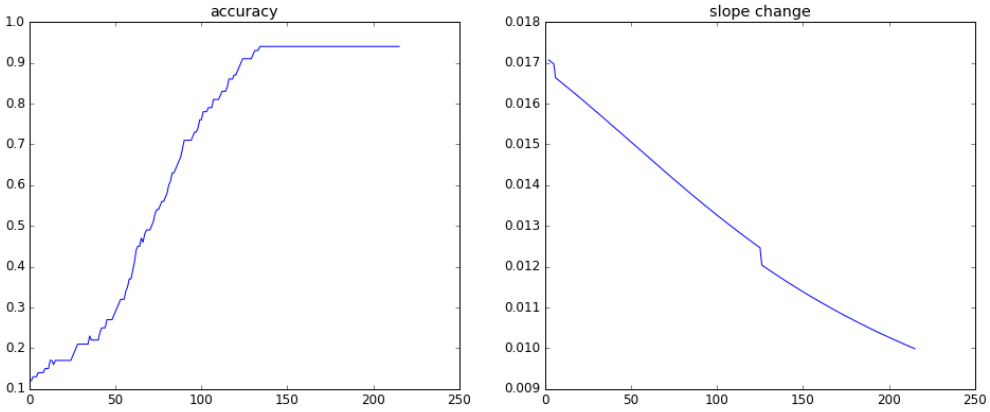
FINISHED

```
%spark.pyspark

# Initialize our plot

fig = plt.gcf()
fig.set_size_inches(16, 6)

# Plot our accuracies

normed_accuracies = [get_accuracy(model_weights) for model_weights in normed_weights
```

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default



Took 49 sec. Last updated by anonymous at November 11 2016, 4:33:12 PM.

Report misclassification error (1 - Accuracy). (Part 2)

FINISHED

```
%spark.pyspark

show_iteration_table(normed_iterations, normed_weights, normed_accuracies, normed_mo
```

iterations	accuracy	weights
1	0.12	[0.03998026 -1.44264413]
10	0.15	[-0.05083082 -1.32279609]
15	0.17	[-0.09948205 -1.25718634]
20	0.17	[-0.14763361 -1.1922851]
25	0.18	[-0.19527892 -1.12810315]
30	0.21	[-0.24241238 -1.06464992]
35	0.23	[-0.2890294 -1.00193339]
40	0.22	[-0.33512647 -0.93996001]

...tions	accuracy	weights
45	0.27	[-0.38070119 -0.87873461]
50	0.29	[-0.42575233 -0.81826039]
55	0.32	[-0.47027982 -0.7585389]
60	0.39	[-0.51428475 -0.69957004]

Took 0 sec. Last updated by anonymous at November 11 2016, 4:33:47 PM.

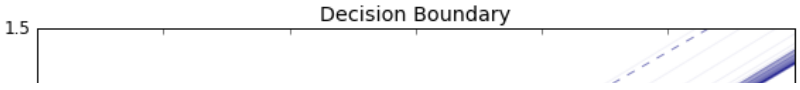
READY

HW11.4 SVMs

Using MLLib train up a soft SVM model with the training dataset and evaluate with the testing set.

