DAMLAS Part 2
Classification via gradient descent, 7/15

James G. Shanahan [1,2]

[1]Church and Duncan Group, *[2]iSchool UC Berkeley, CA*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

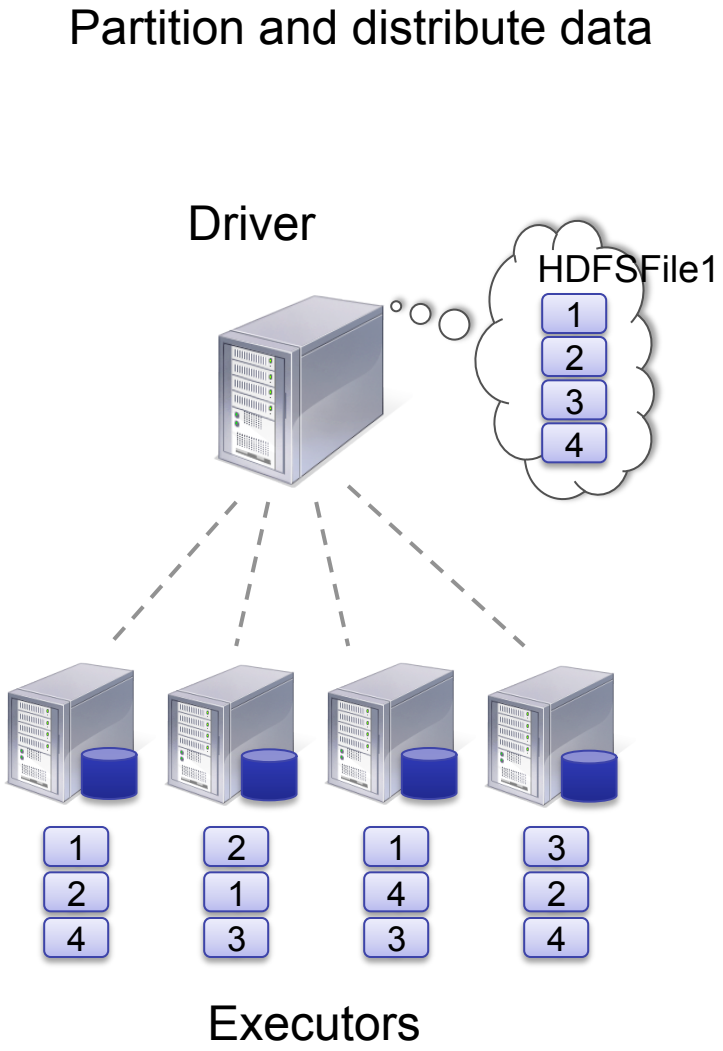
Part 2 Lecture 2
July 13, 2016

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

RDD is a distributed collection of elements

- **In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.**
- **An RDD is laid out across a cluster of machines as collection of Partitions, each including a subset of the data**
- **The framework processes the objects within a partition in sequence, and processes multiple partitions in parallel.**
- **Data (e.g., clicks, or record linkage) is stored in a text file, with one observation on each line.**
  - JSON, zipped, AVRO, Parquet



RDD with 4 elements

Distribute

Map

Gather back to driver

```
In [19]: #RDD mapper
         sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()

         #returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]
```

```
In [19]: #RDD mapper
         sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()

         #returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]
```

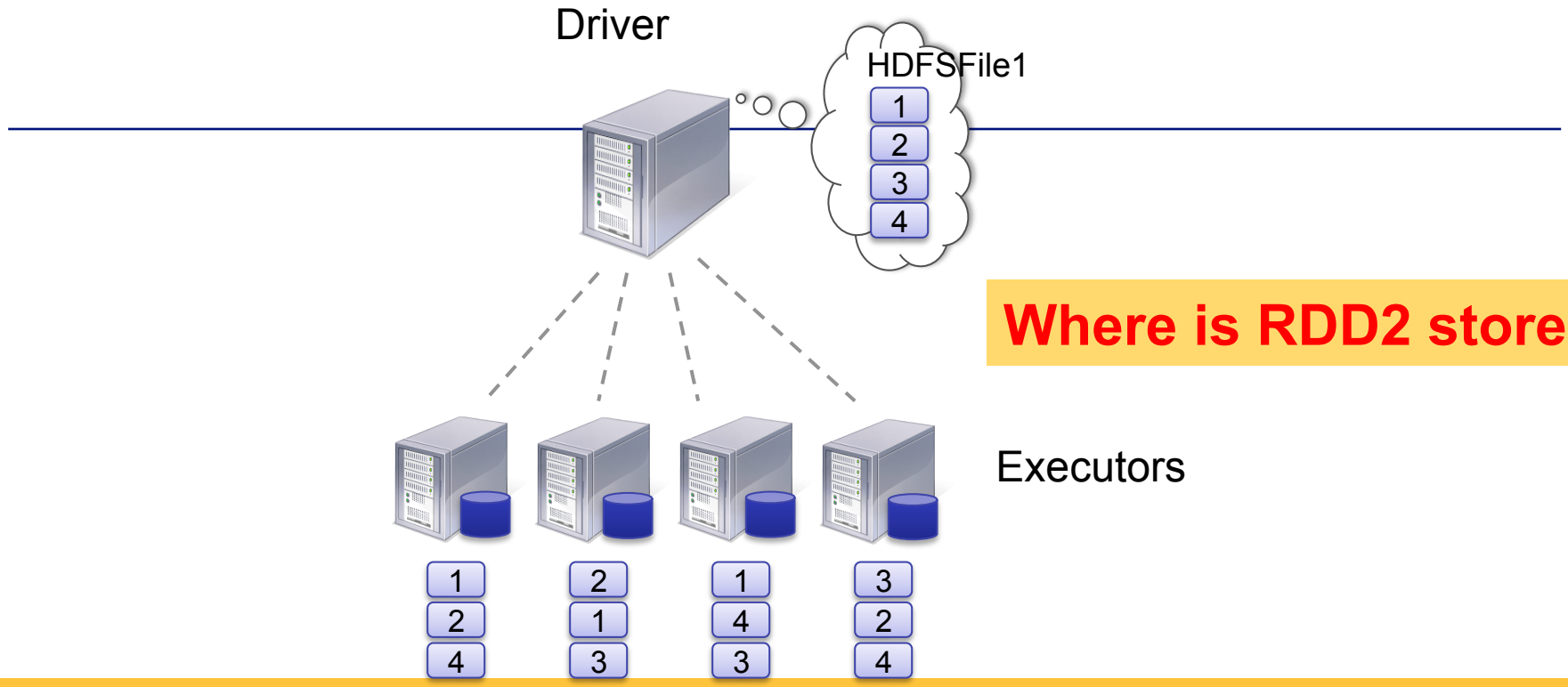
```
In [33]: def powerOfTwo(x):
             return x*x

         def plusOne(x):
             return x+1

         def myReduce(x, y):
             return x+y

         print "Reduce by lambda:   %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
         print "Reduce by fucntion: %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(myReduce)
         #.reduce()
```

Explain this code

```
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```

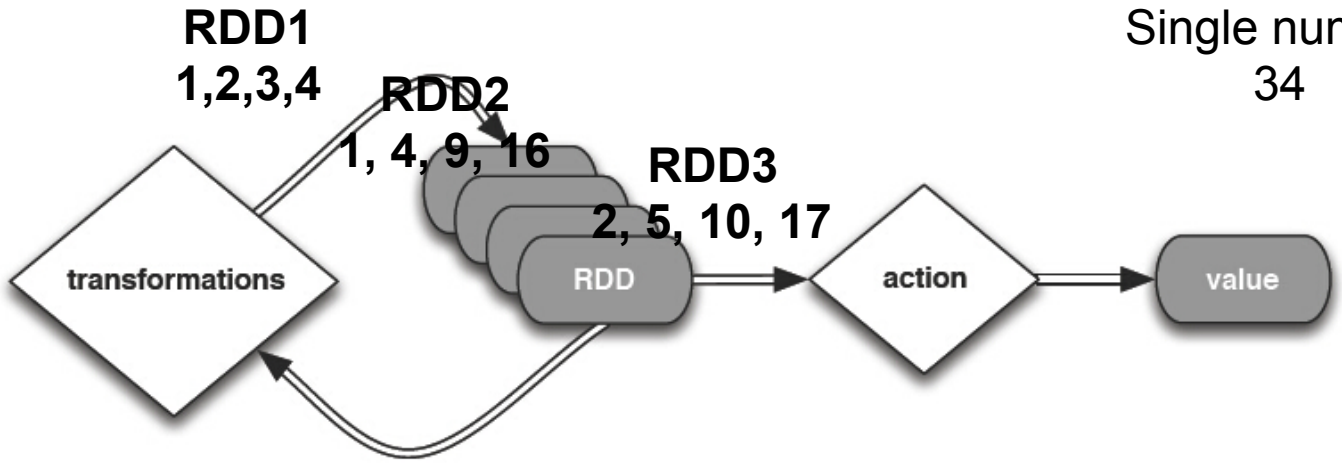
- **How is the data for the mapper distributed?**
- **In memory? NO!**
  - .cache()

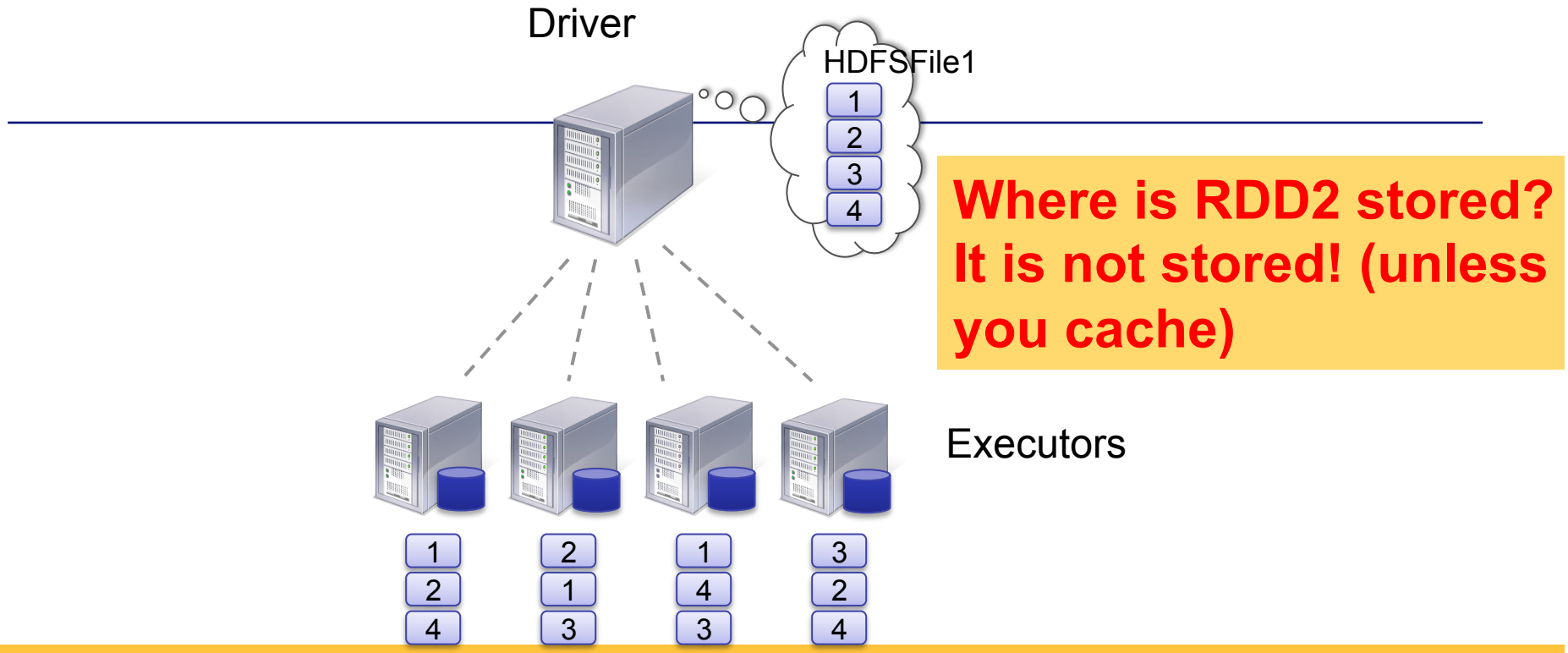
```
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```



Where is RDD2 stored?

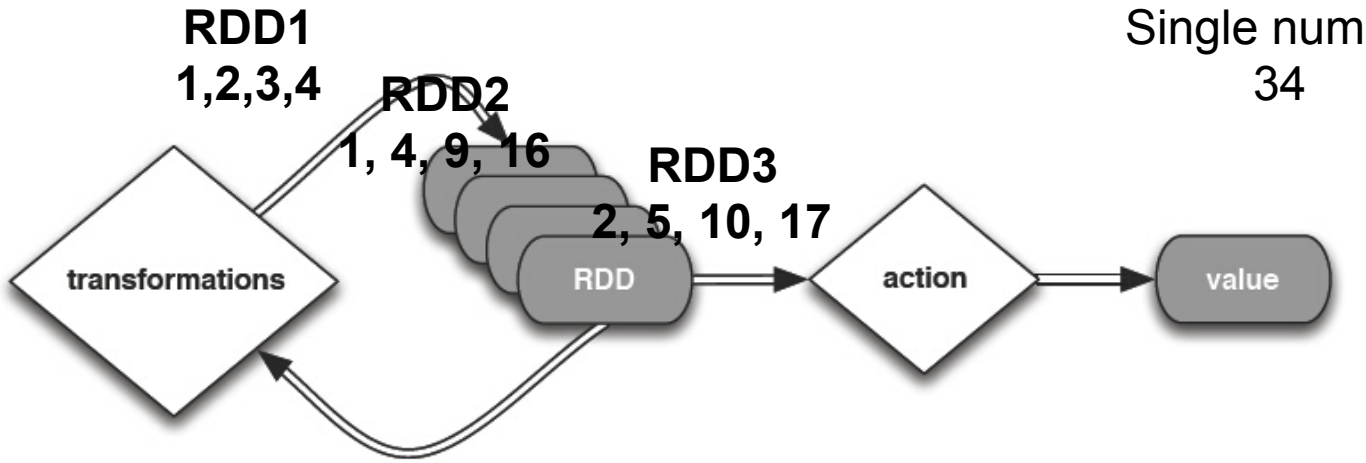
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)





Where is RDD2 stored? It is not stored! (unless you cache)

sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)



Cache versus broadcast

- **In the context of ML, we have a training set**
  - Process training data once per training iteration

STATIC

  - Read from disk or from memory?
    - So cache training data in memory
  - Training data is fixed for the duration of training

- **Broadcasting (it will put it in memory)**

Dynamic

  - 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.

Broadcast Variables analysis

- https://www.dropbox.com/s/th16dnn33c1m9lz/mosharaf-spark-broadcast-variables-report-spring10.pdf?dl=0

Performance and Scalability of Broadcast in Spark

Mosharaf Chowdhury*
University of California, Berkeley

ABSTRACT

Although the MapReduce programming model has so far been highly successful, not all applications are well suited to this model. Spark bridges this gap by providing seamless support for iterative and interactive jobs that are hard to express using the acyclic data flow model pioneered by MapReduce. While benchmarking Spark, we identified that the default broadcast mechanism implemented in the Spark prototype is a hindrance toward its scalability.

In this report, we implement, evaluate, and compare four different broadcast mechanisms (including the default one) for Spark. We outline the basic requirements of a broadcast mechanism for Spark and analyze each of the compared broadcast mechanisms under that guideline. Our experiments in high-speed, low-latency, and cooperative data center environments also shed light on characteristics of multicast and broadcast mechanisms in data centers in general.

1. INTRODUCTION

With the advent of MapReduce [12] and similar frameworks as well as of cloud services like Amazon's EC2 [1], a new model of cluster computing has become mainstream in recent years. In this model, data-parallel computations are executed on commodity clusters by systems that automati-

that the default broadcast mechanism (CHB[1]) in its prototype implementation is an impediment toward its scalability. Specifically, the default broadcast implementation takes a centralized approach where the broadcast variable is serialized and written to the HDFS by the sender and all the receivers read and deserialize it to reconstruct the variable - essentially creating a bottleneck at the HDFS. Further pondering revealed that in addition to scalability, performance, fault tolerance, and adaptability to unstructured clusters are three more major requirements for a viable broadcast mechanism for Spark and similar frameworks running on commodity clusters.

In this report, we consider three more broadcast mechanisms: Chained Streaming Broadcast (CSB), BitTorrent Broadcast (BTB), and SplitStream Broadcast (SSB), each with diverse characteristics. We have implemented, evaluated, and compared them to identify the better choice for broadcast in data center environments. While multicast and broadcast are well understood topics in computer networks, to the best of our knowledge, all of them assume that nodes are distributed across the Internet; whereas, we are concerned about high-speed, low-latency, and cooperative data center environments. From that perspective, in addition to making Spark more scalable, we expect our work to enrich the networking literature as well.

Linear Regression

Objective Function

$$\min_W L(w) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)^2$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)\cdot x_i$$

Gradient descent (no regularization)

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0)  #bias term
    return Point(x, float(d[0]))


def linearRegressionGD(data, wInitial=None, learningRate=0.05, iterations=50):
    featureLen = len(data.take(1)[0].x)
    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 p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
                   .reduce(lambda a, b: a + b)
        w = w - learningRate * gradient/n
    return w
```

Sum up gradients

```
data = sc.textFile('data.csv').map(readPoint).cache()
linearRegressionGD(data)
```

```
array([ 7.98418741, -1.61969498])
```

https://www.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb?dl=0

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb

Get gradients for each instance

Linear Regression: example of using broadcast

Objective Function

$$\min_W L(w) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)^2$$

Question: Combiner?

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)\cdot x_i$$

Gradient descent (no regularization)

```
: from collections import namedtuple
  import numpy as np
  Point = namedtuple('Point', 'x y')

  def readPoint(line):
      d = line.split(',')
      x = [float(i) for i in d[1:]]
      x.append(1.0)  #bias term
      return Point(x, float(d[0]))
```

```
wBroadcast = sc.broadcast(w)
gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)
            .reduce(lambda a, b: a + b)
w = w - learningRate * gradient/n
```

```
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        w = w - learningRate * gradient/n
    return w
```

Sum up gradients

Get gradients for each instance

```
: data = sc.textFile('data.csv').map(readPoint).cache()
  linearRegressionGD(data)
```

```
: array([ 7.98418741, -1.61969498])
```

https://www.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb?dl=0

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb

Example problem: Kmeans: Broadcast or not?!

- **900Gig of data: assume 1Gig chunks**
  - Ship (serialize) each mapper and centroids data to each executor 900 times

- **VERSUS**
  - When we broadcast centroids we only ship 5 times (where we have, say, 5 executors or worker nodes)

Kmeans in Spark

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

- **NBViewer**
  - http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb

Kmeans In Spark:
E-Step: Cluster Assignment Function

```
SparkContext available as sc, HiveContext available as sqlContext.
```

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2, \; i = 1, \ldots, N$$

```
1  import numpy as np
2
3  #Calculate which class each data point belongs to
4  def nearest_centroid(line):
5      x = np.array([float(f) for f in line.split(',')])
6      closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin()
7      return (closest_centroid_idx,(x,1))
8
9  #plot centroids and data points for each iteration
10 def plot_iteration(means):
11     pylab.plot(samples1[:, 0], samples1[:, 1], '.', color = 'blue')
12     pylab.plot(samples2[:, 0], samples2[:, 1], '.', color = 'blue')
13     pylab.plot(samples3[:, 0], samples3[:, 1],'.', color = 'blue')
14     pylab.plot(means[0][0], means[0][1],'*',markersize =10,color = 'red')
15     pylab.plot(means[1][0], means[1][1],'*',markersize =10,color = 'red')
16     pylab.plot(means[2][0], means[2][1],'*',markersize =10,color = 'red')
17     pylab.show()
```

centroidsBroadcast = broadcast(centroids) #to broadcast

centroidsBroadcast.value #to access the variable

Kmeans In Spark:

Lines 9-21 are local to driver

```
3  #Calculate which class each data point belongs to
4  def nearest_centroid(line):
5      x = np.array([float(f) for f in line.split(',')])
6      closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin(
7      return (closest_centroid_idx,(x,1))
```

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1,\ldots,K$$

E-Step

M-Step Sum Part 1

```
1  K = 3
2  # Initialization: initialization of parameter is fixed to show an example
3  centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
4
5  D = sc.textFile("./data.csv").cache()
6  iter_num = 0
7  for i in range(10):
8      res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).collect(
9      #res [(0, (array([  2.66546663e+00,   3.94844436e+03]), 1001)  ),
10     #     (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
11     #     (1, (array([ 3986.85984761,    15.93153464]), 999))]
12     # res[1][1][1] returns 1000 here
13     res = sorted(res,key = lambda x : x[0])  #sort based on clusted ID
14     centroids_new = np.array([x[1][0]/x[1][1] for x in res])  #divide by cluster size
15     if np.sum(np.absolute(centroids_new-centroids))<0.01:
16         break
17     print "Iteration" + str(iter_num)
18     iter_num = iter_num + 1
19     centroids = centroids_new
20     print centroids
21     plot_iteration(centroids)
22 print "Final Results:"
23 print centroids
```

M-Step Sum Part 2

Kmeans In Spark: Cluster Assignment Function