FINISHED

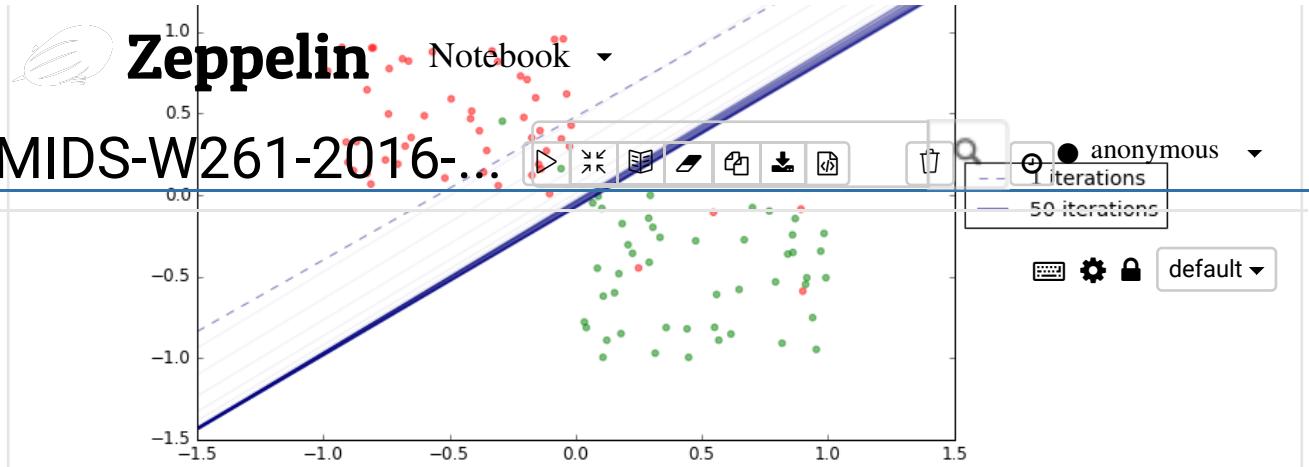
```
%spark.pyspark

from pyspark.mllib.classification import SVMModel, SVMWithSGD

# Define a function to iterate from previous SVM

def mllib_svm(n):
    model = SVMWithSGD.train(data = training_rdd, regType='l2', iterations=n, initia
    return list(model.weights) + [model.intercept]

# Initialize the weights and the tracking of the iterations

previous_weights = [1.0, 1.0]

mllib_svm_iterations = []
mllib_svm_weights = []

# Run iterations

for i in range(50):
    next_weights = mllib_svm(i + 1)

    mllib_svm_iterations.append(i + 1)
    mllib_svm_weights.append(next_weights)

    previous_weights = next_weights

evaluate_model_progress(mllib_svm_iterations, mllib_svm_weights, 10)
```



Zeppelin Notebook

MIDS-W261-2016-...

anonymous



default

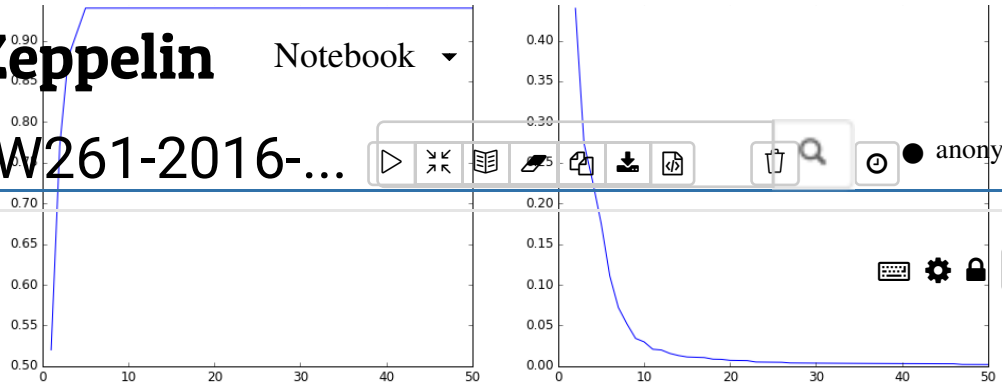
Took 47 sec. Last updated by anonymous at November 11 2016, 4:43:32 PM.

What is a good number of iterations for training the SVM model? Justify with plots.

FINISHED

```
%spark.pyspark

# Initialize our plot

fig = plt.gcf()
fig.set_size_inches(16, 6)

# Compute accuracy

def svm_accuracy(model_weights):
    model = SVMModel(model_weights[:-1], model_weights[-1])

    # Compare the predictions with the actual result

    predicted_labels_rdd = model.predict(testing_features_rdd)

    return predicted_labels_rdd.zip(actual_labels_rdd).map(lambda x: 1.0 if x[0] ==

# Plot our accuracies

mllib_svm_accuracies = [svm_accuracy(model_weights) for model_weights in mllib_svm_w
plt.subplot(1, 2, 1)
plt.title('accuracy')
plt.plot(mllib_svm_iterations, mllib_svm_accuracies)

# Plot our model changes

mllib_svm_model_changes = [get_model_changes(prev_weights, next_weights) for prev_we
plt.subplot(1, 2, 2)
plt.title('model change')
plt.plot(mllib_svm_iterations[1:], mllib_svm_model_changes)

showMPL()
```

Zeppelin Notebook MIDS-W261-2016-... anonymous default



Took 11 sec. Last updated by anonymous at November 11 2016, 5:17:31 PM.

What is a good number of iterations for training the SVM model? Justify with words. (Part 1)

FINISHED

```
%spark.pyspark

show_iteration_table(mllib_svm_iterations, mllib_svm_weights, mllib_svm_accuracies,
```

iterations	accuracy	weights
1	0.52	[-0.20400607811723528, 0.23212146734005124]
10	0.94	[-1.0745085957528198, 1.1471873119286888]
15	0.94	[-1.1258929137030946, 1.2075436817472494]
20	0.94	[-1.1523016956802838, 1.2431270791379925]
25	0.94	[-1.1673858371725805, 1.2664891252296382]
30	0.94	[-1.1773837356023422, 1.2815765662001093]
35	0.94	[-1.1859332957936908, 1.2943960515045529]
40	0.94	[-1.1938319380876052, 1.3062395295953162]
45	0.94	[-1.2011977021582001, 1.3172839929531559]
50	0.94	[-1.2042176281708676, 1.3271853604703352]

Zeppelin Notebook

MIDS-W261-2016-... anonymous

default

Took 0 sec. Last updated by anonymous at November 11 2016, 5:18:00 PM.

What is a good number of iterations for training the SVM model? Justify with words. (Part 2)

FINISHED

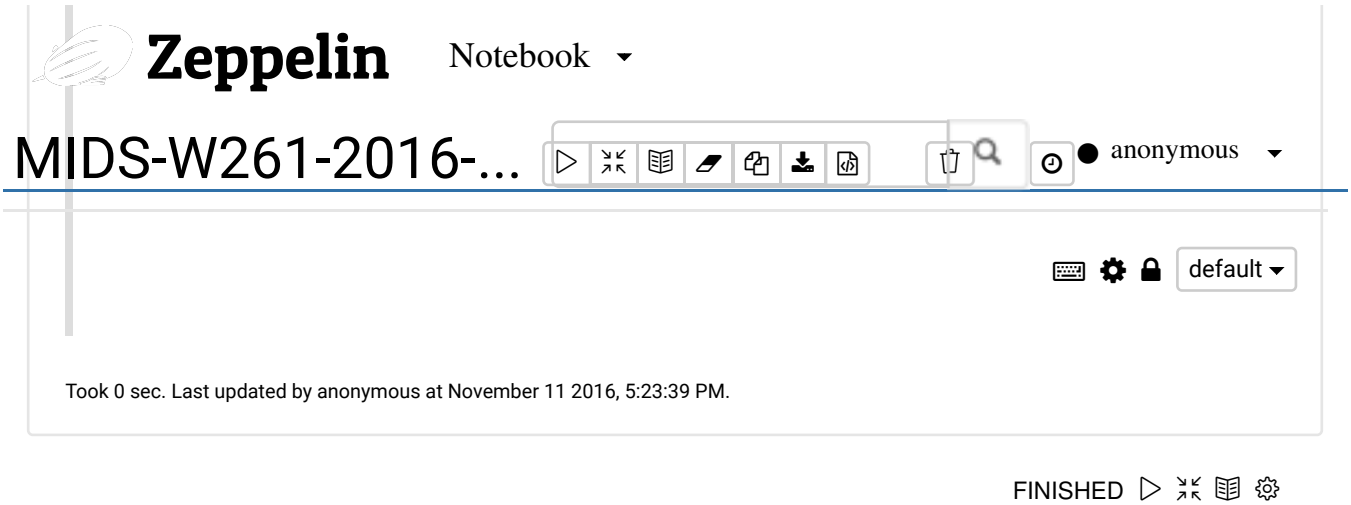
With the current initial weights, accuracy peaks after 5 iterations, so we will need at least that many iterations. Since the SVM model is actually trying to minimize the margin, the model change is more important than the accuracy. We notice that the model is very stable after 10 iterations, which makes 10 iterations a good choice for this model.