```
Iteration0
[[ 0.80217059  0.61622248]
 [ 3.94191443  2.67285704]
 [ 2.18436067  5.72221018]]
```

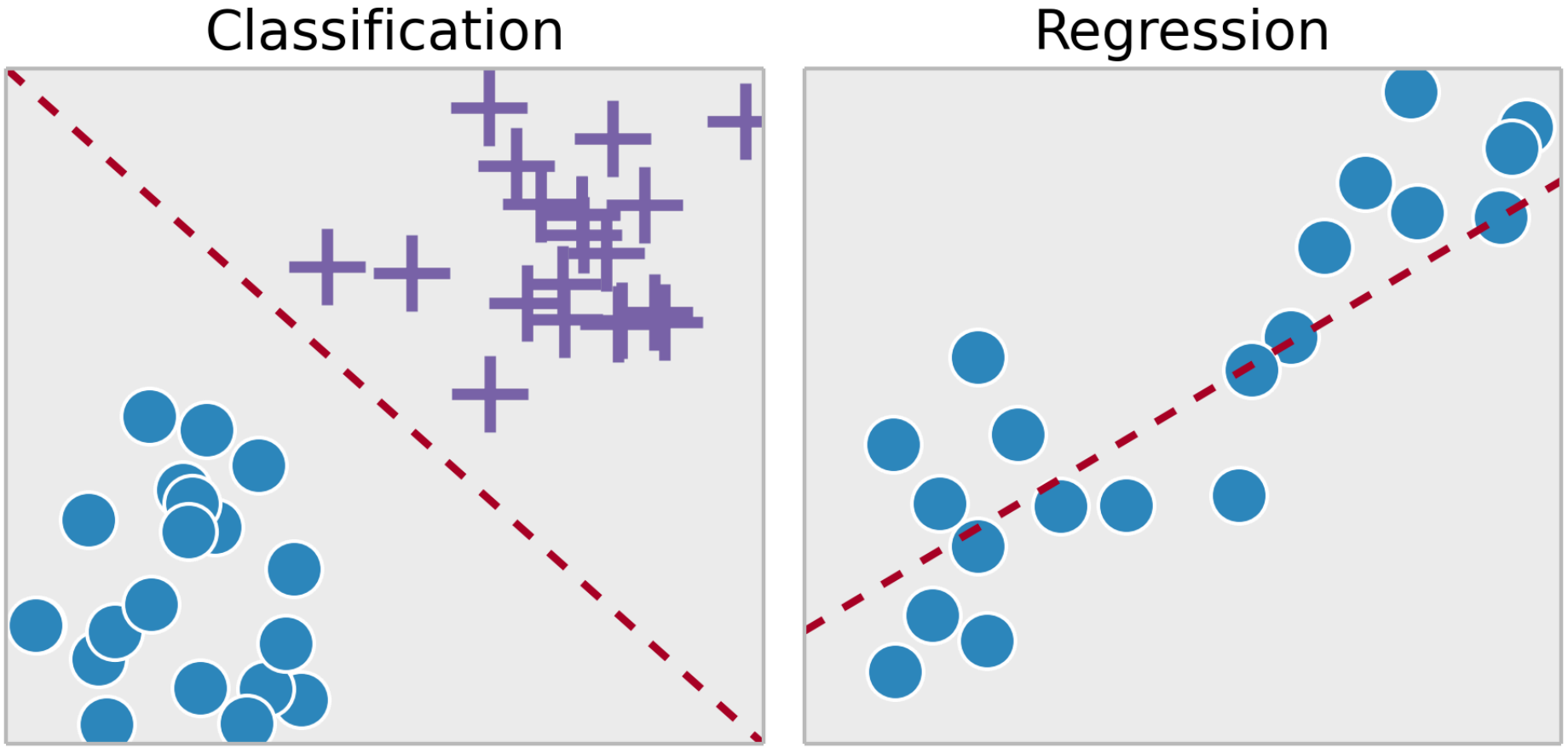
Kmeans Question

- How can we make the above code be more efficient (especially when our centroid vectors are long, i.e., having, say 10s of millions of elements)?

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Convex optimization in ML



Flexibility of "loss + penalty" framework

Minimize (*Loss* + λ *Penalty*)

Loss function	Penalty function	Resulting algorithm
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_2^2$	SVMs
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_2^2$	Ridge regression
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_1$	Lasso
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda_1\|\vec{w}\|_1 + \lambda_2\|\vec{w}\|_2^2$	Elastic net
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_1$	1-norm SVM

List-based learning data

SearchEngine1		SearchEngine2
MIDS		MIDS
1. + 5*	100% gain	1. - 2*
2. - 1*	50%	2. - 1*
3. + 3*	25%	3. + 4*
4. - 1*	12%	4. + 4*
5. + 2*		5. + 5*
Precision@5=3/5		Precision@5=3/5

5* rating system
5* required result
4*
3*
2*
1* (Spam/not relevant

https://en.wikipedia.org/wiki/Discounted_cumulative_gain

DCG

https://en.wikipedia.org/wiki/Discounted_cumulative_gain

Google Docs | Word2Vec: an introd | nbviewer.ipython.org | Bookmarks | (2) MIDS-MLS-2015 | Stanford Machine Le | Getting Started | Statistical Analysis H | eBay/Google 2013: E | Other Bookmarks

ort
e a book
load as PDF
able version
es
Add links

Cumulative Gain [edit]

Cumulative Gain (CG) is the predecessor of DCG and does not include the position of a result in the consideration of the usefulness of a result set. In this way, it is the sum of the graded relevance values of all results in a search result list. The CG at a particular rank position p is defined as:

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

Where rel_i is the graded relevance of the result at position i.

The value computed with the CG function is unaffected by changes in the ordering of search results. That is, moving a highly relevant document d_i above a higher ranked, less relevant, document d_j does not change the computed value for CG. Based on the two assumptions made above about the usefulness of search results, DCG is used in place of CG for a more accurate measure.

Discounted Cumulative Gain [edit]

The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result. The discounted CG accumulated at a particular rank position p is defined as:[2]

$$\mathrm{DCG_p} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i)}$$

Previously there has not been any theoretically sound justification for using a logarithmic reduction factor[3] other than the fact that it produces a smooth reduction. But Wang et al. (2013)[4] give theoretical guarantee for using the logarithmic reduction factor in NDCG. The authors show that for every pair of substantially different ranking functions, the NDCG can decide which one is better in a consistent manner.

An alternative formulation of DCG[5] places stronger emphasis on retrieving relevant documents:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

The latter formula is commonly used in industry including major web search companies[2] and data science competition platform such as Kaggle.[6]

In Croft, Metzler and Strohman (page 320, 2010), the authors mistakenly claim that these two formulations of DCG are the same when the relevance values of documents are binary; $rel_i \in \{0,1\}$. To see that they are not the same, let there be one relevant document and that relevant document is at rank 2. The first version of DCG equals 1 / log2(2) = 1. The second version of DCG equals 1 / log2(2+1) = 0.631. The way that the two formulations of DCG are the same for binary judgments is in the way gain in the numerator is calculated. For both formulations of DCG, binary relevance produces gain at rank i of 0 or 1. No matter the number of relevance grades, the two formulations differ in their discount of gain.

Note that Croft et al. (2010) and Burges et al. (2005) present the second DCG with a log of base e, while both versions of DCG above use a log of base 2. When computing NDCG with the second formulation of DCG, the base of the log does not matter, but the base of the log does affect the value of NDCG for the first formulation. Clearly, the base of the log affects the value of DCG in both formulations.

Normalized DCG [edit]

Search result lists vary in length depending on the query. Comparing a search engine's performance from one query to the next cannot be consistently achieved using DCG alone, so the cumulative gain at each position for a chosen value of p should be normalized across queries. This is done by sorting documents of a result list by relevance, producing the maximum possible DCG till position p, also called Ideal DCG (IDCG) till that position. For a query, the *normalized discounted cumulative gain*, or nDCG, is computed as:

DCG Optimization for Learning-to-Rank

- **Ifada, N. & Nayak, R. (2015), Do-Rank: DCG Optimization for Learning-to-Rank in Tag-Based Item Recommendation Systems., in Tru Cao; Ee-Peng Lim; Zhi-Hua Zhou; Tu-Bao Ho; David Wai-Lok Cheung & Hiroshi Motoda, ed., 'PAKDD (2)' , Springer, , pp. 510-521 .**

Paper is available here for download

https://www.dropbox.com/s/sxg2s2cfh7aezi6/Do-Rank-DCG-Based-Machine-Leanring.pdf?dl=0

Chapter

Advances in Knowledge Discovery and Data Mining

Volume 9078 of the series Lecture Notes in Computer Science pp 510-521

Date: 09 May 2015

Do-Rank: DCG Optimization for Learning-to-Rank in Tag-Based Item Recommendation Systems

Noor Ifada , Richi Nayak

Affiliated with
School of Electrical Engineering and Computer Science, Queensland University of Technology

Buy eBook

$29.95 / €24.95 / £19.95 *

$99.00 / €74.96 / £62.99*

Get Access

* Final gross prices may vary according to local VAT.

Reference tools

Export citation

Add to Papers

Other actions

» About this Book

» Reprints and Permissions

Share

Abstract

Discounted Cumulative Gain (DCG) is a well-known ranking evaluation measure for models built with multiple relevance graded data. By handling tagging data used in recommendation systems as an ordinal relevance set of $\{negative, null, positive\}$, we propose to build a DCG based recommendation model. We present an efficient and novel learning-to-rank method by optimizing DCG for a recommendation model using the tagging data interpretation scheme. Evaluating the proposed method on real-world datasets, we demonstrate that the method is scalable and outperforms the benchmarking methods by generating a quality top-*N* item recommendation list.

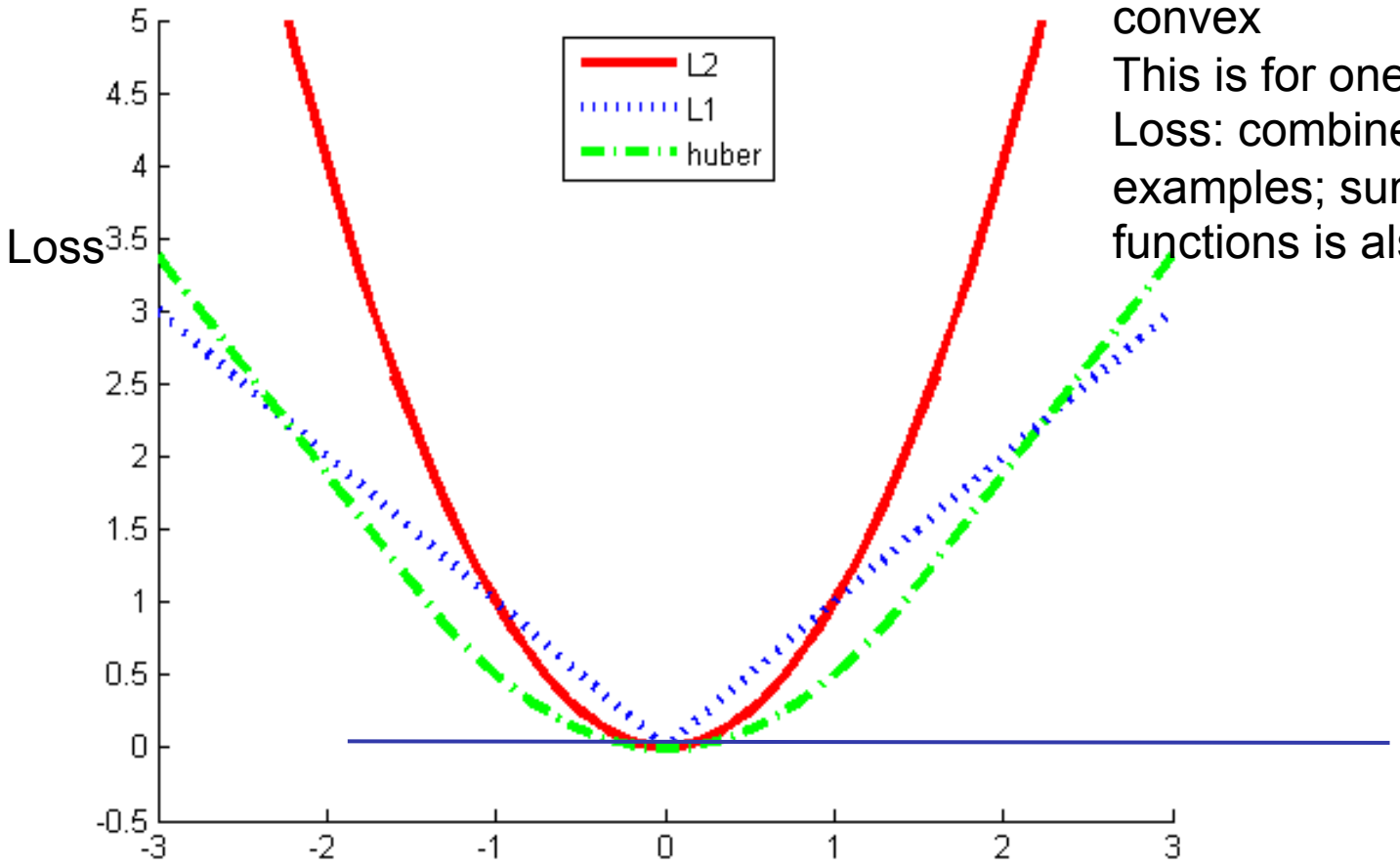
Keywords

Tagging data – Tag-based item recommendation – Discounted cumulative gain – Top-*N* recommendation

Machine Learning Objective Function: regression loss function

Model: $y = \theta^T x$

Goal: $\theta^* = \min_\theta \sum_{i=1}^{m} (\theta^T x_i - y_i)^2$



Regression loss functions: all are convex

This is for one example in 1.7Dim; Loss: combine over all training examples; sum of such convex functions is also convex

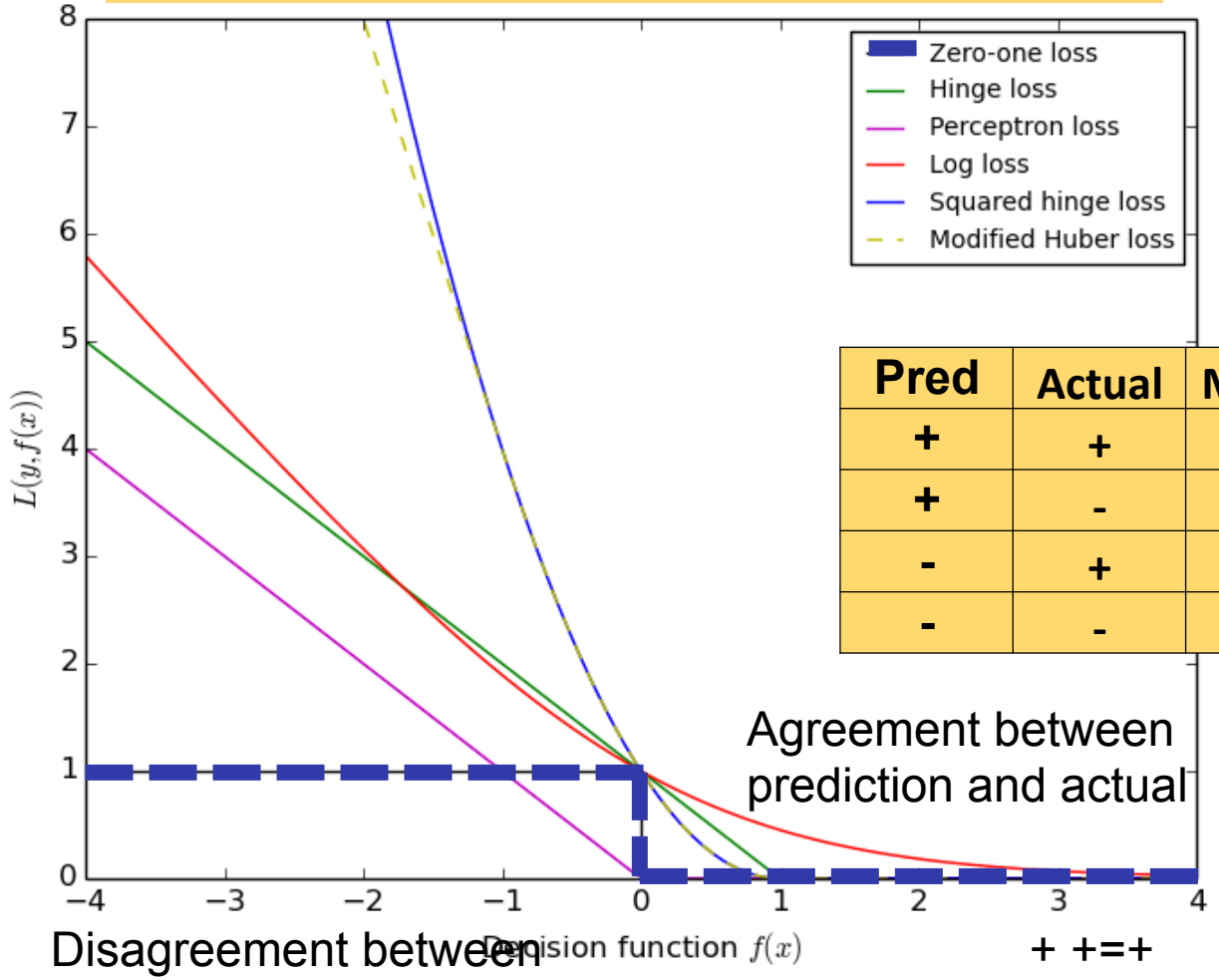
Flexibility of "loss + penalty" framework

Minimize (*Loss* + λ *Penalty*)

Loss function	Penalty function	Resulting algorithm
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_2^2$	SVMs
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_2^2$	Ridge regression
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_1$	Lasso
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda_1\|\vec{w}\|_1 + \lambda_2\|\vec{w}\|_2^2$	Elastic net
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_1$	1-norm SVM

Machine Learning Objective Function: classification

Classification loss functions



Loss

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Agreement between prediction and actual

+ +=+

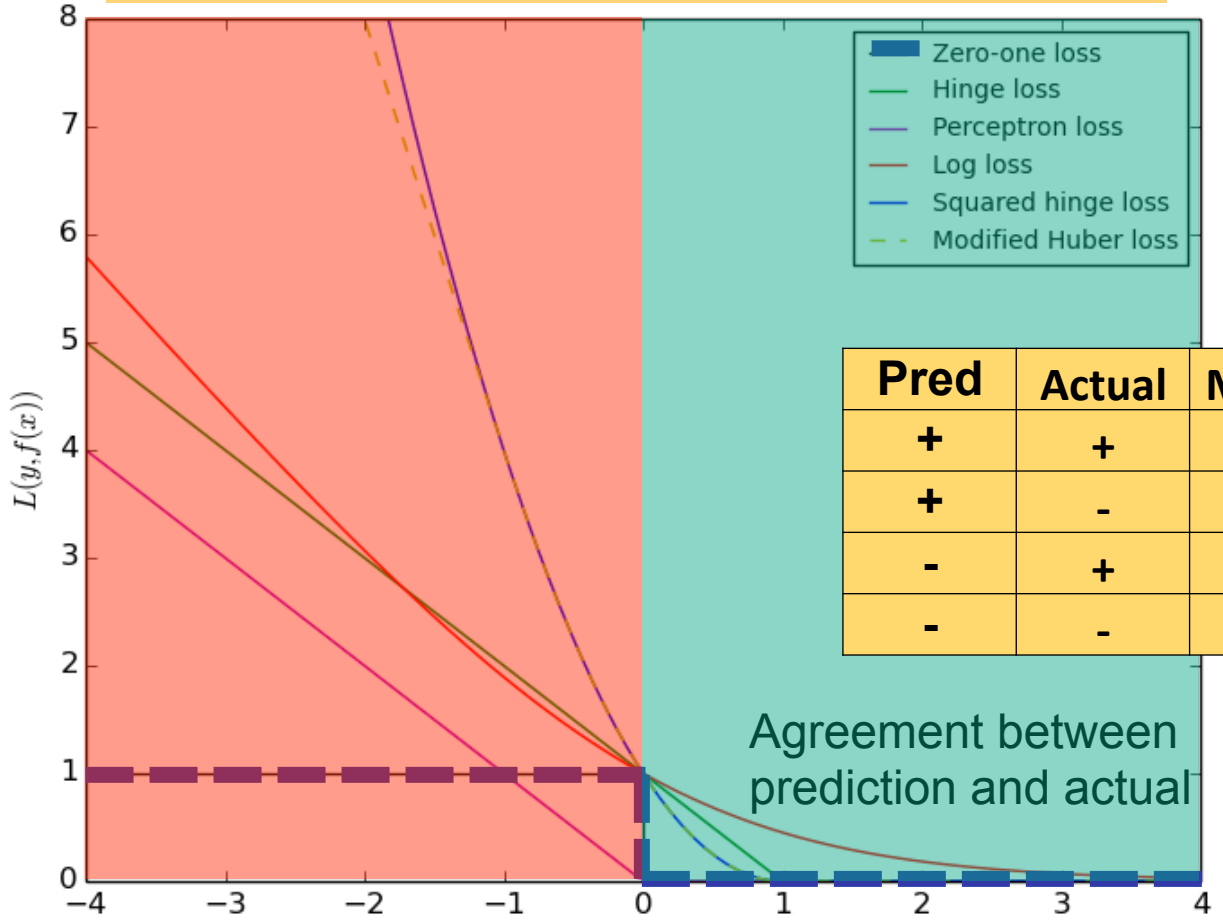
Disagreement between prediction and actual

y ✕ f(X)

Margin

Machine Learning Objective Function: classification

Classification loss functions



Loss

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Disagreement between prediction and actual

y ✕ f(X)

+ +=+

Margin

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term ($R(w)$ expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \tag{14.1}$$

$$m_i = y^{(i)} f_w(x^{(i)}) \tag{14.2}$$

$$y^{(i)} \in \{-1, 1\} \tag{14.3}$$

$$f_w(x^{(i)}) = w^T x^{(i)} \tag{14.4}$$

Empirical Risk Minimization

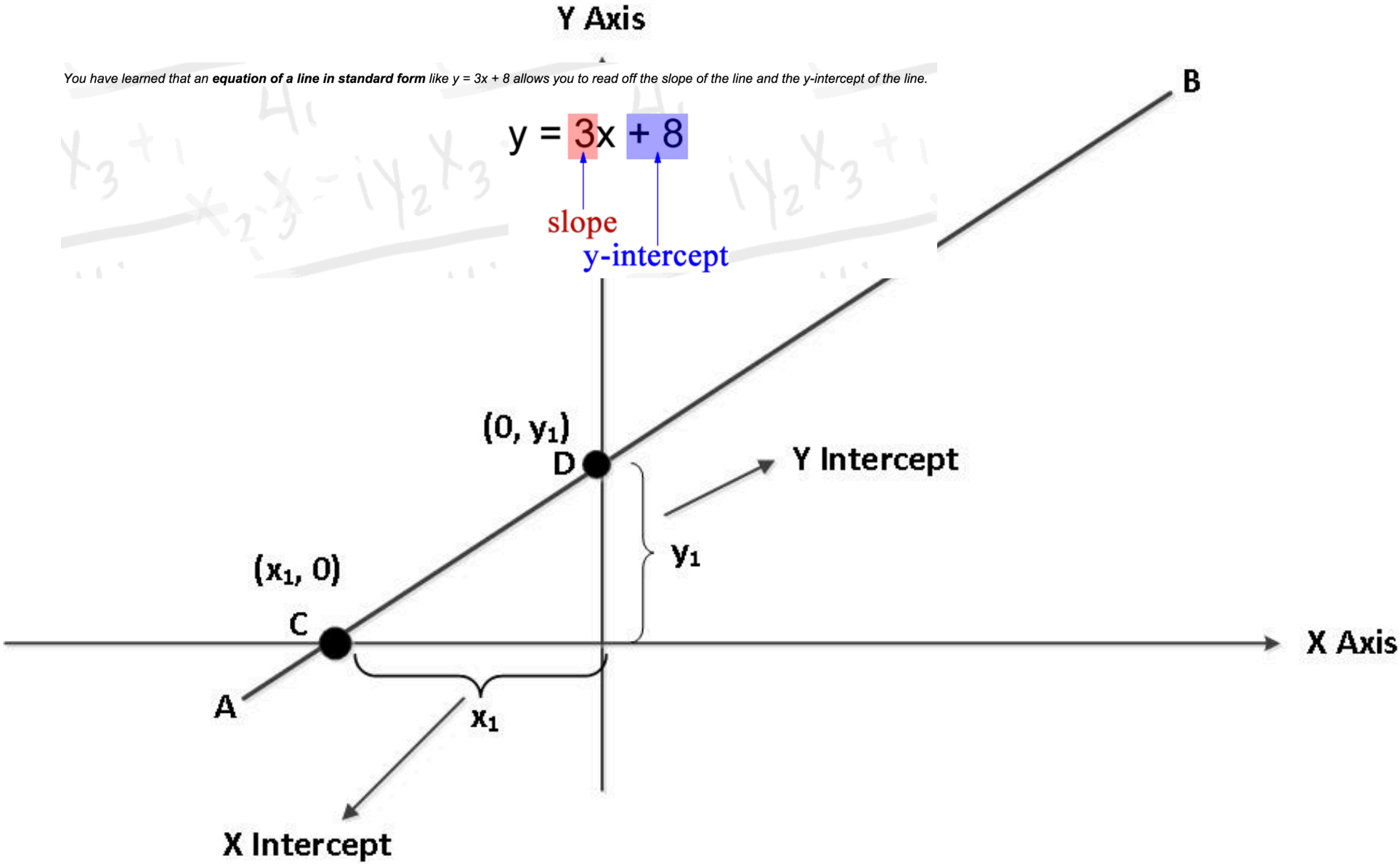
- Provides a criterion to decide on h:

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} loss(\boldsymbol{x}_i, \boldsymbol{y}_i; h)$$

- Background preferences over h can be included in **regularized** empirical risk minimization:

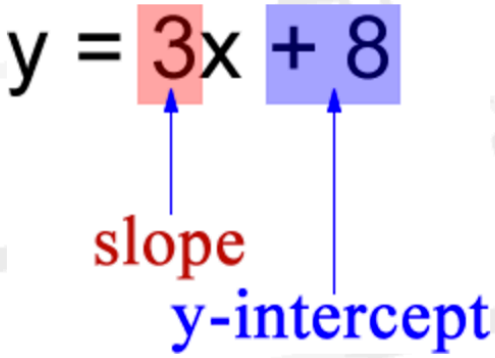
$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} loss(\boldsymbol{x}_i, \boldsymbol{y}_i; h) + R(h)$$

*You have learned that an **equation of a line in standard form** like y = 3x + 8 allows you to read off the slope of the line and the y-intercept of the line.*



Slope and y-intercept

*You have learned that an **equation of a line in standard form** like y = 3x + 8 allows you to read off the slope of the line and the y-intercept of the line.*

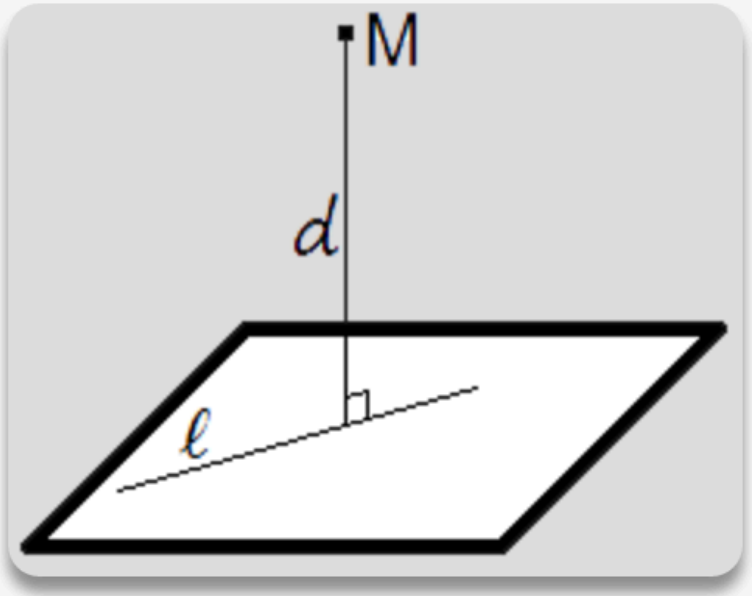


Perpendicular distance from a point to a plane

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$



Quiz

- **How far from the origin is the line 3x + 4y = 10?**
  - (a) 1.5 units
  - (b) 2 units
  - (c) 2.5 units
  - (d) 2.75 units
  - (e) None of the above

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