Took 0 sec. Last updated by anonymous at November 11 2016, 5:20:30 PM. (outdated)

Derive and Implement in Spark a weighted linear SVM classification learning algorithm.

FINISHED

```
%spark.pyspark

# Define a function to iterate on top of the weights from a previous SVM iteration

def svm_weight(previous_weights=None, learning_rate = 0.05, reg_param = 0.01):
    n = weighted_training_rdd.count()
    featureLen = len(weighted_training_rdd.take(1)[0].x)

    # Initialize weights to random
    if previous_weights is None:
        previous_weights = np.random.normal(size=featureLen)

    w = sc.broadcast(previous_weights)

    # Get support vectors
    suport_vectors = weighted_training_rdd.filter(lambda p: p.y * np.dot(w.value, p.

    if suport_vectors.isEmpty():
        return previous_weights

    # Calculate gradient
    gradient_tuple = suport_vectors.map(lambda p: (p.w * p.y * np.array(p.x), p.w)).
    gradient = -gradient_tuple[0] / n
```


Zeppelin
Notebook
MIDS-W261-2016-...
anonymous
default
Took 0 sec. Last updated by anonymous at November 11 2016, 5:23:39 PM.

FINISHED

Evaluate your homegrown weighted soft linear SVM classification learning algorithm on the weighted training dataset and test dataset from HW11.3. Report how many iterations it took to converge.

```
%spark.pyspark

# Initialize the weights and the tracking of the iterations

previous_weights = None

normed_svm_iterations = []
normed_svm_weights = []

converged = False

# Run iterations until we have converged. We'll assume convergence is when the chang
# in the model is below 0.01.

iteration_id = 0

while not converged and iteration_id < 500:
    iteration_id += 1
    next_weights = svm_weight(previous_weights)

    normed_svm_iterations.append(iteration_id)
    normed_svm_weights.append(next_weights)

    if iteration_id > 1:
        model_change = get_model_changes(previous_weights, next_weights)
        converged = model_change <= 0.01

    previous_weights = next_weights

if converged:
    print 'Converged in', len(normed_svm_iterations), 'iterations'
else:
    print 'Failed to converge in', len(normed_svm_iterations), 'iterations'
```

default

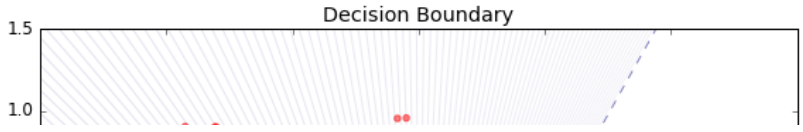
```
Converged in 228 iterations
```

Took 1 min 4 sec. Last updated by anonymous at November 11 2016, 5:25:00 PM.

Evaluate Your Homegrown Weighted Linear SVM On The Test Dataset.

FINISHED

```
%spark.pyspark

evaluate_model_progress(normed_svm_iterations, normed_svm_weights, 5)
```

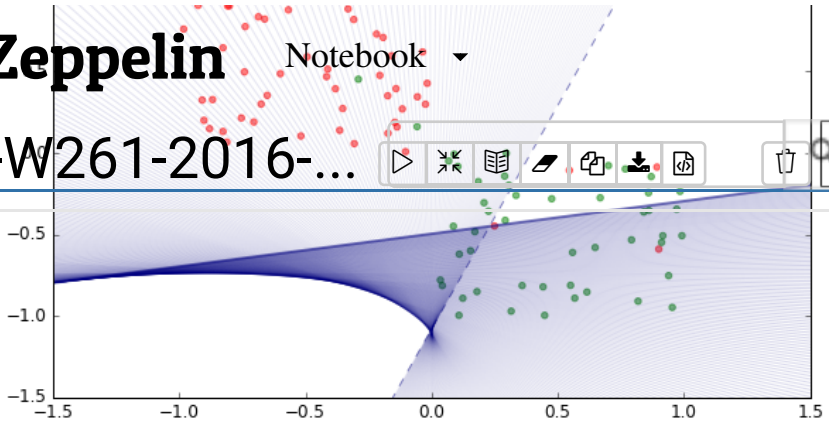


Zeppelin Notebook

MIDS-W261-2016-...