Quiz

- **How far from the origin is the line 3x + 4y = 10?**
  - (a) 1.5 units
  - (b) 2 units
  - (c) 2.5 units
  - (d) 2.75 units
  - (e) None of the above

Shortcut: Pdist from (0,0) to 3x + 4y = 10

$$\frac{(3(0) + 4(0) - 10)}{\text{SQRT}((3 * 3) + (4*4) + (0*0))} = \frac{-10}{5} = -2$$

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

How far from the origin is the line 3x + 4y = 10?

- How far from the origin is the line 3x + 4y = 10?
- (a) 1.5 units (b) 2 units (c) 2.5 units (d) 2.75 units (e) None of the above

Another way to ask this question is:
What is the perpendicular distance from the origin to this line?

$$3x+4y=10$$

$$4y=-3x+10$$

$$y=\left(-\frac{3}{4}\right)\cdot x+\frac{5}{2}$$

Any line perpendicular to this line will have slope = $-\left(\frac{1}{-\frac{3}{4}}\right)=\frac{4}{3}$

The line through origin with this slope is:

$$y=\left(\frac{4}{3}\right)\cdot x$$

Now the question is: What is the point where this line intersects the given line?

By substitution

$$3x+4\cdot\left(\frac{4}{3}\right)\cdot x=10$$

$$9x+16x=30$$

$$25x=30$$

$$x=\frac{6}{5}$$

and, substituting:

$$y=\left(\frac{4}{3}\right)\cdot\left(\frac{6}{5}\right)$$

$$y=\frac{8}{5}$$

They intersect at
x = 6/5, y = 8/5
the distance from the (0,0) to (6/5,8/5) is
d=sqrt(y-y)^2+(x-x)^2
d=sqrt((6/5)^2+(8/5)^2)
d=2

The distance formula is:

$$d=\sqrt{\left(\frac{6}{5}\right)^2+\left(\frac{8}{5}\right)^2}$$

$$d=\sqrt{\frac{36}{25}+\frac{64}{25}}$$

$$d=\sqrt{\frac{100}{25}}$$

$$d=\sqrt{4}$$

$$d=2$$

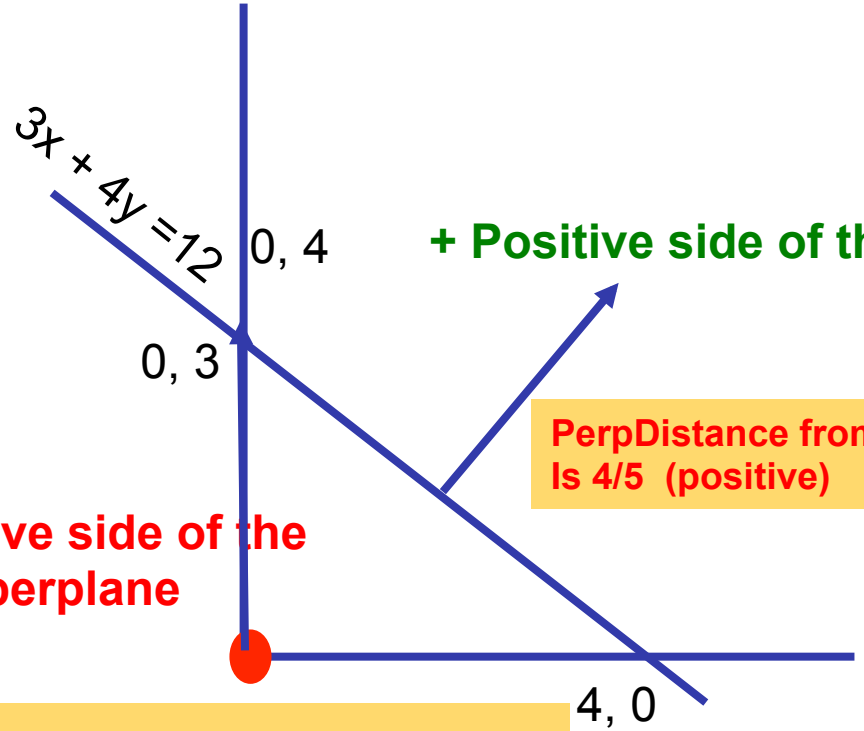
the distance is (b) 2 units

Shortcut:
(3(0) + 4(0) -10)
---------- = -10/5 =-2
SQTR((3 * 3) (4*4))

HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2



+ Positive side of the hyperplane

PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

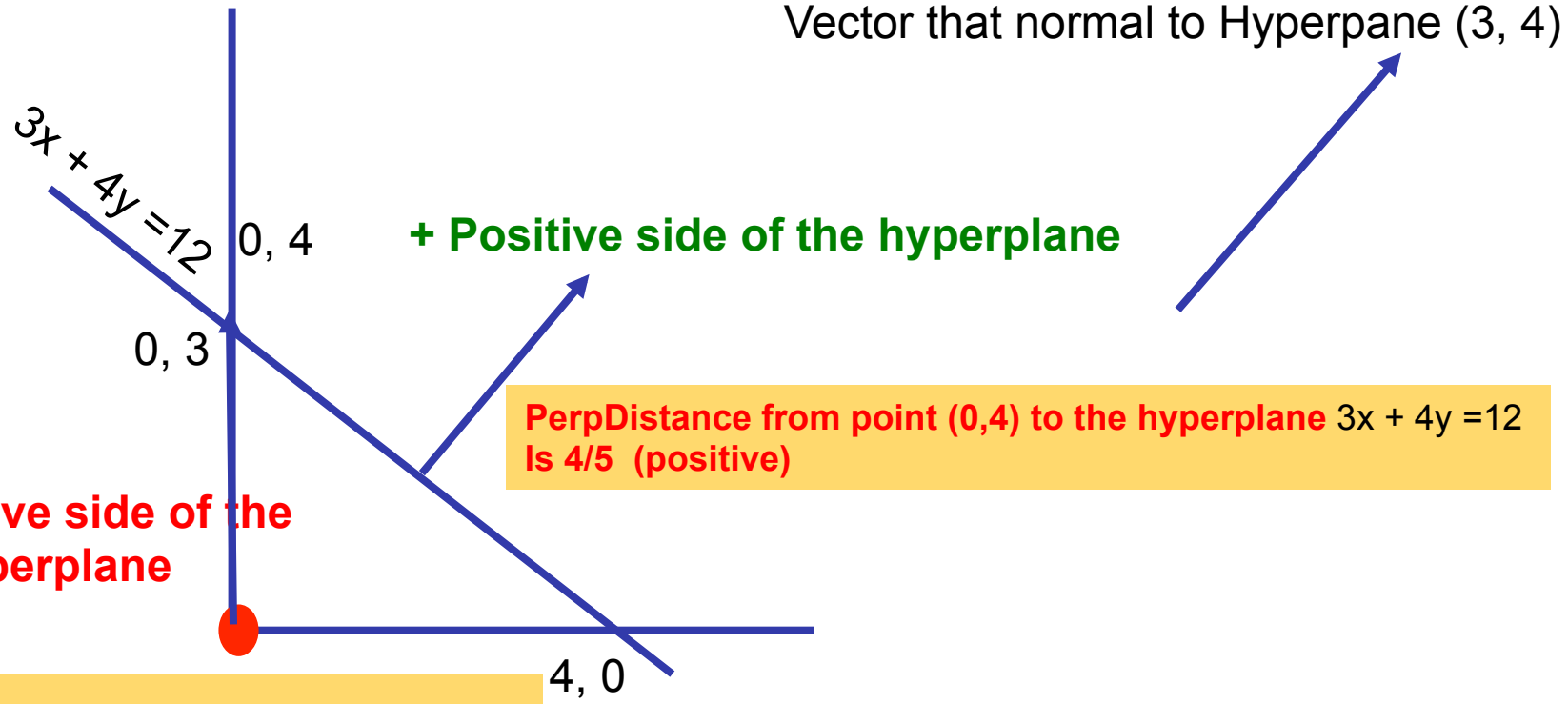
- Negative side of the hyperplane

Shortcut:
(3(0) + 4(0) -12)
---------- = -12/5 = -2.4
SQTR((3 * 3) + (4*4))

HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2



Shortcut:
(3(0) + 4(0) -12)
---------- = -12/5 = -2.4
SQTR((3 * 3) + (4*4))

Distance is signed

- **+ distance means the point is the same side where the normal is pointing**

Note that the distance formula looks like inserting P_2 into the plane equation, then dividing by the length of the normal vector. For example, the distance from a point (-1, -2, -3) to a plane x + 2y + 2z - 6 = 0 is;

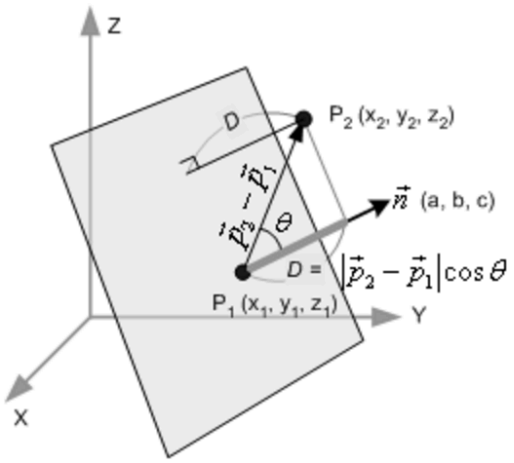
$$D = \frac{1x + 2y + 2z - 6}{\sqrt{1^2 + 2^2 + 2^2}}, \qquad P_2 = (-1, -2, -3)$$

$$= \frac{1(-1) + 2(-2) + 2(-3) - 6}{\sqrt{1^2 + 2^2 + 2^2}}$$

$$= \frac{-1 - 4 - 6 - 6}{3} = \frac{-17}{3}$$

Notice this distance is signed; can be negative value. It is useful to determine the direction of the point. For example, if the distance is positive, the point is in the same side where the normal is pointing to. And, a negative distance means the point is in opposite side.

Distance from a Point



Distance between Plane and Point

The shortest distance from an arbitrary point P_2 to a plane can be calculated by the dot product of two vectors $\vec{n}$ and $(\vec{p}_2 - \vec{p}_1)$, projecting the vector $(\vec{p}_2 - \vec{p}_1)$ to the normal vector $\vec{n}$ of the plane.

The distance *D* between a plane $ax + by + cz + d = 0$ and a point P_2 becomes;

$$D = |\vec{p}_2 - \vec{p}_1| \cos\theta = \frac{\vec{n} \cdot (\vec{p}_2 - \vec{p}_1)}{|\vec{n}|} = \frac{\vec{n} \cdot (\vec{p}_2 - \vec{p}_1)}{\sqrt{a^2 + b^2 + c^2}}$$

$$\because \vec{n} \cdot (\vec{p}_2 - \vec{p}_1) = |\vec{n}||\vec{p}_2 - \vec{p}_1| \cos\theta$$

The numerator part of the above equation, $\vec{n} \cdot (\vec{p}_2 - \vec{p}_1)$ is expanded;

$$\begin{aligned}\vec{n} \cdot (\vec{p}_2 - \vec{p}_1) &= (a, b, c) \cdot (x_2 - x_1, y_2 - y_1, z_2 - z_1) \\ &= ax_2 + by_2 + cz_2 - (ax_1 + by_1 + cz_1)\end{aligned}$$

Finally, we put it to the previous equation to complete the distance formula;

$$D = \frac{ax_2 + by_2 + cz_2 - (ax_1 + by_1 + cz_1)}{\sqrt{a^2 + b^2 + c^2}}$$

Note that the distance formula looks like inserting P_2 into the plane equation, then dividing by the length of the normal vector. For example, the distance from a point (-1, -2, -3) to a plane x + 2y + 2z - 6 = 0 is;

$$\begin{aligned}D &= \frac{1x + 2y + 2z - 6}{\sqrt{1^2 + 2^2 + 2^2}}, \qquad P_2 = (-1, -2, -3) \\ &= \frac{1(-1) + 2(-2) + 2(-3) - 6}{\sqrt{1^2 + 2^2 + 2^2}} \\ &= \frac{-1 - 4 - 6 - 6}{3} = \frac{-17}{3}\end{aligned}$$

Notice this distance is signed; can be negative value. It is useful to determine the direction of the point. For example, if the distance is positive, the point is in the same side where the normal is pointing to. And, a negative distance means the point is in opposite side.

Basic operation on vectors in $\mathbb{E}^n$

1. Multiplication by a scalar

Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$ and a scalar c

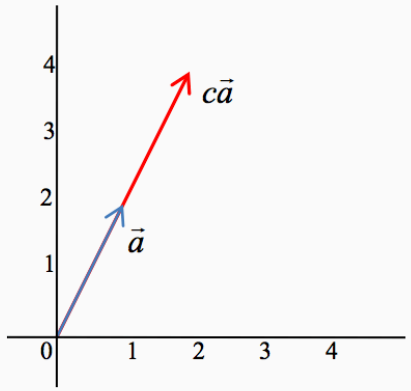
Define: $c\vec{a} = (ca_1, ca_2, \ldots, ca_n)$

When you multiply a vector by a scalar, you "stretch" it in the same or opposite direction depending on whether the scalar is positive or negative.

$\vec{a} = (1,2)$

$c = 2$

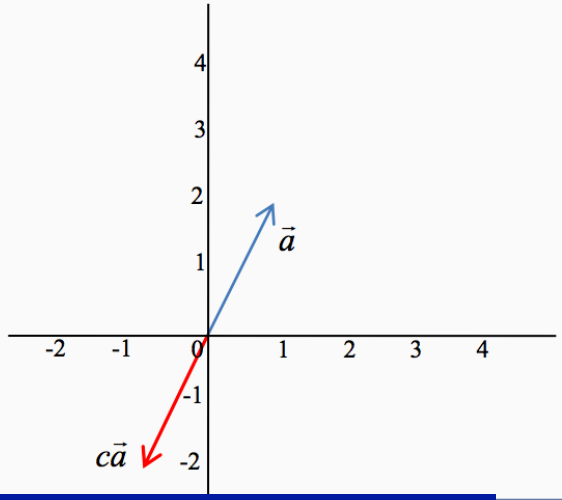
$\vec{c}a = (2,4)$



$\vec{a} = (1,2)$

$c = -1$

$\vec{c}a = (-1,-2)$



Adopted from http://www.med.nyu.edu/chibi/sites/default/files/chibi/Final.pdf

Basic operation on vectors in $\mathbb{E}^n$

2. Addition

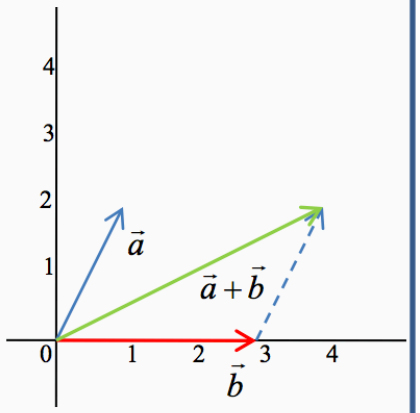
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} + \vec{b} = (4,2)$



Recall addition of forces in classical mechanics.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

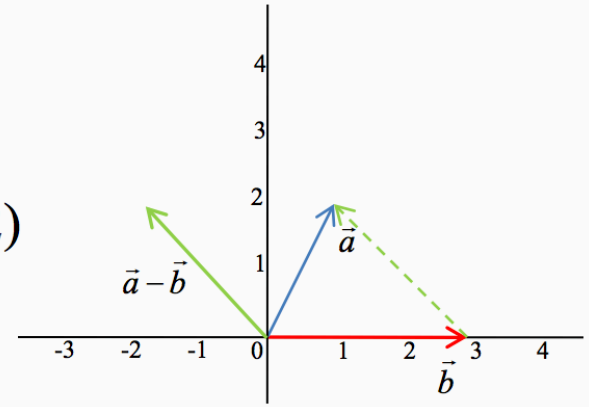
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

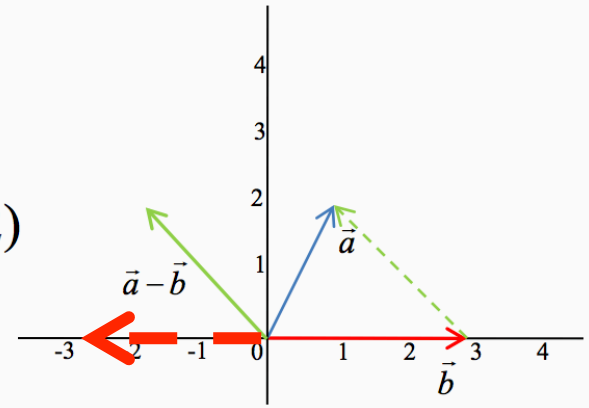
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

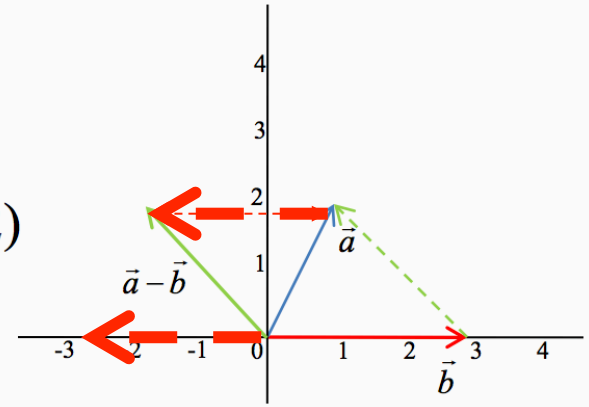
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

4. Euclidian length or L2-norm

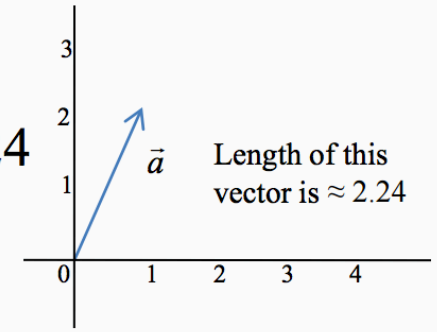
Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$

Define the L2-norm: $\|\vec{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

We often denote the L2-norm without subscript, i.e. $\|\vec{a}\|$

$\vec{a} = (1,2)$

$\|\vec{a}\|_2 = \sqrt{5} \approx 2.24$



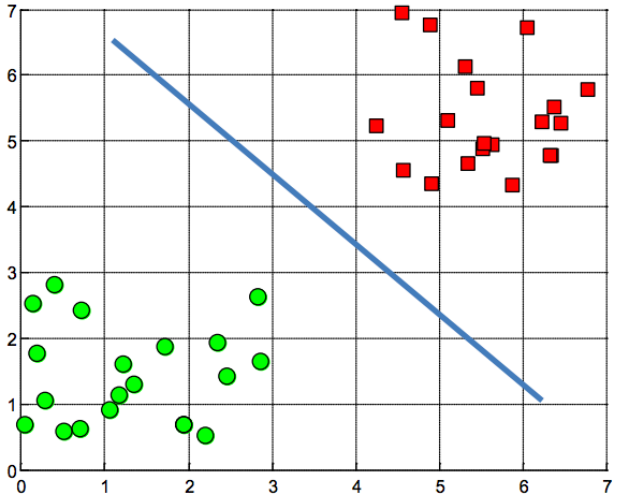
L2-norm is a typical way to measure length of a vector; other methods to measure length also exist.

Adopted from http://www.med.nyu.edu/chibi/sites/default/files/chibi/Final.pdf

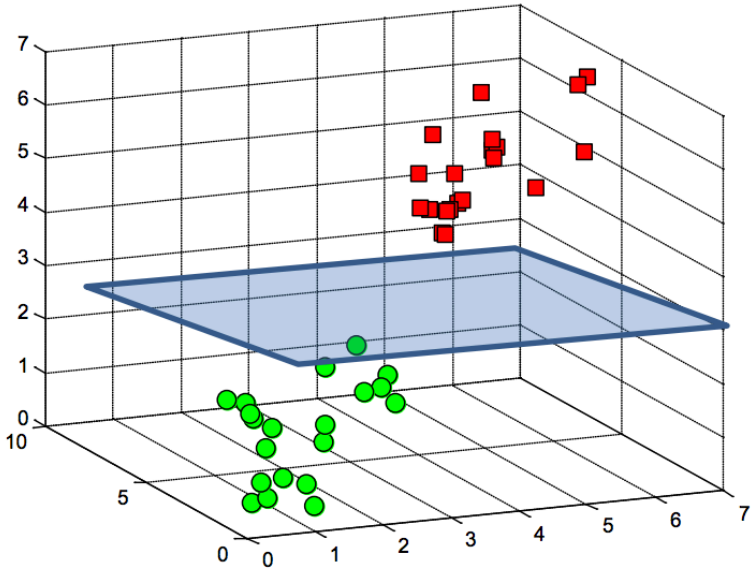
Hyperplanes as decision surfaces

- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

A hyperplane in $\mathbb{E}^2$ is a line



A hyperplane in $\mathbb{E}^3$ is a plane



A hyperplane in $\mathbb{E}^n$ is an n-1 dimensional subspace

Equation of a hyperplane

First we show with show the definition of hyperplane by an interactive demonstration.

Click here for demo to begin

or go to http://www.dsl-lab.org/svm_tutorial/planedemo.html

Source: http://www.math.umn.edu/~nykamp/

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term ($R(w)$ expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$f_w \quad (14.3)$$

Focus only on the loss term
E.g., Linear regression, Soft SVMs

Flexibility of "loss + penalty" framework

Minimize (*Loss* + λ *Penalty*)

Loss function	Penalty function	Resulting algorithm
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_2^2$	SVMs
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_2^2$	Ridge regression
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda\|\vec{w}\|_1$	Lasso
Mean squared error: $\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2$	$\lambda_1\|\vec{w}\|_1 + \lambda_2\|\vec{w}\|_2^2$	Elastic net
Hinge loss: $\sum_{i=1}^{N}[1-y_i f(\vec{x}_i)]_+$	$\lambda\|\vec{w}\|_1$	1-norm SVM

Augmented Representations and Classification Rule

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	-1
2					+1
…	…	…	…	…	…
L (aka m)	0	4	…	8	-1

Augmented Dataset

Instance\Attr	x_0	x_1	x_2	…	x_n	y
1	1	3	0	..	7	-1
2	1					+1
…	1	…	…	…	…	…
L (aka m)	1	0	4	…	8	-1

Hyperplane as an Augmented weight vector

$$W = \begin{bmatrix} w_0 \\ w_1 \\ .. \\ w_n \end{bmatrix} = \begin{bmatrix} b \\ w_1 \\ .. \\ w_n \end{bmatrix}$$

Augmented Data vector

$$X = \begin{bmatrix} x_1 \\ .. \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ .. \\ x_n \end{bmatrix}$$

$$Class\ (X) = sign(\langle W, X \rangle + b)$$

Classification rule simplifies

$$Class\ (X) = sign(\langle W, X \rangle)$$

Version Space

- A version space in concept learning or induction is the subset of all hypotheses that are consistent with the observed training examples (Mitchell 1997).
- This set contains all hypotheses that have not been eliminated as a result of being in conflict with observed data.

Augmented feature vector PLUS an Augmented weight vector

From W → A

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i \tag{9}$$

where we set $x_0 = 1$. Thus we can write

$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}, \tag{10}$$

and $\mathbf{y}$ is sometimes called an *augmented feature vector*. Likewise, an *augmented weight vector* can be written as:

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}. \tag{11}$$

a = augmented weight vector
y is augmented sample

From X-space (d-dimensions) to the augmented y-space (d+1 dims)

- This mapping from **d** -dimensional **x** -space to (**d**+1)-dimensional **y** -space is mathematically trivial but nonetheless quite convenient.
- The addition of a constant component to **x** preserves all distance relationships among samples.
- The resulting **y** vectors all lie in a **d** -dimensional subspace, which is the **x** -space itself.