1 iterations
28 iterations
anonymous

default

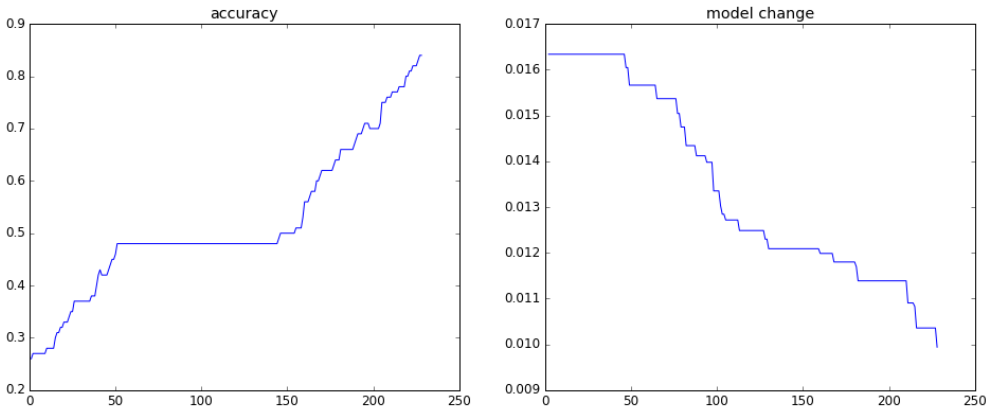


Took 2 sec. Last updated by anonymous at November 11 2016, 5:26:52 PM.

Report misclassification error (1 - Accuracy). (Part 1)

FINISHED

```
%spark.pyspark

# Initialize our plot

fig = plt.gcf()
fig.set_size_inches(16, 6)

# Plot our accuracies

normed_svm_accuracies = [svm_accuracy(model_weights) for model_weights in normed_svm
plt.subplot(1, 2, 1)
plt.title('accuracy')
plt.plot(normed_svm_iterations, normed_svm_accuracies)

# Plot our model changes

normed_svm_model_changes = [get_model_changes(prev_weights, next_weights) for prev_w
plt.subplot(1, 2, 2)
plt.title('model change')
plt.plot(normed_svm_iterations[1:], normed_svm_model_changes)

showMPL()
```



Took 51 sec. Last updated by anonymous at November 11 2016, 5:28:53 PM.

Zeppelin Notebook

MIDS-W261-2016-...

Report misclassification error (1 - Accuracy) (Part 2)

FINISHED anonymous

```
%spark.pyspark

show_iteration_table(normed_svm_iterations, normed_svm_weights, normed_svm_accuracie
```

default

iterations	accuracy	weights

iterations	accuracy	weights
45	0.42	[1.18874024 -0.03414027]
50	0.46	[1.13800838 0.02737136]
55	0.48	[1.08729525 0.08703478]
60	0.48	[1.03658212 0.14669821]
65	0.48	[0.98594783 0.20604249]
70	0.48	[0.9356289 0.26411022]
75	0.48	[0.88530997 0.32217795]
80	0.48	[0.8358146 0.37839292]
85	0.48	[0.7873514 0.43140863]
90	0.48	[0.73932187 0.48316208]

Took 0 sec. Last updated by anonymous at November 11 2016, 5:29:25 PM.

How many support vectors do you end up with?

FINISHED

Zeppelin Notebook

MIDS-W261-2016-... anonymous

```
75 support vectors
```

default

Took 0 sec. Last updated by anonymous at November 11 2016, 5:41:00 PM.

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.

FINISHED

```
%md

We could not identify a built-in weighted soft SVM learner.
```

We could not identify a built-in weighted soft SVM learner.

Took 0 sec. Last updated by anonymous at November 11 2016, 5:30:11 PM.

READY