- The hyperplane decision surface ˆ**H** defined by $\mathbf{a}^t\mathbf{y}$ = 0 passes through the origin in **y** -space, even though the corresponding hyperplane **H** can be in any position in **x** -space

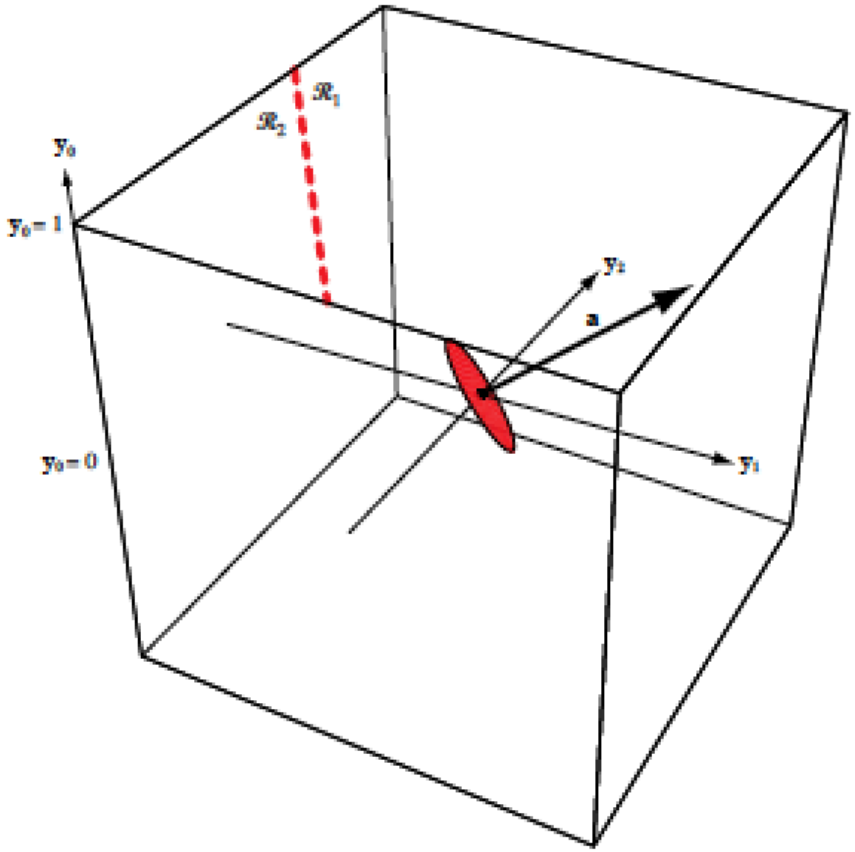


Figure 5.7: A three-dimensional augmented feature space $\mathbf{y}$ and augmented weight vector $\mathbf{a}$ (at the origin). The set of points for which $\mathbf{a}^t\mathbf{y} = 0$ is a plane (or more generally, a hyperplane) perpendicular to $\mathbf{a}$ and passing through the origin of $\mathbf{y}$-space, as indicated by the red disk. Such a plane need not pass through the origin of the two-dimensional $\mathbf{x}$-space at the top, of course, as shown by the dashed line. Thus there exists an augmented weight vector $\mathbf{a}$ that will lead to any straight decision line in $\mathbf{x}$-space.

Find a single weight vector a

- By using this mapping we reduce the problem of finding a weight vector w and a threshold weight **w**0 to the problem of finding a single weight vector a

Augmented Vector

From W → A

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i \tag{9}$$

where we set $x_0 = 1$. Thus we can write

$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}, \tag{10}$$

AUGMENTED VECTOR

and $\mathbf{y}$ is sometimes called an *augmented feature vector*. Likewise, an *augmented weight vector* can be written as:

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}. \tag{11}$$

This mapping from d-dimensional $\mathbf{x}$-space to $(d+1)$-dimensional $\mathbf{y}$-space is mathematically trivial but nonetheless quite convenient. The addition of a constant component to $\mathbf{x}$ preserves all distance relationships among samples. The resulting $\mathbf{y}$ vectors all lie in a d-dimensional subspace, which is the $\mathbf{x}$-space itself. The hyperplane decision surface $\hat{H}$ defined by $\mathbf{a}^t\mathbf{y} = 0$ passes through the origin in $\mathbf{y}$-space, even though the corresponding hyperplane H can be in any position in $\mathbf{x}$-space. The distance from $\mathbf{y}$ to $\hat{H}$ is given by $|\mathbf{a}^t\mathbf{y}|/\|\mathbf{a}\|$, or $|g(\mathbf{x})|/\|\mathbf{a}\|$. Since $\|\mathbf{a}\| > \|\mathbf{w}\|$, this distance is less

Assume Linearly Separable (for now)

- Suppose now that we have a set of n samples **y1**, ..., **yn** , some labelled ω**1** and some labelled ω**2** . We want to use these samples to determine the weights **a** in a linear discriminant function g (**x**) = $\mathbf{a}^t\mathbf{y}$.
- Suppose we have reason to believe that there exists a solution for which the probability of error is very low.
- Then a reasonable approach is to look for a weight vector that classifies all of the samples correctly. If such a weight vector exists, the samples are said to be linearly separable .

+1 and -1 as class labels normalization of each training example

- A sample **y**i is classified correctlyif $\mathbf{a}^t\mathbf{y}i > 0$ and **y**i is labelled ω1 or if $\mathbf{a}^t\mathbf{y}i < 0$ separable and **y**i is labelled ω2 .
- This suggests a "normalization" that simplifies the treatment of the two-category case, viz., **the replacement of all samples labelled ω2 by their negatives.**
- With this "normalization" we can forget the labels and look for a weight vector **a** such that $\mathbf{a}^t\mathbf{y}i > 0$ for all of the samples. Such a weight vector is called a separating vector or more generallya solution vector

Multiply each negative example by -1

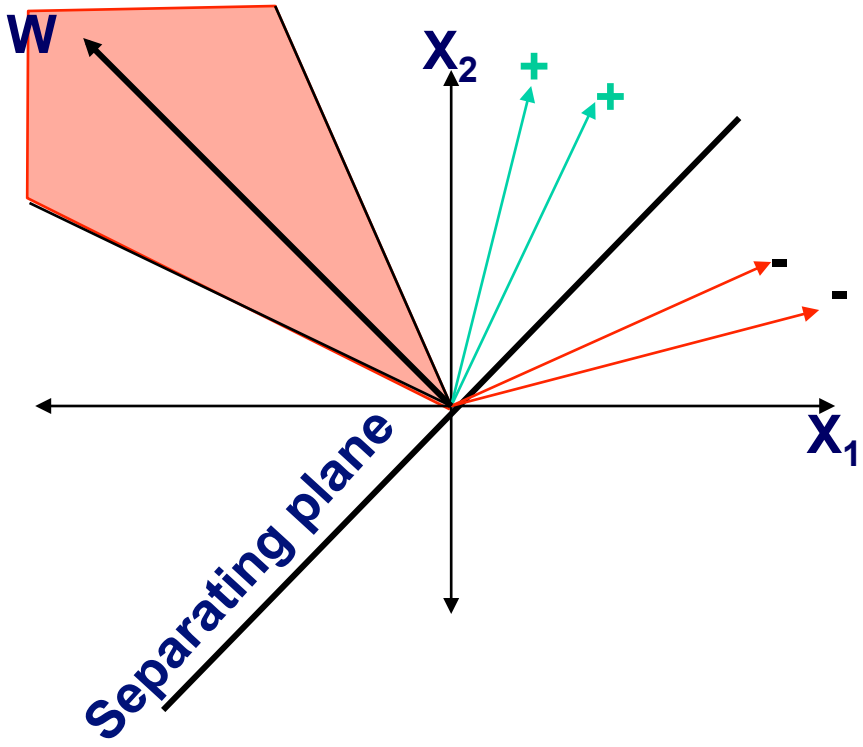
-1 X y_i

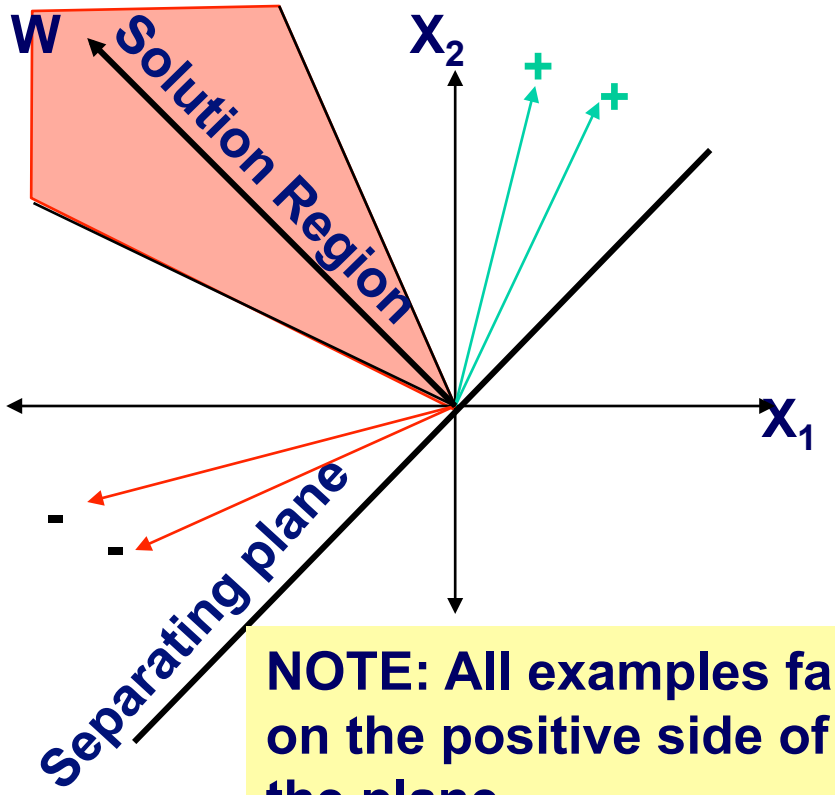
Label Normalization

Label Normalized Vector

$$Xy = \begin{bmatrix} x_1 \\ x_2 \\ .. \\ x_n \end{bmatrix} y = \begin{bmatrix} x_1 y \\ x_2 y \\ .. \\ x_n y \end{bmatrix}$$

Solution Region





NOTE: All examples fall on the positive side of the plane

Normalization "-1" training examples

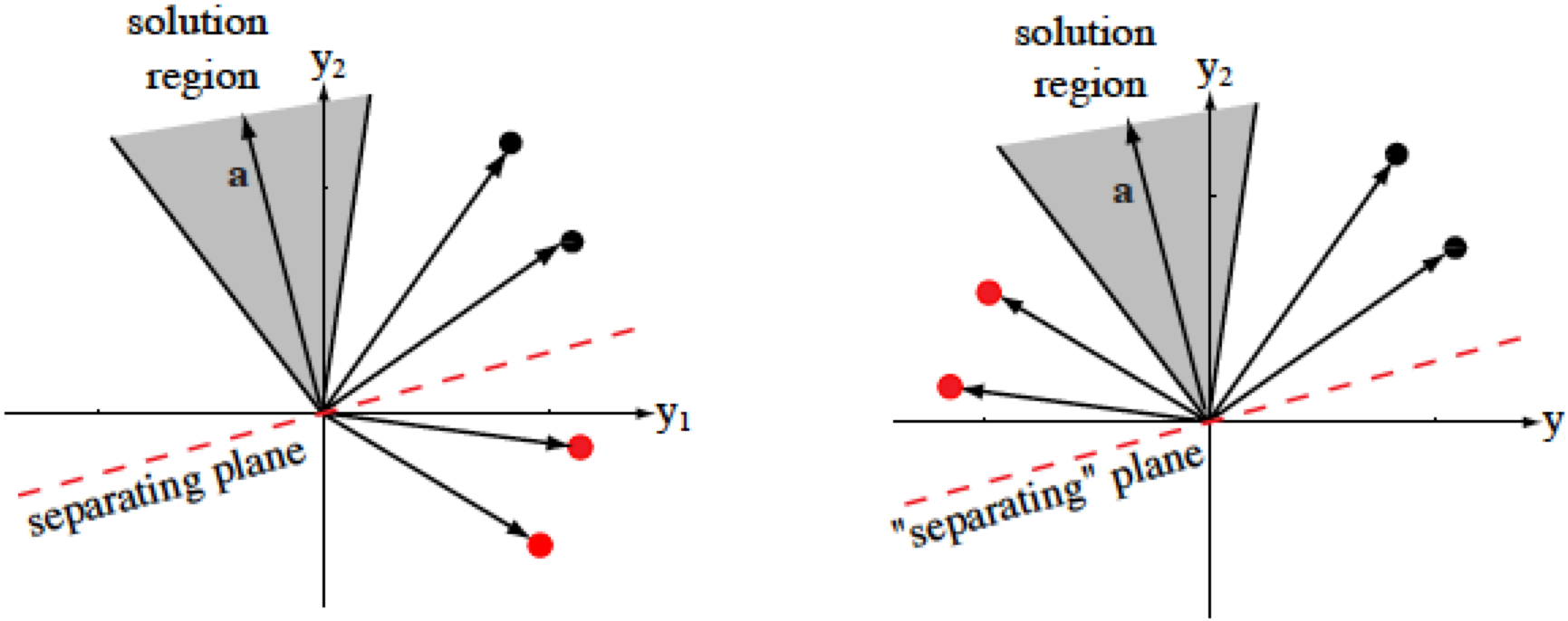


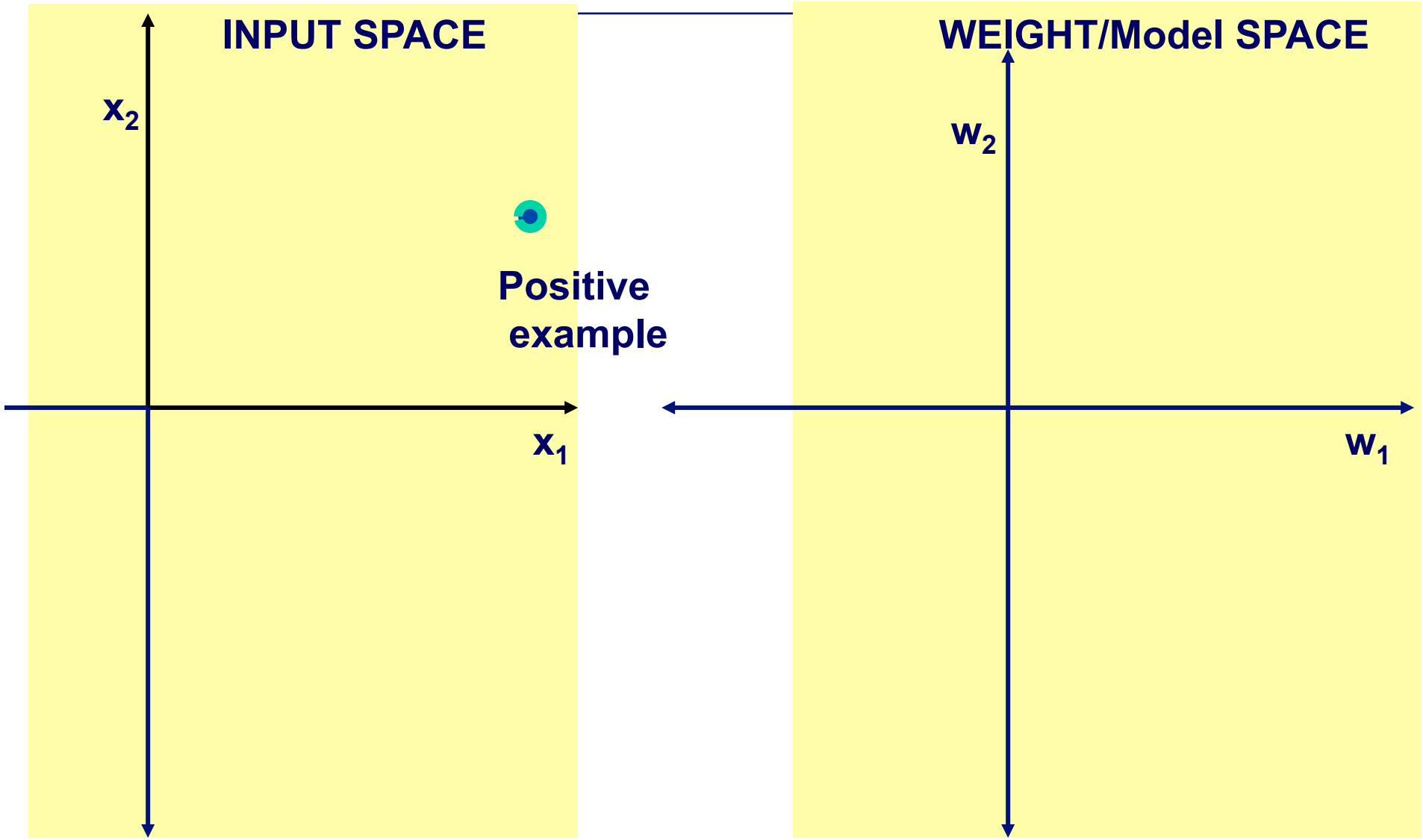
Figure 5.8: Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized" — i.e., changed in sign. Now the solution vector leads to a plane that places all "normalized" points on the same side.

Multiply each negative example by -1, i.e., -1 X y

Solution region

- The weight vector a can be thought of as specifying a point in weight space . Each vector sample yi places a constraint on the possible location of a solution vector.
- The equation $a^t yi = 0$ defines a hyperplane through the origin of weight space having yi as a normal vector. The solution vector — if it exists — must be on the positive side of every hyperplane
- Thus, a solution vector must lie in the intersection of n halfspaces; indeed any vector in this region is a solution vector.
- The corresponding region solution is called the solution region , and should not be confused with the decision region in region feature space corresponding to any particular category.

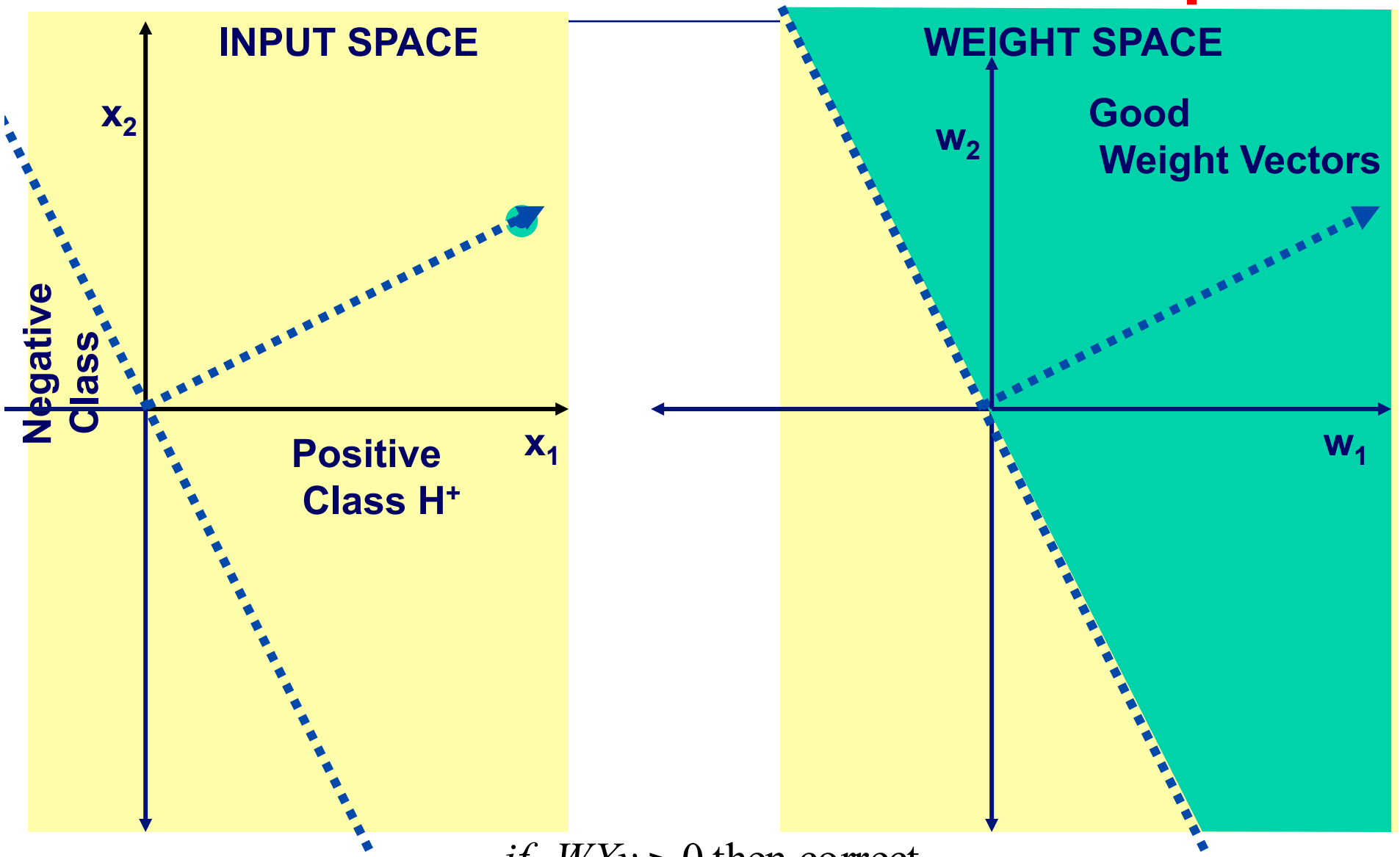
Positive Class Version Space



MODEL: *if* $WXy \geq 0$ then correct

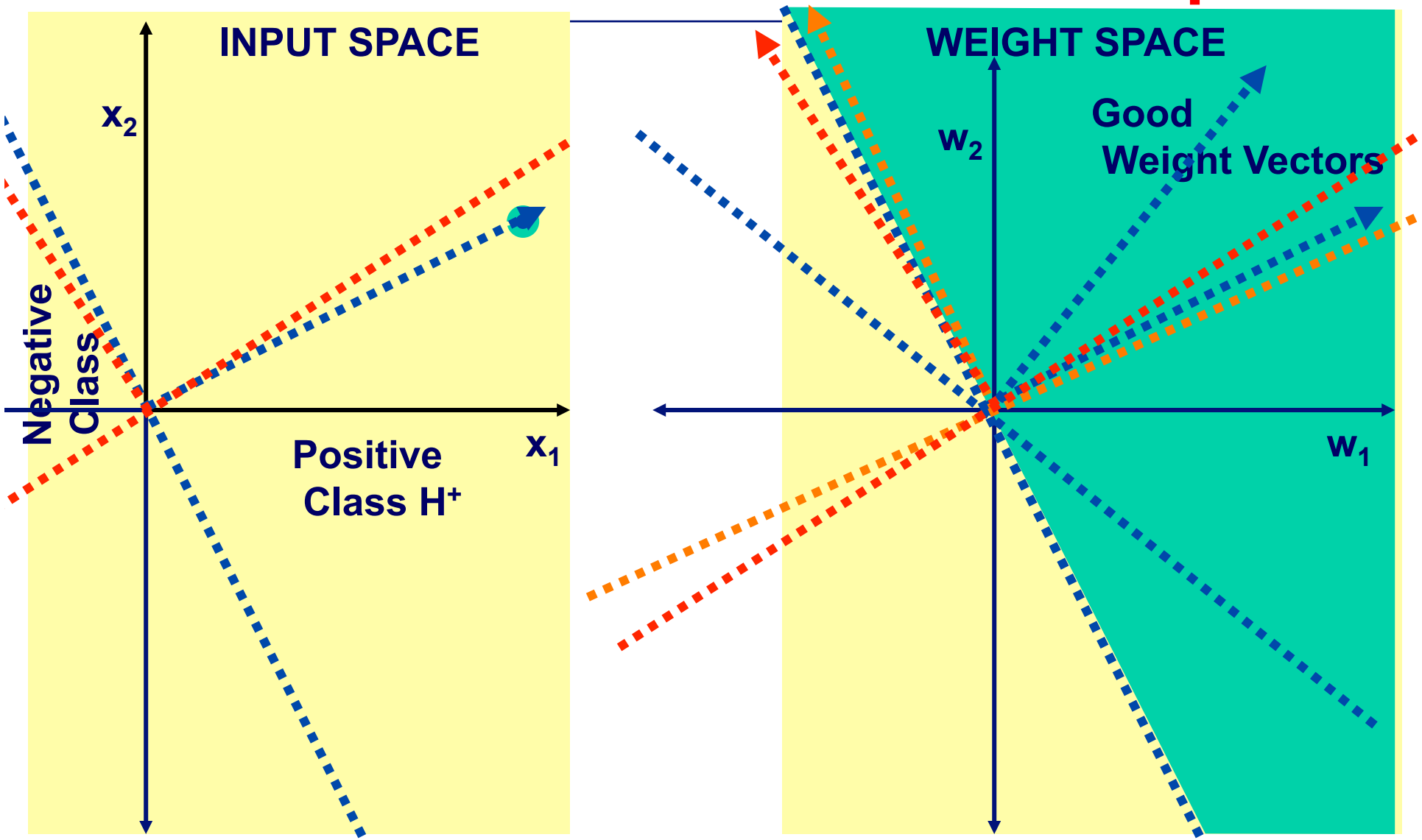
Positive Margin

Positive Class Version Space



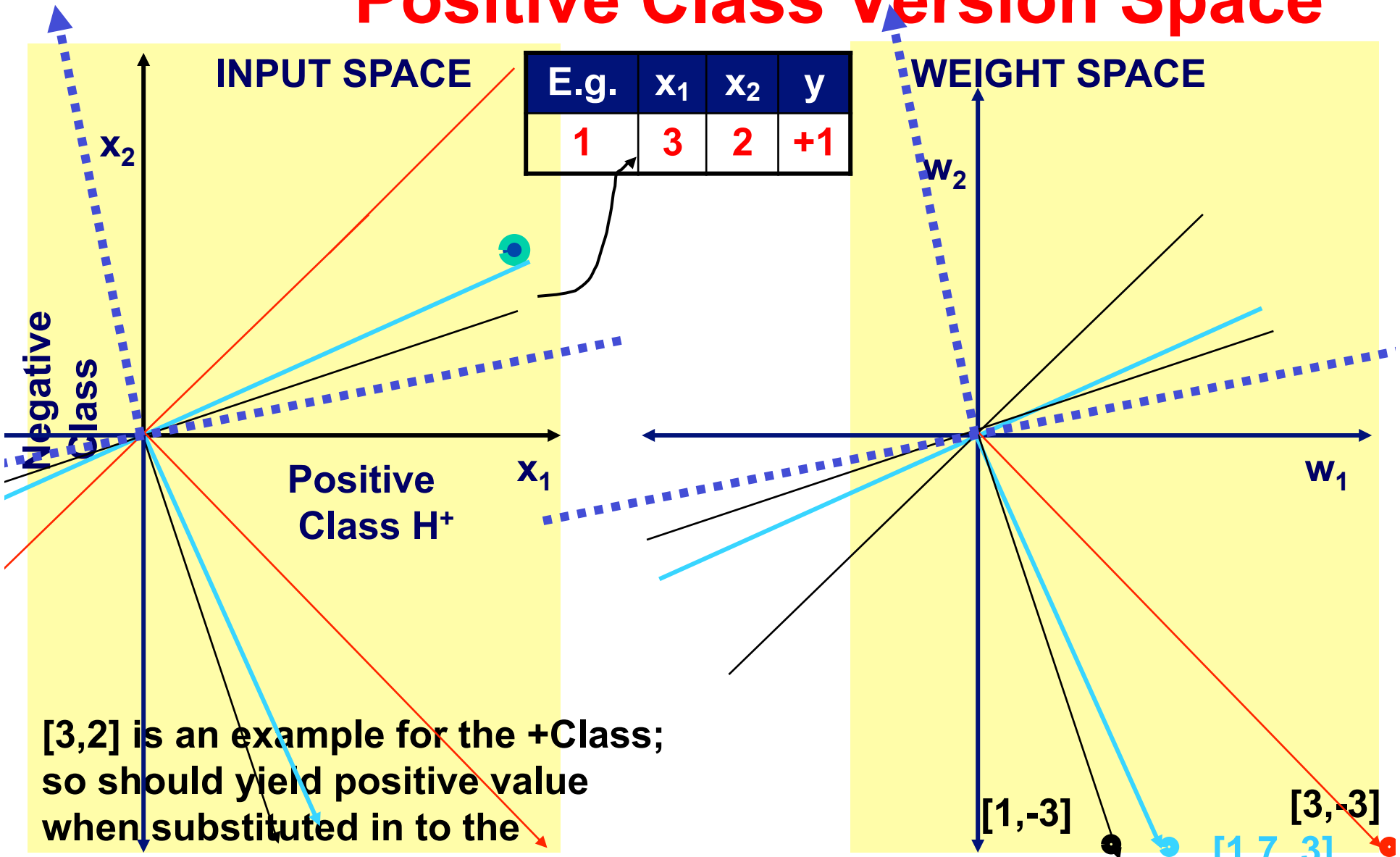
$$if \;\; WXy \geq 0 \text{ then correct}$$

Positive Class Version Space

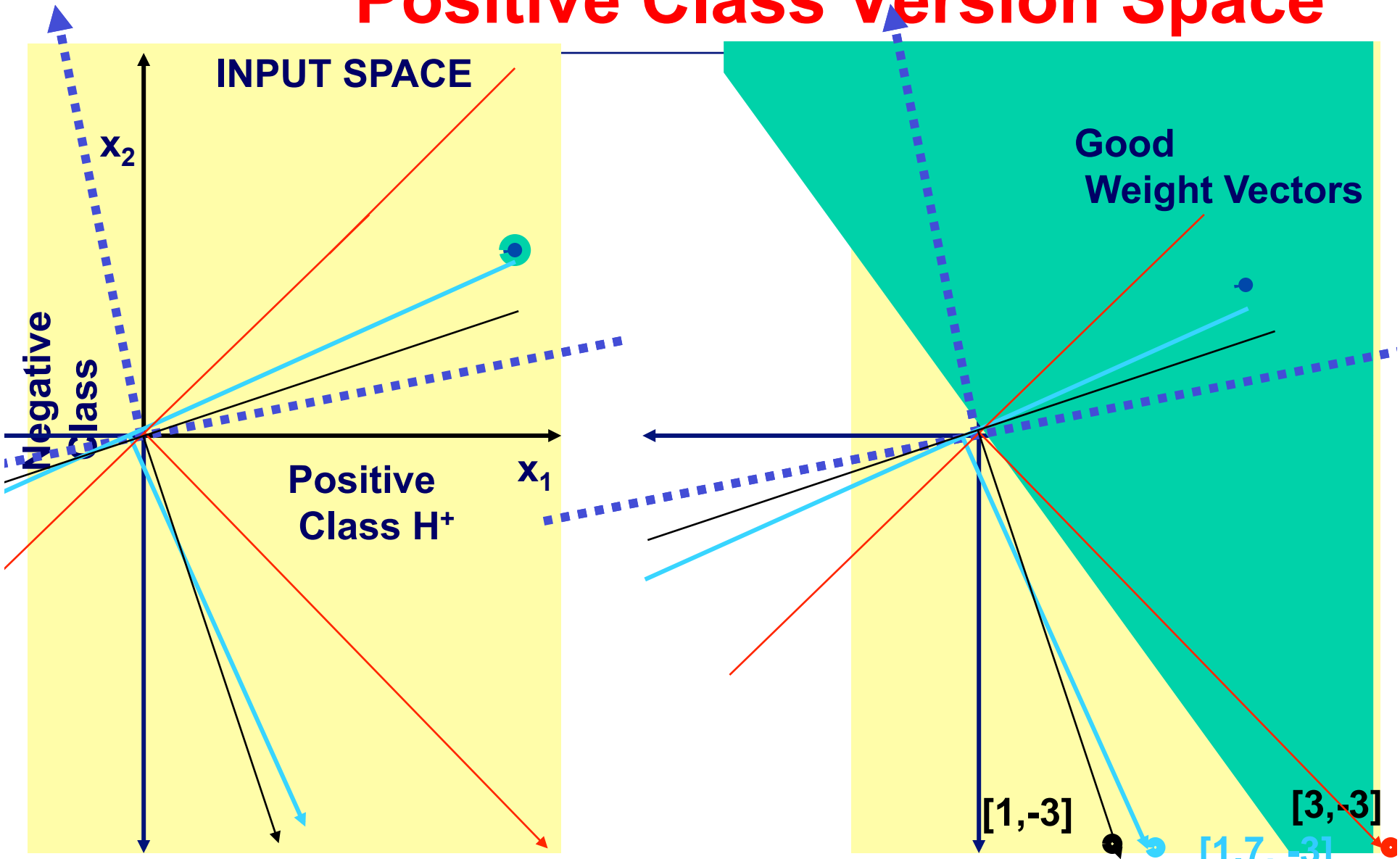


$$if \;\; WXy \geq 0 \text{ then correct}$$

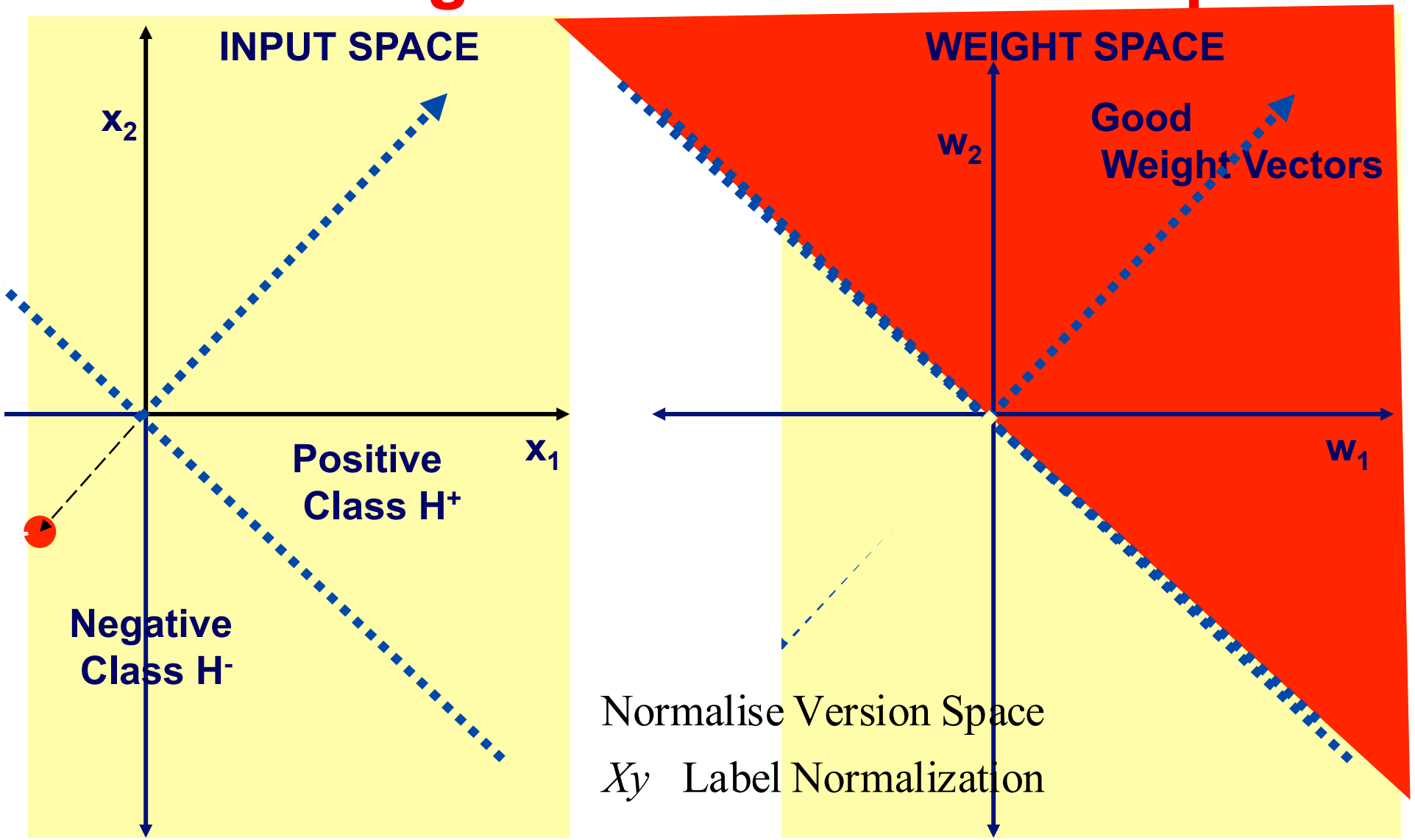
Positive Class Version Space



Positive Class Version Space

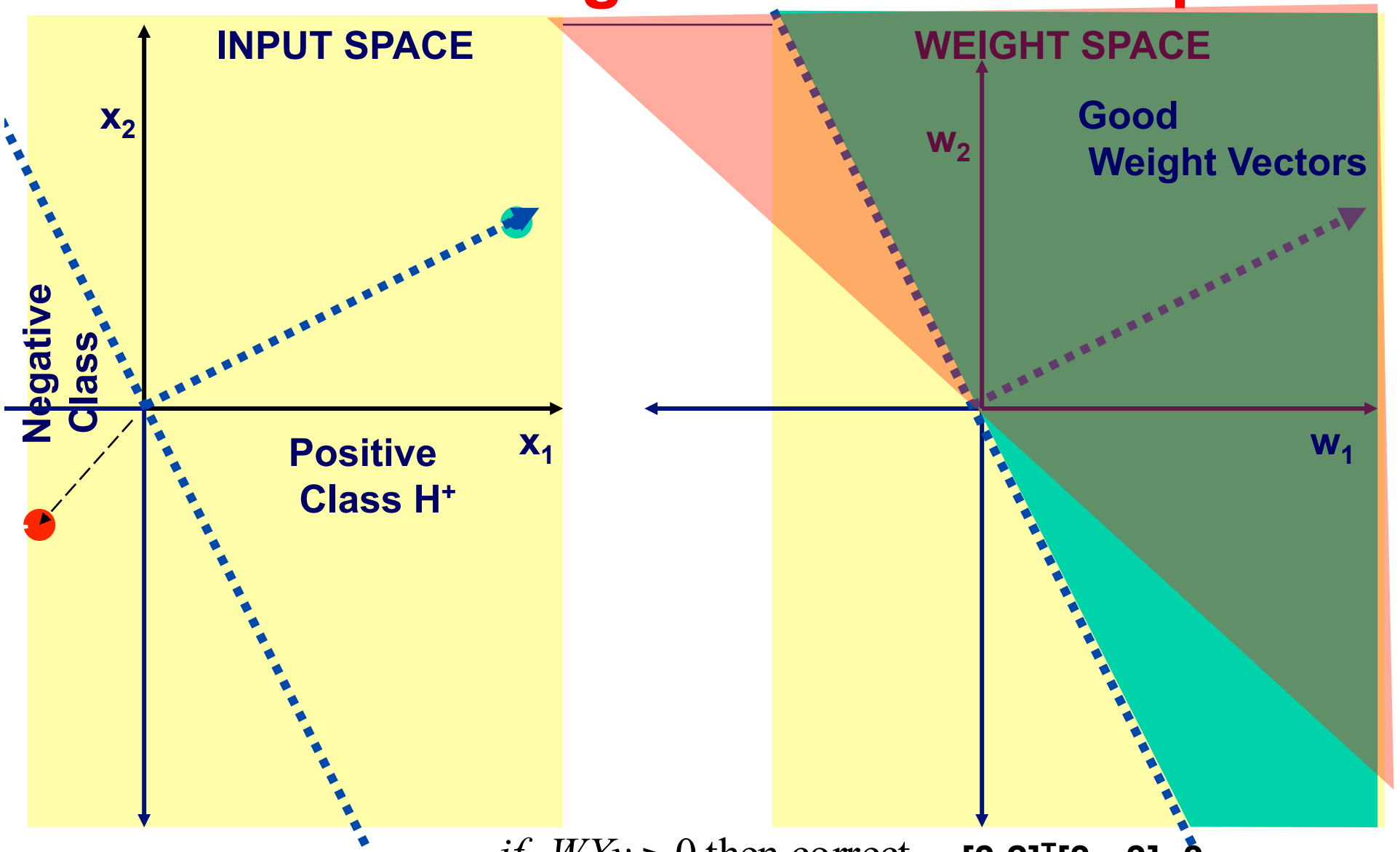


Negative Class Version Space



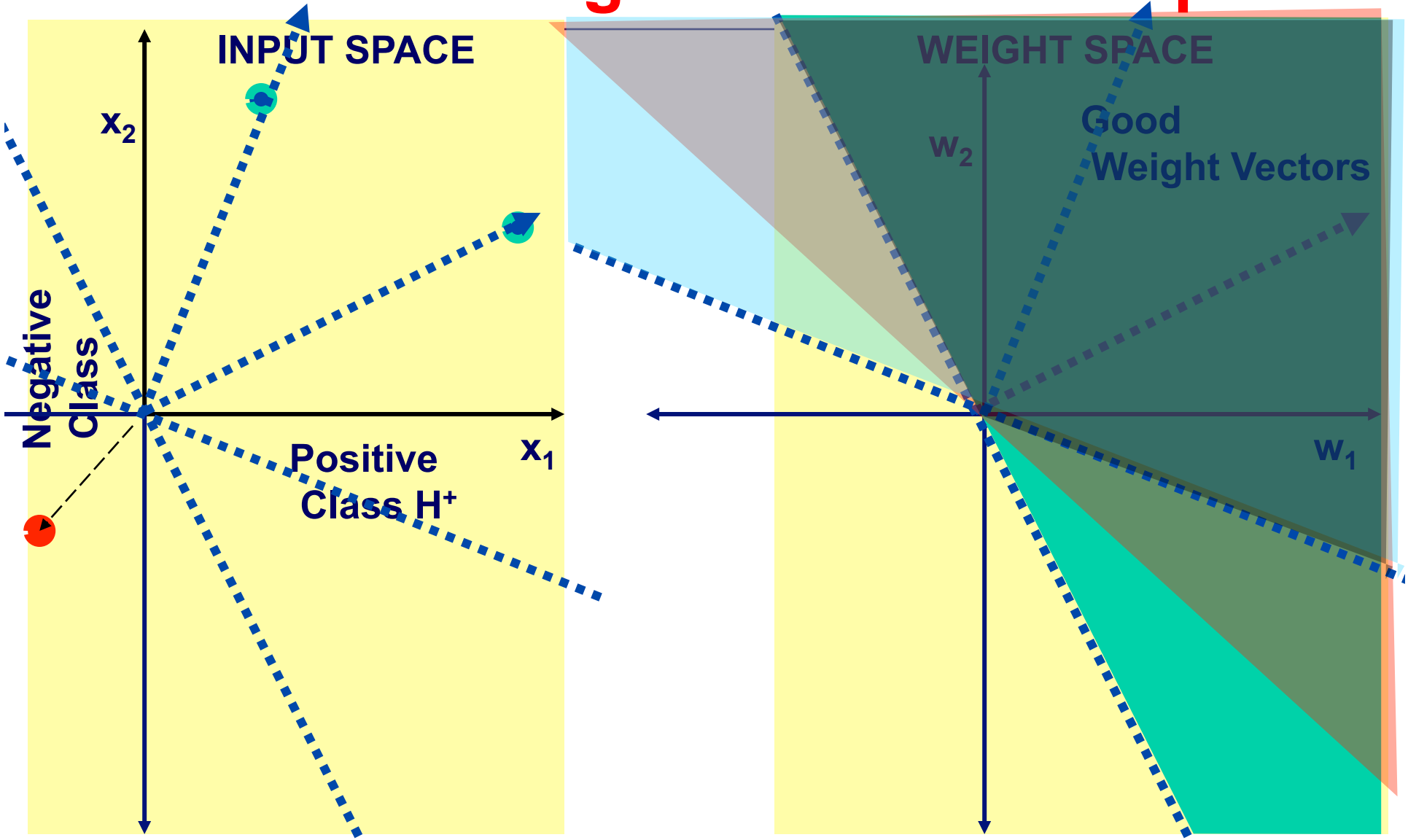
$if \;\; WXy \geq 0$ then correct

Pos/Neg Class Version Space



$if \;\; WXy \geq 0$ then correct $\quad [3,2]^T[3, -3]=3$

Pos/Neg Class Version Space



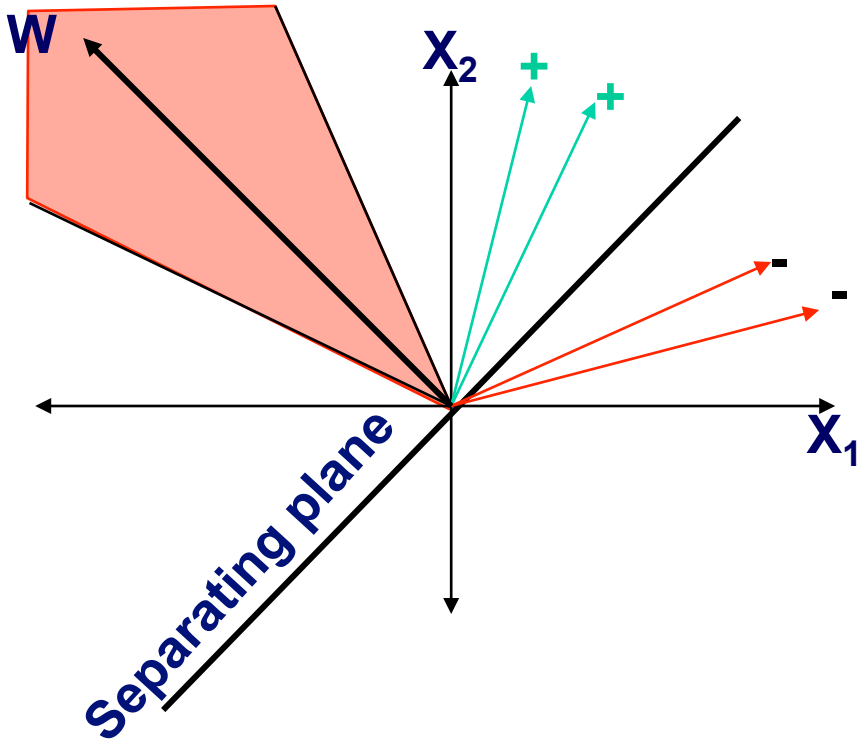
$$if \;\; WXy \geq 0 \text{ then correct}$$

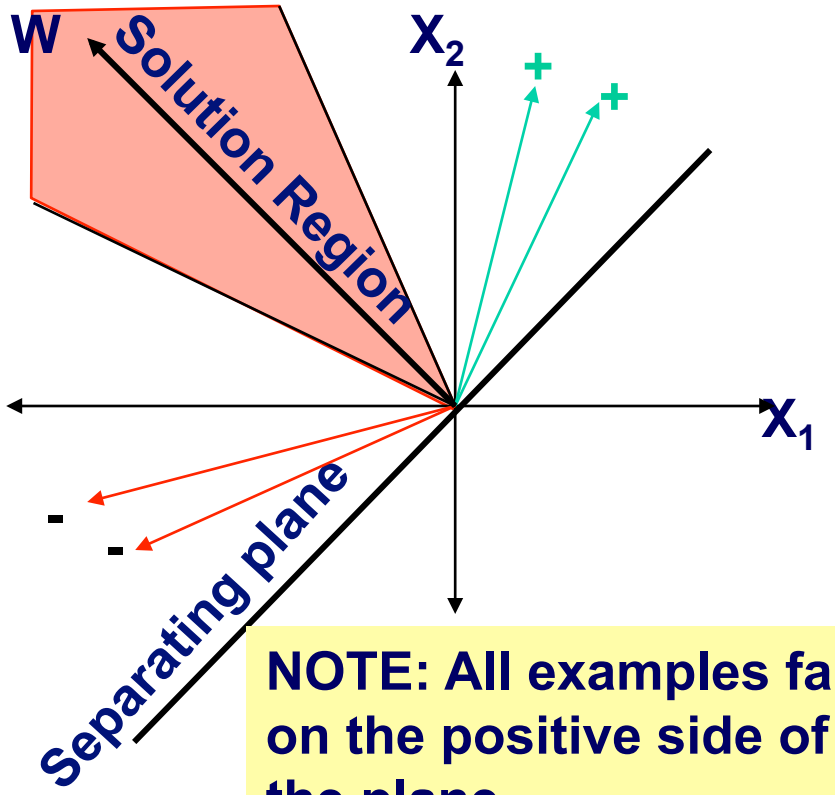
Label Normalization

Label Normalized Vector

$$Xy = \begin{bmatrix} x_1 \\ x_2 \\ .. \\ x_n \end{bmatrix} y = \begin{bmatrix} x_1 y \\ x_2 y \\ .. \\ x_n y \end{bmatrix}$$

Solution Region

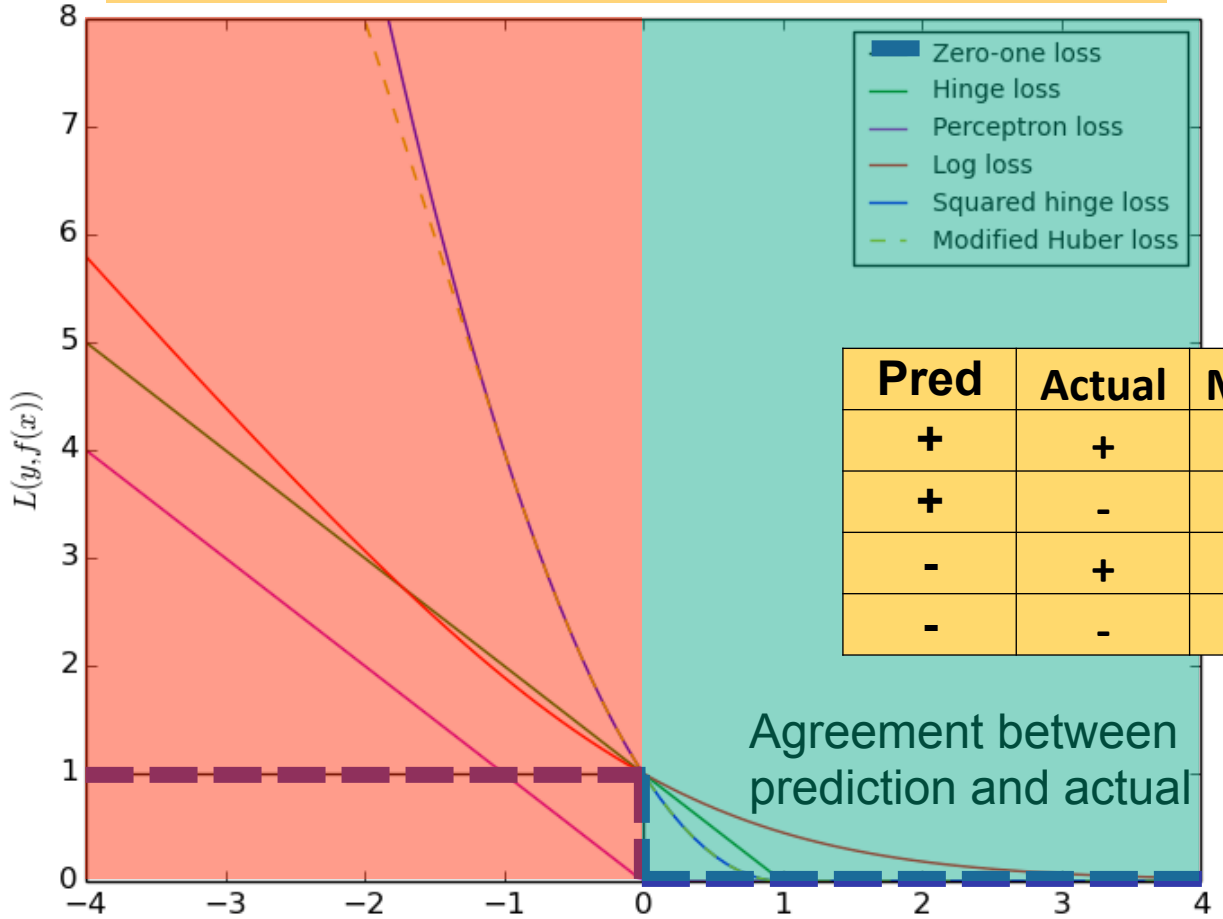




NOTE: All examples fall on the positive side of the plane

Machine Learning Objective Function: classification

Classification loss functions



Loss

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Disagreement between prediction and actual

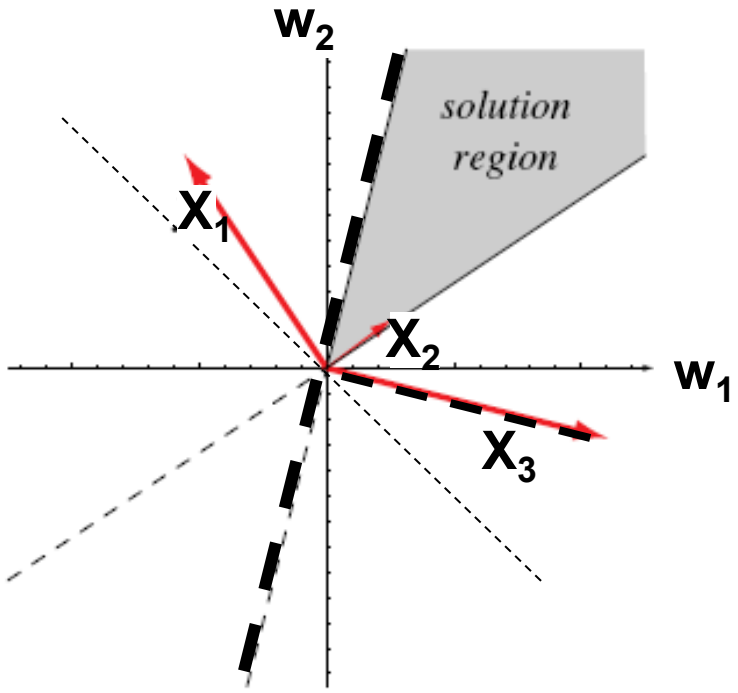
y ✕ f(X)

+ +=+

Margin

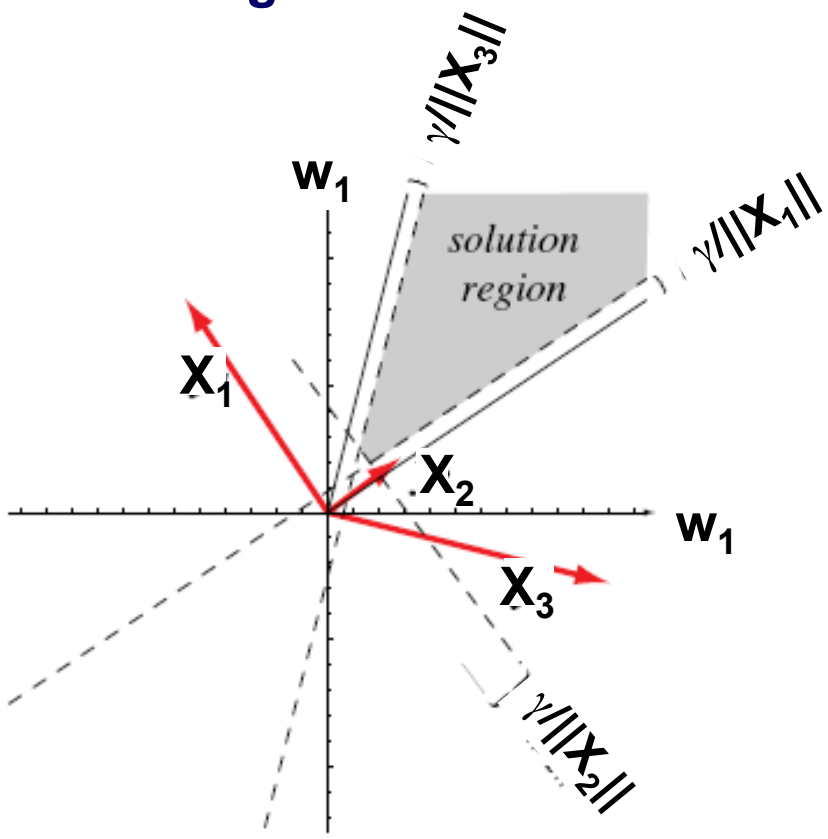
Version Space with add$^{al.}$ constraints

Get every example on the right side of the tracks



$$\langle W, X_i \rangle y_i \geq 0 \; \forall i$$

Get every example WELL INSIDE the right side of the tracks

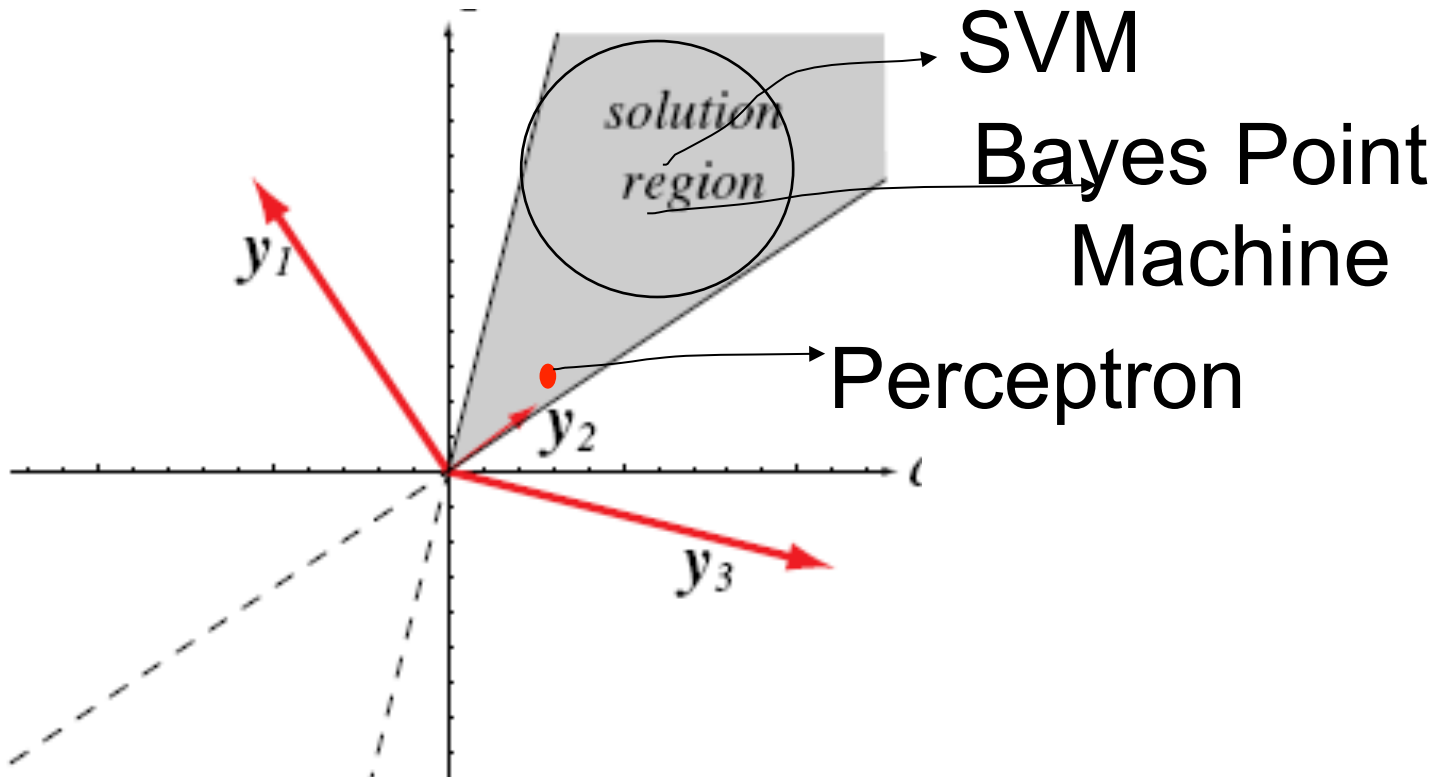


$$\langle W, X_i \rangle y_i \geq \gamma > 0 \; \forall i$$

Weight Vector and Solution Region

- **The hyperplane weight vector, *W*, can be thought of as specifying a point in the weight/version space**
- **Each example places a constraint on *W***
  - $(\langle W, X_i \rangle) y_i > 0$
- **The solution hyperplane must be on the positive side of each data induced hyperplane**
- **Solution region = the intersection of *L* half-spaces**
- **Impose additional constraints**
  - Find solution that is in the middle of the solution region (i.e., that is insulated from data anomalies)
  - Maximize the minimum distance from the training examples to the separating hyperplane
    - $(\langle W, X_i \rangle) y_i > \gamma$
  - γ is known as the classifier margin

Learning Algorithms in Version Space



SVMs find the center of the largest radius hypersphere whose center can be placed in version space and whose surface does not intersect with the hyperplanes corresponding to the labeled instances.

Gradient Descent

- **To find a solution to the set of linear inequalities $\langle W, X_i \rangle y_i > 0$;**
- **We define a criterion function J(W) that is minimized if *W* is a solution.**
- **This kind of problem can be solved by gradient descent.**
- **General approach**
  - Start with some vector W(1).
  - Generate then W(2) by taking a small step in the direction of *the steepest descent, i.e.,* "$-\nabla J(W(k))$"

Gradient Descent: *Minimize J(W)*

General Case

- **Basic Algorithm**
  - Set k = 0
  - Set initial weight vector, i.e., W(0) = **0**
  - **Repeat**
    - Compute the gradient $\nabla J\ (W(k))$
    - Update the weight vector
      - Calculate the next weight vector, *W(k+1),* by moving some distance from *W(k)* in the direction of the steepest descent, i.e., along the negative of the gradient
      - $W(k+1)=W(k) - \eta \nabla J\ (W(k))$
  - **Until a solution is found**

Where η is the learning rate

Potential Problems: learning rate, solution exists

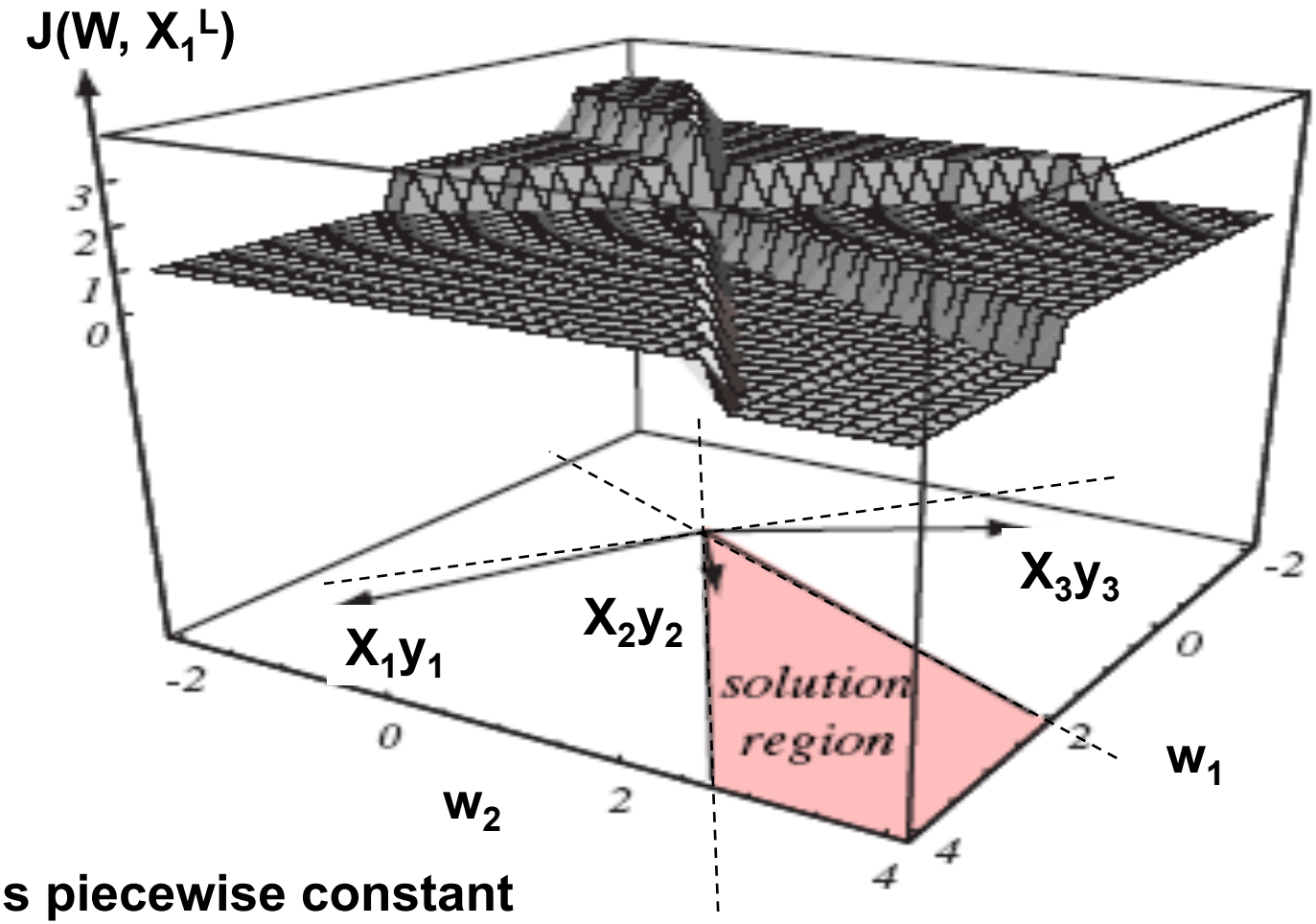
Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Objective Function: Number of Errors

E.g.	x_1	x_2	y
1	..	..	..
2	..	..	..
3	..	..	..

$J(W, X_1^L)$ = Number of Errors



3 examples (label normalized, i.e., Xy)
⇒Legal region corresponds to the intersection of positive half-spaces
=> Max $J(W, X_1^L)$ =3

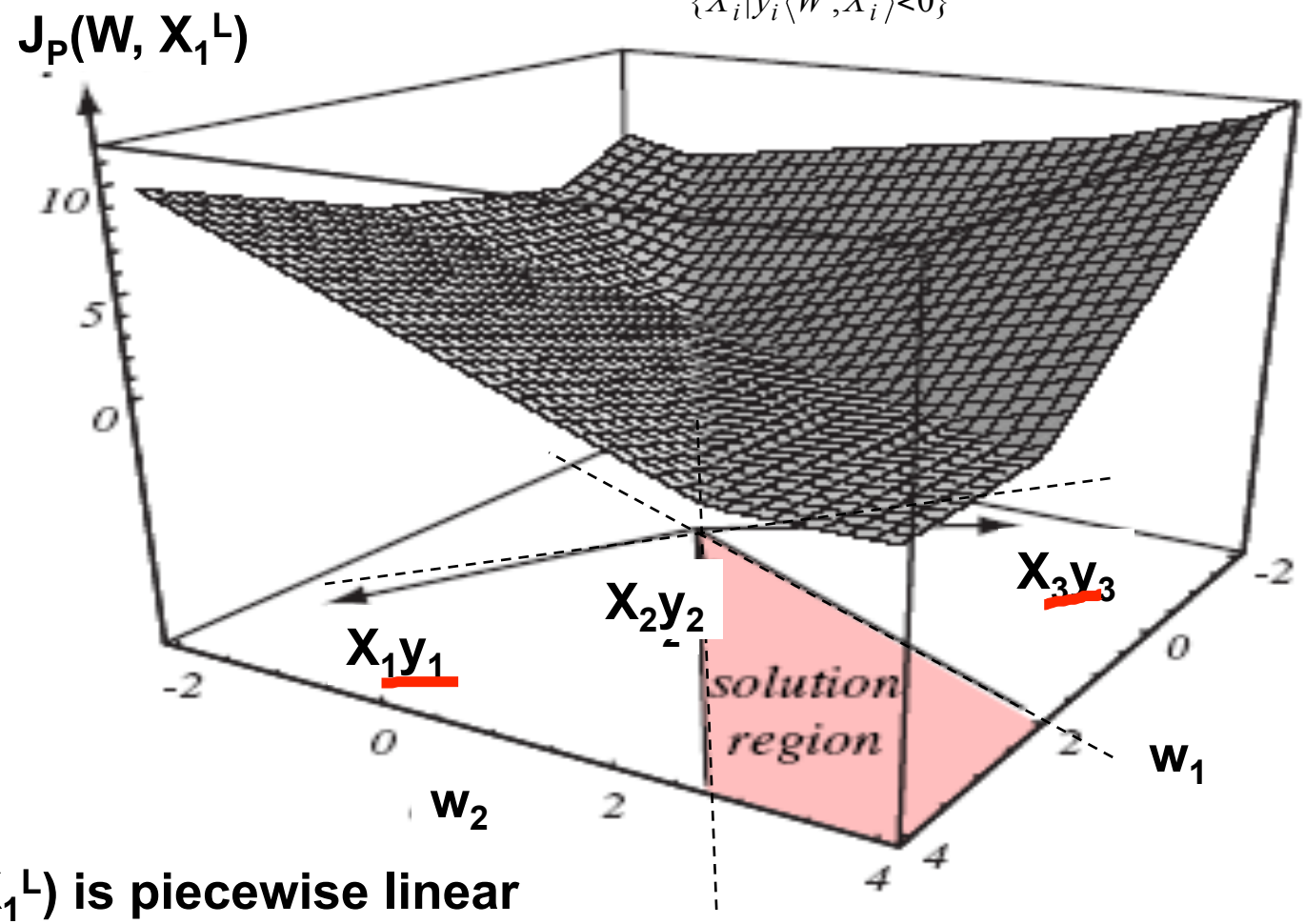
However, $J(W, X_1^L)$ is piecewise constant
=> Very poor candidate for gradient search

Objective Function: Perceptron

E.g.	x_1	x_2	y
1	..	..	..
2	..	..	..
3	..	..	..

$$J_P(W, X_1^L) = \sum_{\{X_i | y_i \langle W, X_i \rangle < 0\}} (-W^T X_i y_i)$$

$J_p(W, X_1^L)$



However, $J_p(W, X_1^L)$ is piecewise linear
=> Acceptable for gradient search

Perceptron using Gradient Descent

Single Update Rule

- General Update Rule
  - **W(k+1)=W(k) - $\eta \nabla J$ (W(k))**

$$J_P(W, X_1^L) = \sum_{\{X_i | y_i \langle W, X_i \rangle < 0\}} (-W^T X_i y_i)$$

Perceptron Objective Function

$$\nabla J_P = \frac{\partial (J_P(W, X_1^L))}{\partial w} = \sum_{\{X_i | y_i \langle W, X_i \rangle < 0\}} (-X_i y_i)$$

Gradient if WXy<0 otherwise 0

$$W(K+1) = W(k) + \eta(k) X y_i$$

Perceptron SINGLE Update Rule

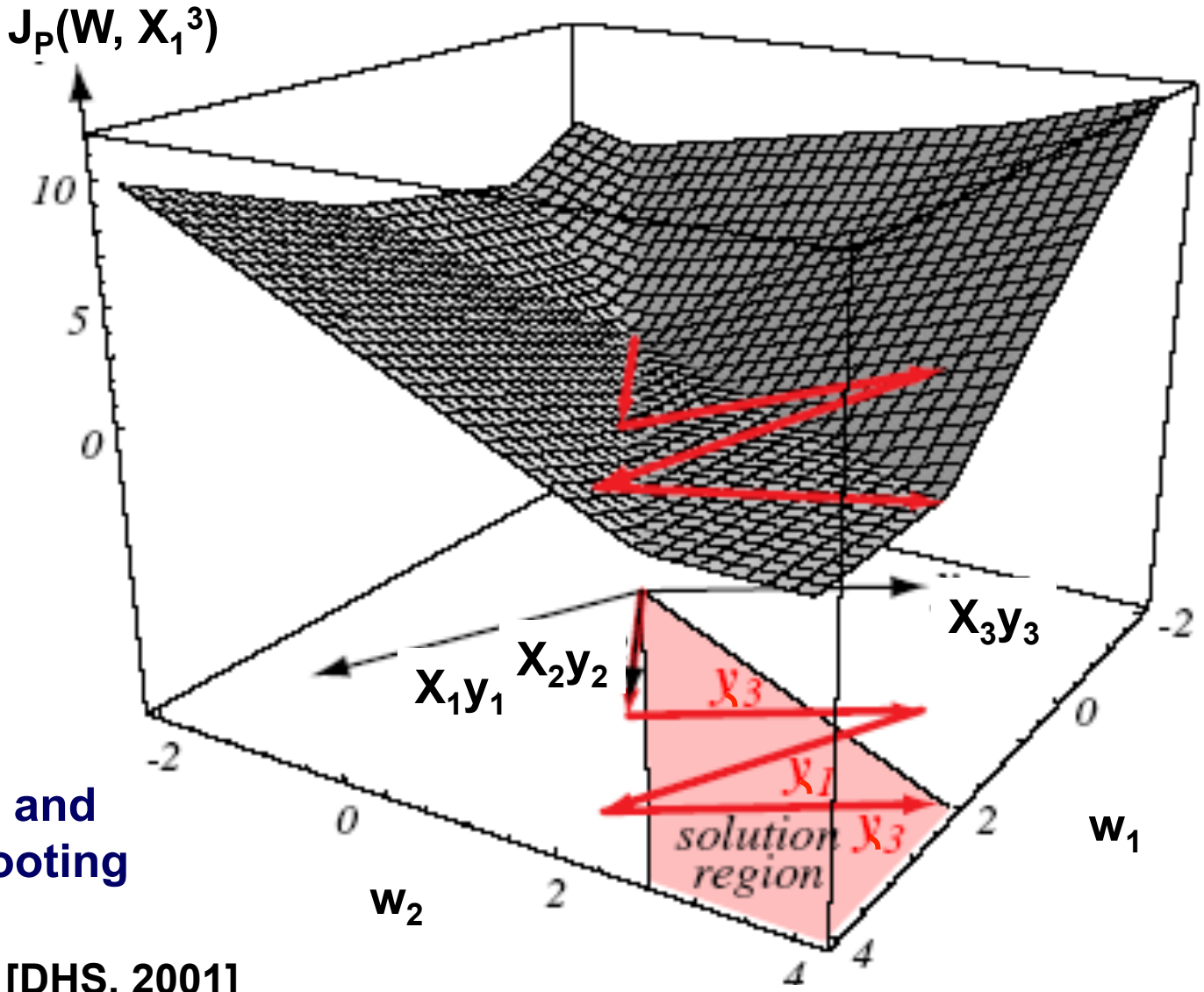
Intuitively, drag weight vector closer to the misclassified example

Perceptron Update Example

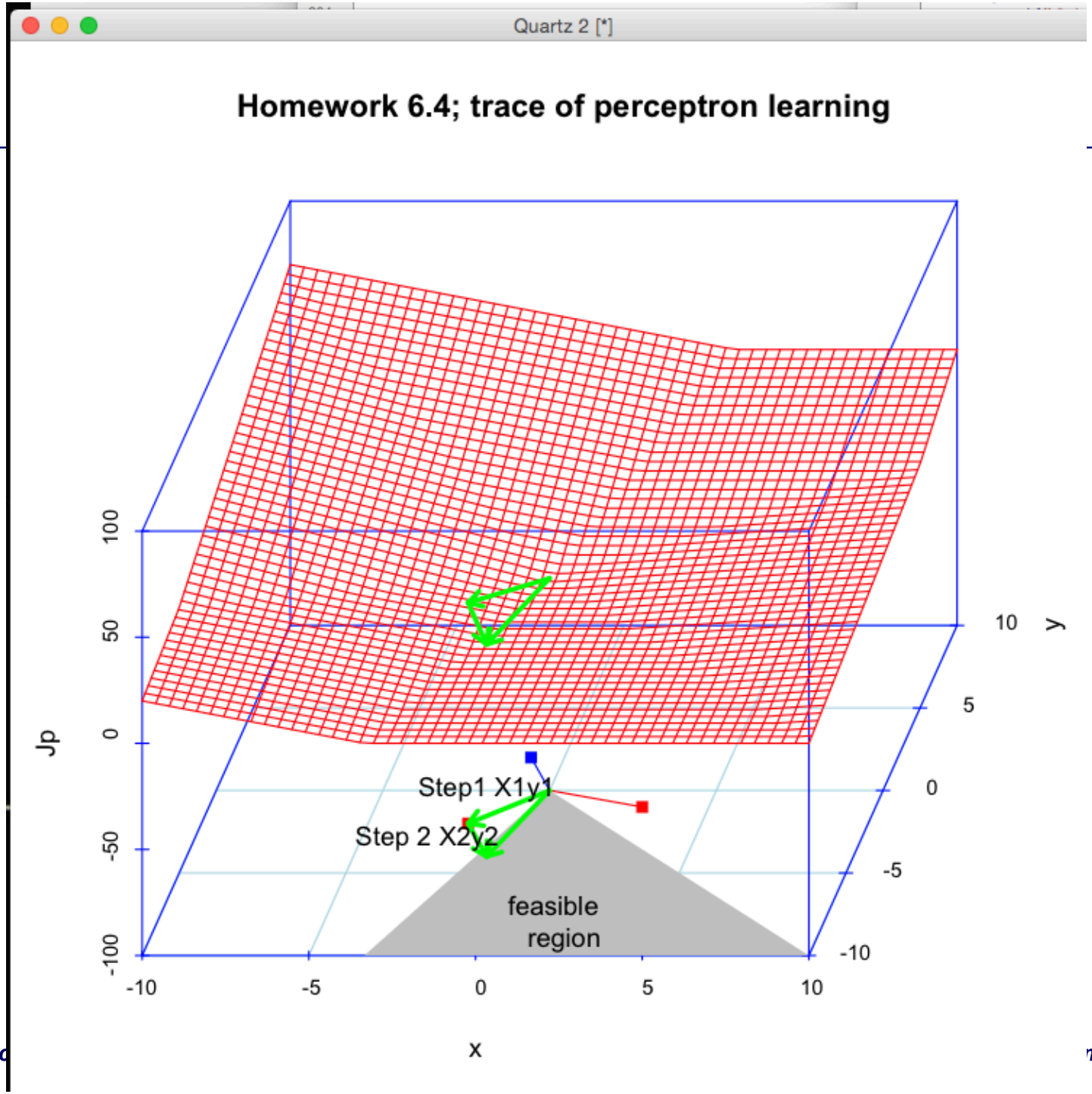
$$W' = W + Xy_i$$

Start with W = [0, 0]
Update Sequence:
X_2, X_3, X_1, X_3

Updating with X_3y_3 and X_2y_1 cause overshooting

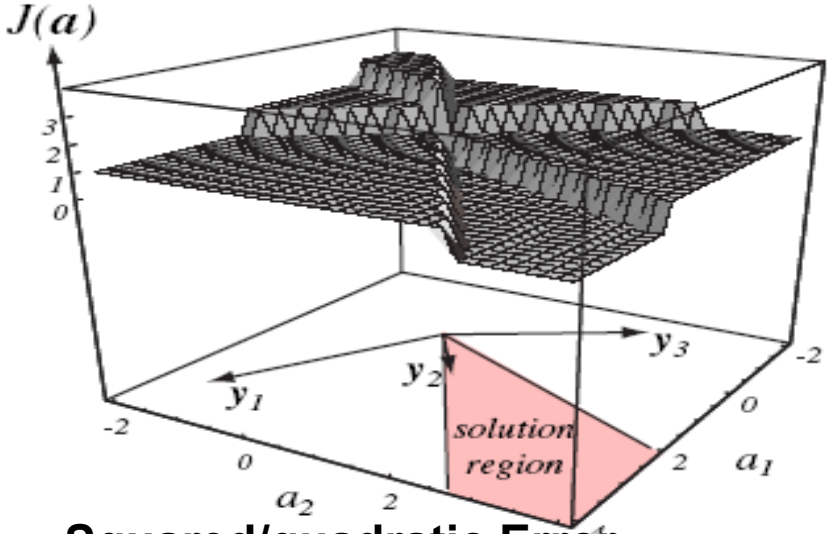


Adapted from: [DHS, 2001]

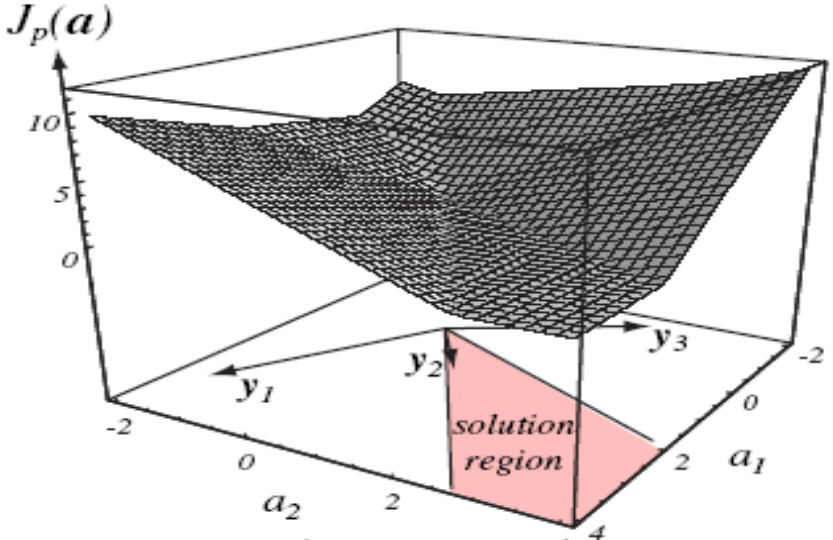

Quartz 2 [*]
Homework 6.4; trace of perceptron learning
Jp
100
50
0
-50
-100
-10
-5
0
5
10
x
y
10
5
0
-5
-10
Step1 X1y1
Step 2 X2y2
feasible
region

Different Objective Functions

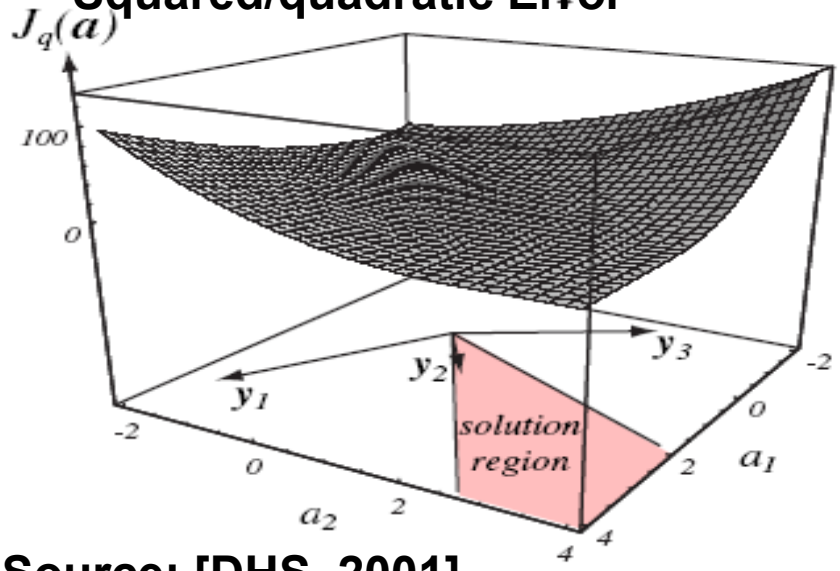
Number misclassified examples



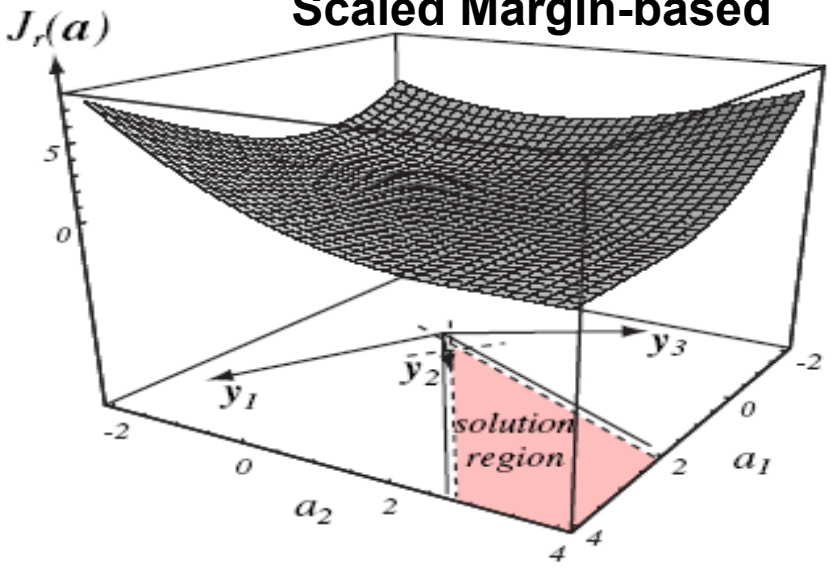
Perceptron Objective



Squared/quadratic Error



Scaled Margin-based



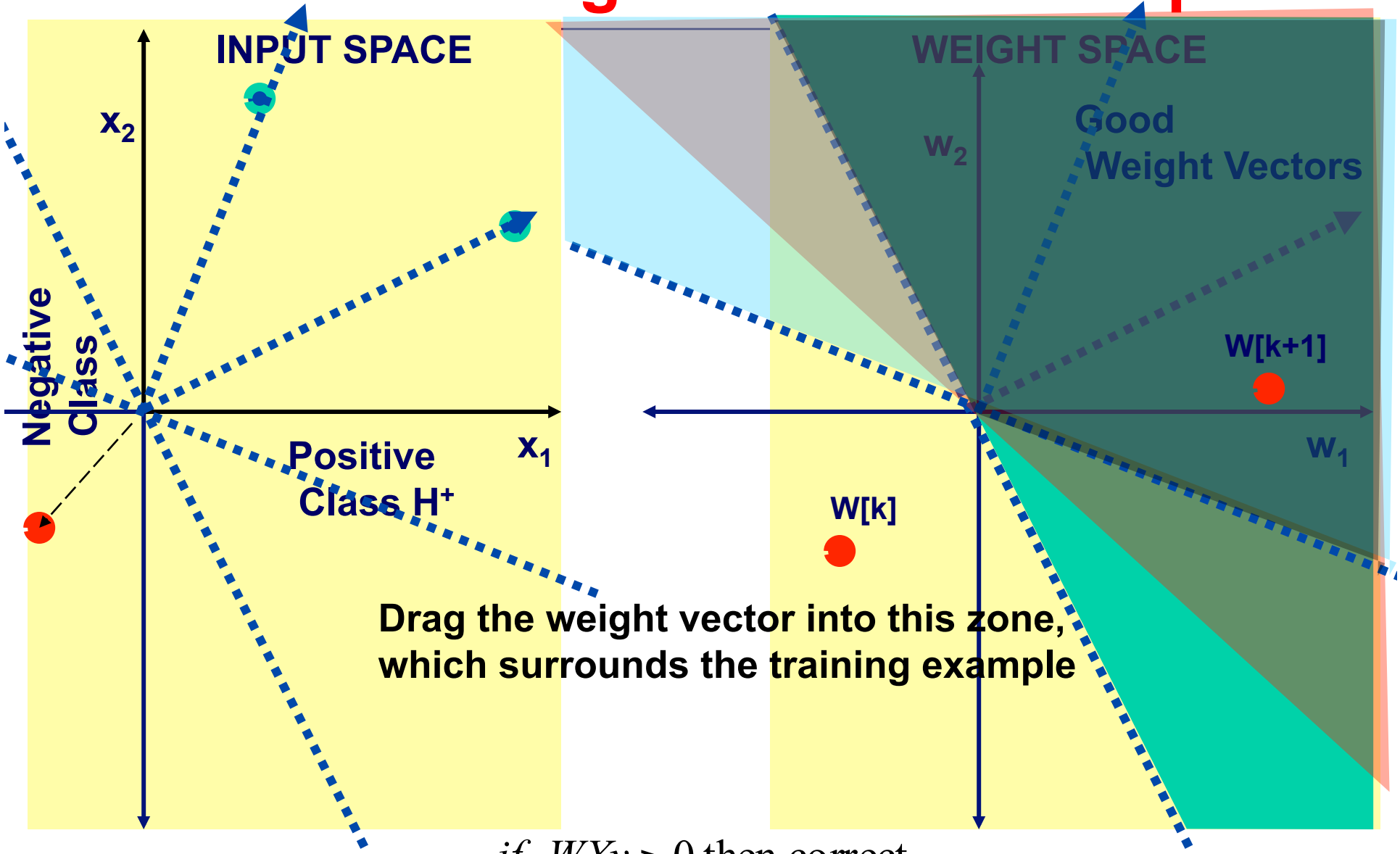
Source: [DHS, 2001]

Gradient Descent

- **Basic Algorithm**
  - Set k = 0
  - Set initial weight vector, i.e., W(0) = **0**
  - **Repeat**
    - Compute the gradient $\nabla J\ (W(k))$
    - Update the weight vector
      - Calculate the next weight vector, *W(k+1),* by moving some distance from *W(k)* in the direction of the steepest descent, i.e., along the negative of the gradient
      - $W(k+1)=W(k) - \eta \nabla J\ (W(k))$
  - **Until a solution is found**

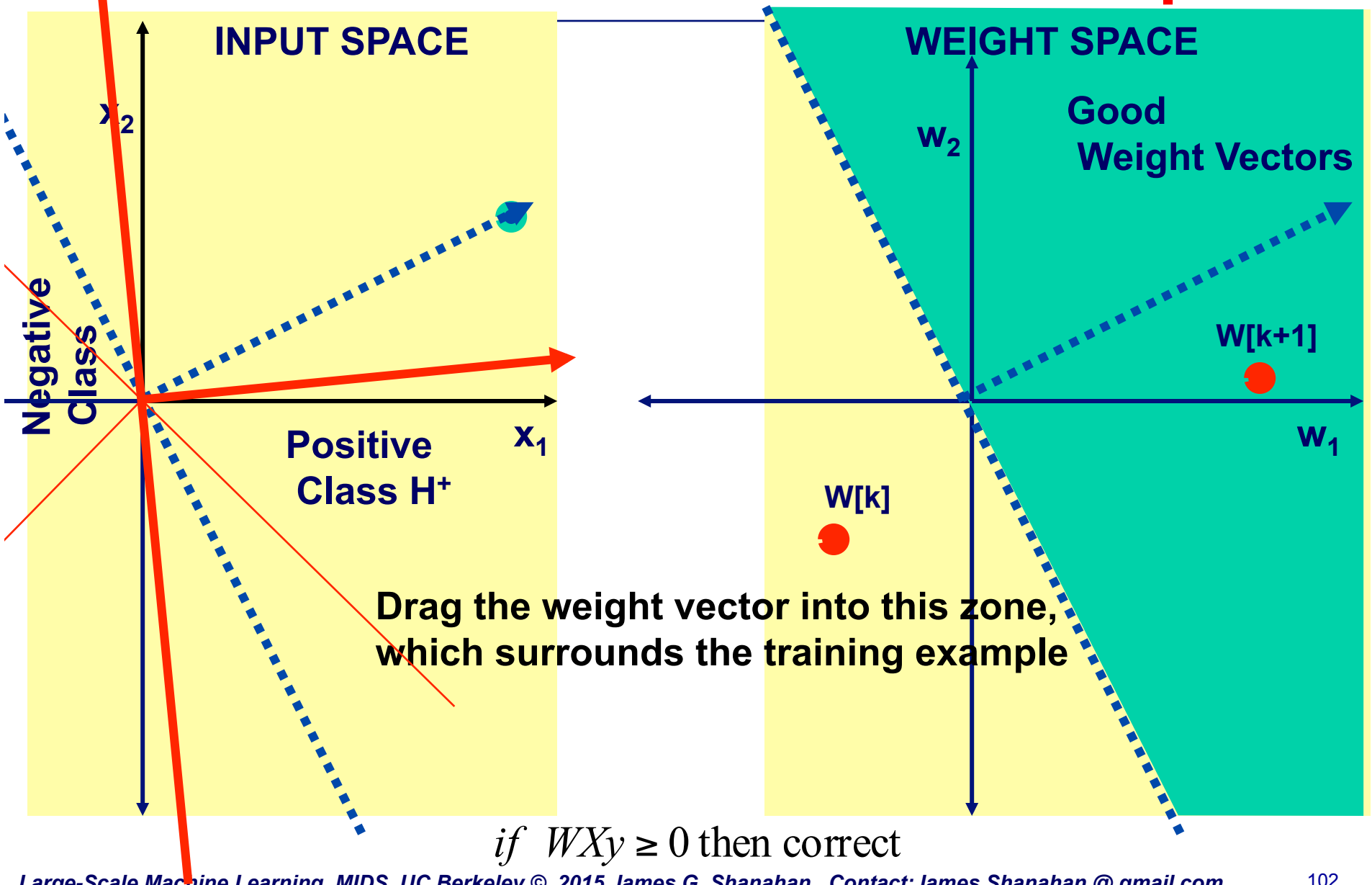
Where η is the learning rate

Pos/Neg Class Version Space



$$if \;\; WXy \geq 0 \text{ then correct}$$

Remember: Class Version Space



$$if \;\; WXy \geq 0 \text{ then correct}$$

Perceptron Algorithm

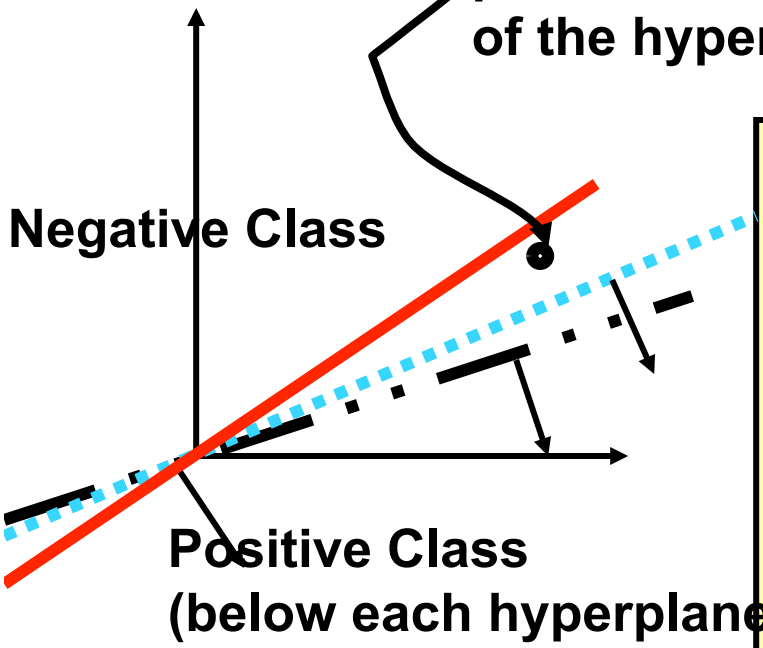
Single-sample Primal Form

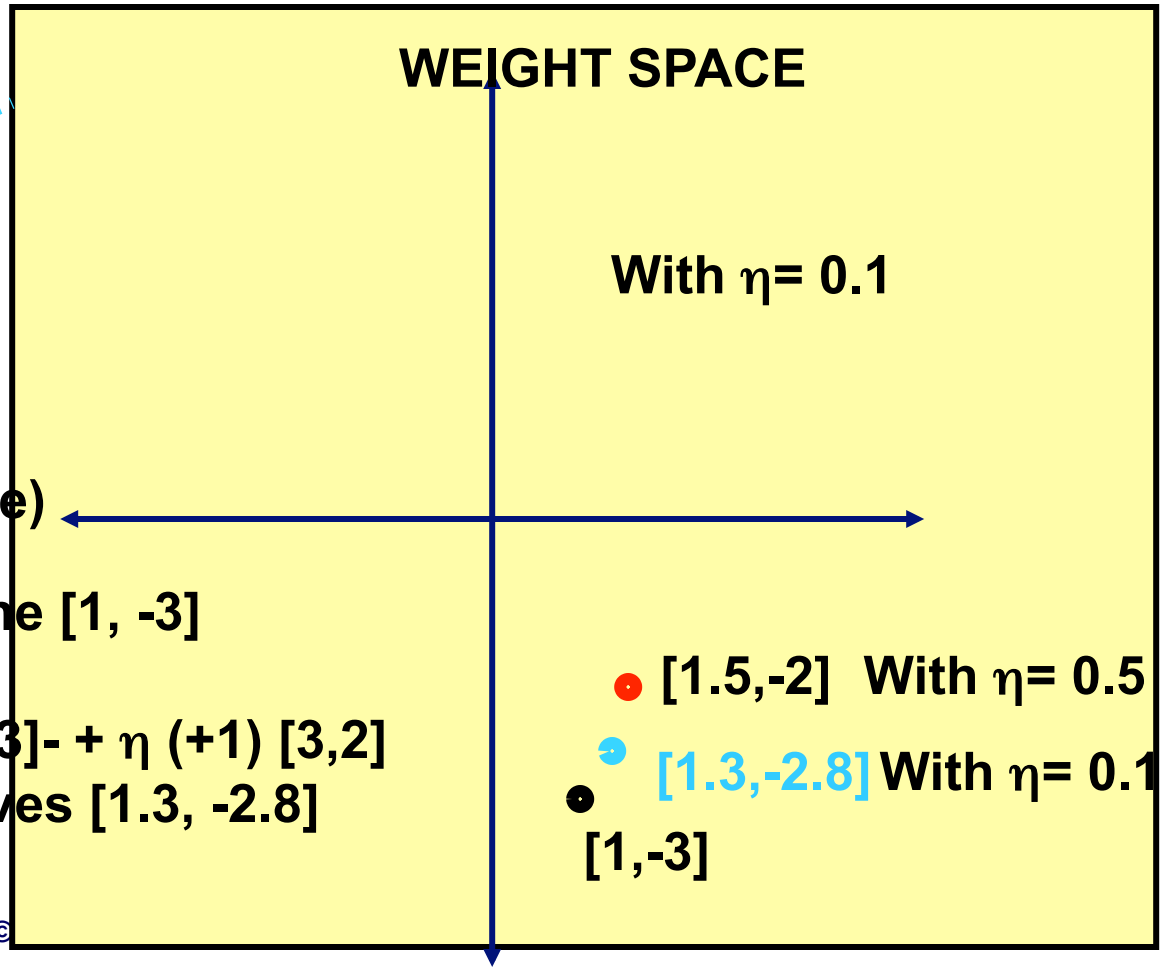
- **Given Training data S where each example *i* is of the form $(x_{i,1},\ldots, x_{i,n},y_i)$, and a learning rate η**
- **Set W_o to zeros; k=0;**
- **Repeat**
  - For i = 1 to |*Train*| do
    - If $(y_i(\langle W_k, X_i\rangle + b_k)) \leq 0$ then // $y_i \neq \text{Sgn}(\langle W_k X_i\rangle + b_k)$ MISTAKE
      - $W_{k+1} = W_k + \eta\, y_i X_i$ // Update weights with example i
      - $k = k + 1$ // *Update number of mistakes*
    - End-If
  - End-For
- **Until no mistakes are made**
- **Return *k*, *W***

$$J_P(W, X_1^L) = \sum_{\{X_i | y_i \langle W, X_i\rangle < 0\}} (-W^T X_i y_i)$$

Example of weights updating

[3,2] is an example for the +Class; so should yield positive value when substituted in to the equation of the hyperplane





Must update since hyperplane [1, -3] incorrectly classifies
Update weight vector to [1, -3]- + η (+1) [3,2]
E.g. [1, -3]- + 0.1 (+1) [3,2] gives [1.3, -2.8]

Perceptron Algorithm: Alternative Form

Single-sample Primal Form

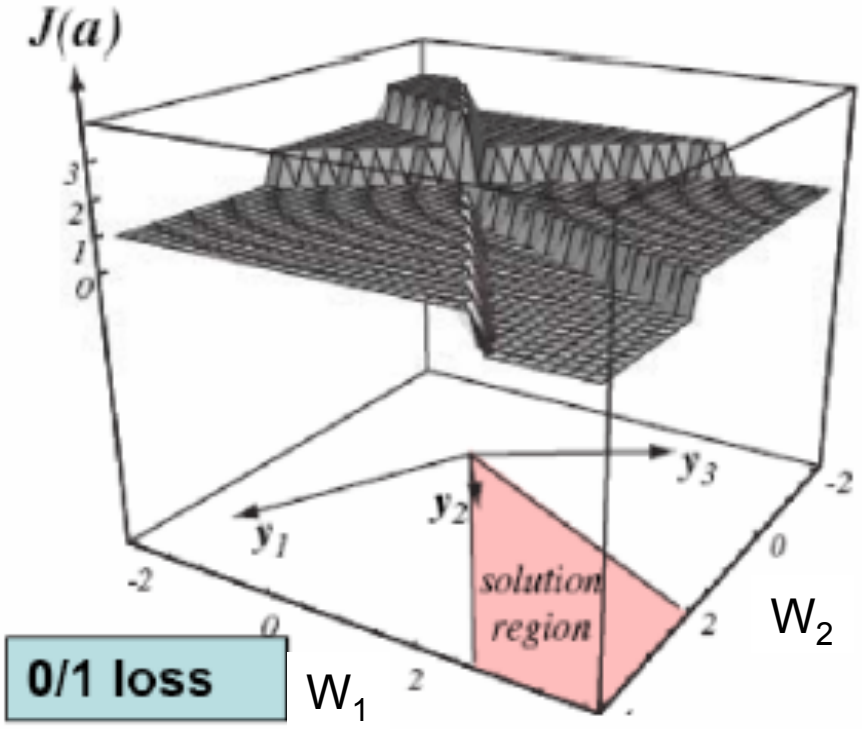
- **Given Training data S where each example *i* is of the form $(x_{i,1}, \ldots, x_{i,n}, y_i)$, and a learning rate η**
- **Set W_o to zeros; b_o=0; *k*=0;**
- **Repeat**
  - For i = 1 to |*Train*| do
    - If $(y_i(\langle W_k X_i \rangle + b_k)) \leq 0$ then // $y_i \neq Sgn(\langle W_k X_i \rangle + b_k)$ *MISTAKE*
      - $W_{k+1} = W_k + \eta\, y_i X_i$ // Update weights with example i
      - $b_{k+1} = b_k + \eta\, y_i R^2$ // Update bias
      - $k = k + 1$ // *Update number of mistakes*
    - End-If
  - End-For
- **Until no mistakes are made**
- **Return *k*, *W* and *b***
  - where *k* is the number of mistakes made and *W* is the weight vector, b is the bias and R is the radius of the ball containing S, i.e.,

$$R = \max_{i=1}^{|Train|} \sqrt[2]{\langle X_i, X_i \rangle} \text{ a.k.a. two-norm } \|X\|_2$$

0-1 Loss function for Classification

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N}\sum_{i=1}^{N} L(\mathrm{sgn}(w \cdot x_i), y_i)$

  L(y',y) = 0 when y'=y, otherwise L(y',y)=1
- Does not produce useful gradient since the surface of J is flat



Count the number of mistakes

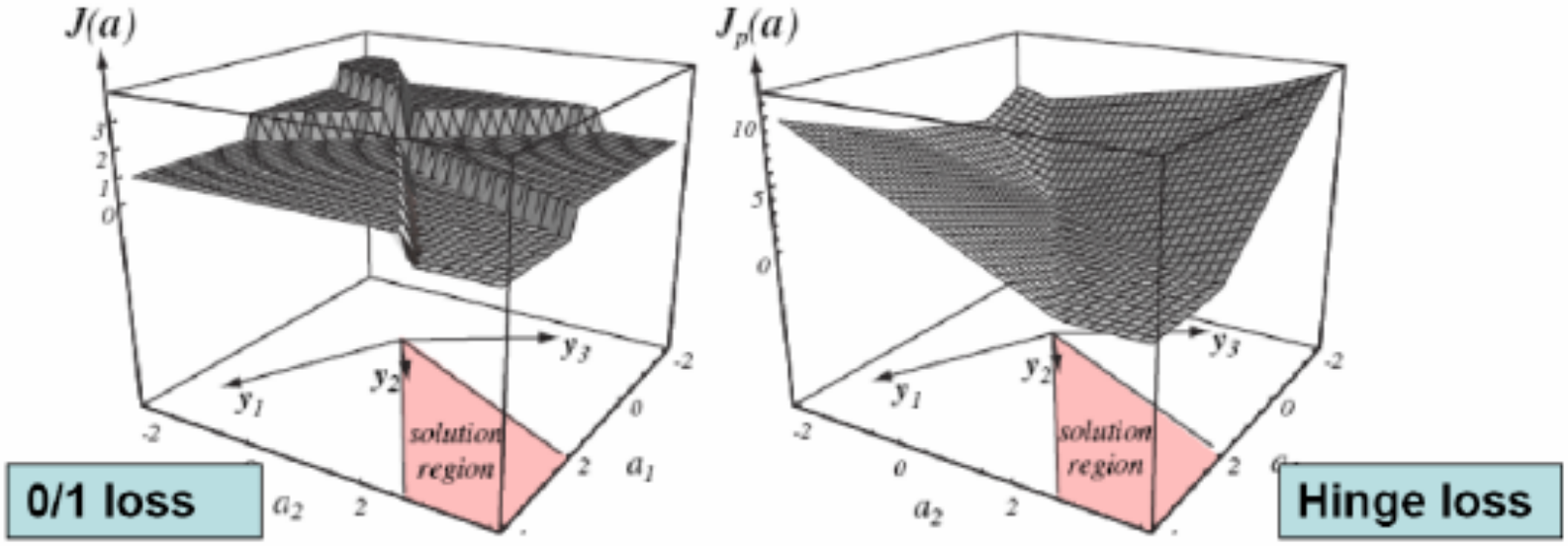
Slide Improvisation

Hinge Loss

- Instead we will consider the "**hinge loss**":

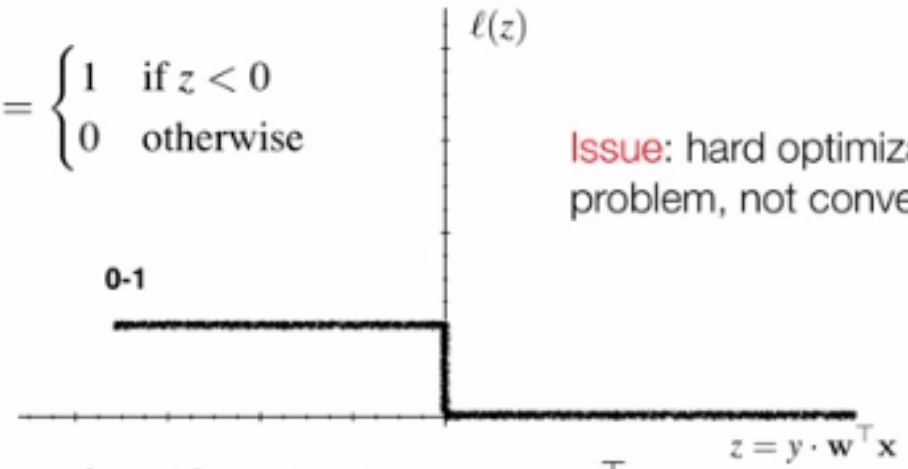
$$J_p(w) = \frac{1}{N}\sum_{i-1}^{N} \max(0, -y_i w \cdot x_i)$$

- The term $\max(0, -y_i w \cdot x_i)$ is 0 when y_i is predicted correctly otherwise it is equal to the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region



0/1 Loss Minimization

$$\ell_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$



Issue: hard optimization problem, not convex!

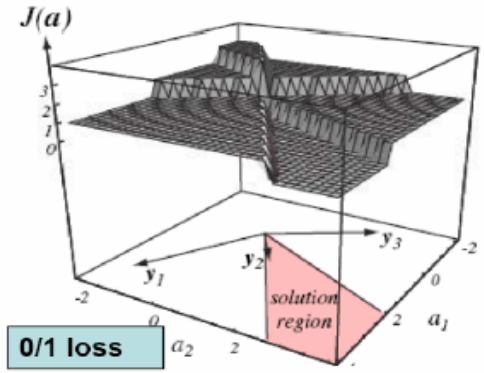
Let $y \in \{-1, 1\}$ and define $z = y \cdot \mathbf{w}^\top \mathbf{x}$

z is positive if y and $\mathbf{w}^\top \mathbf{x}$ have same sign, negative otherwise

0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

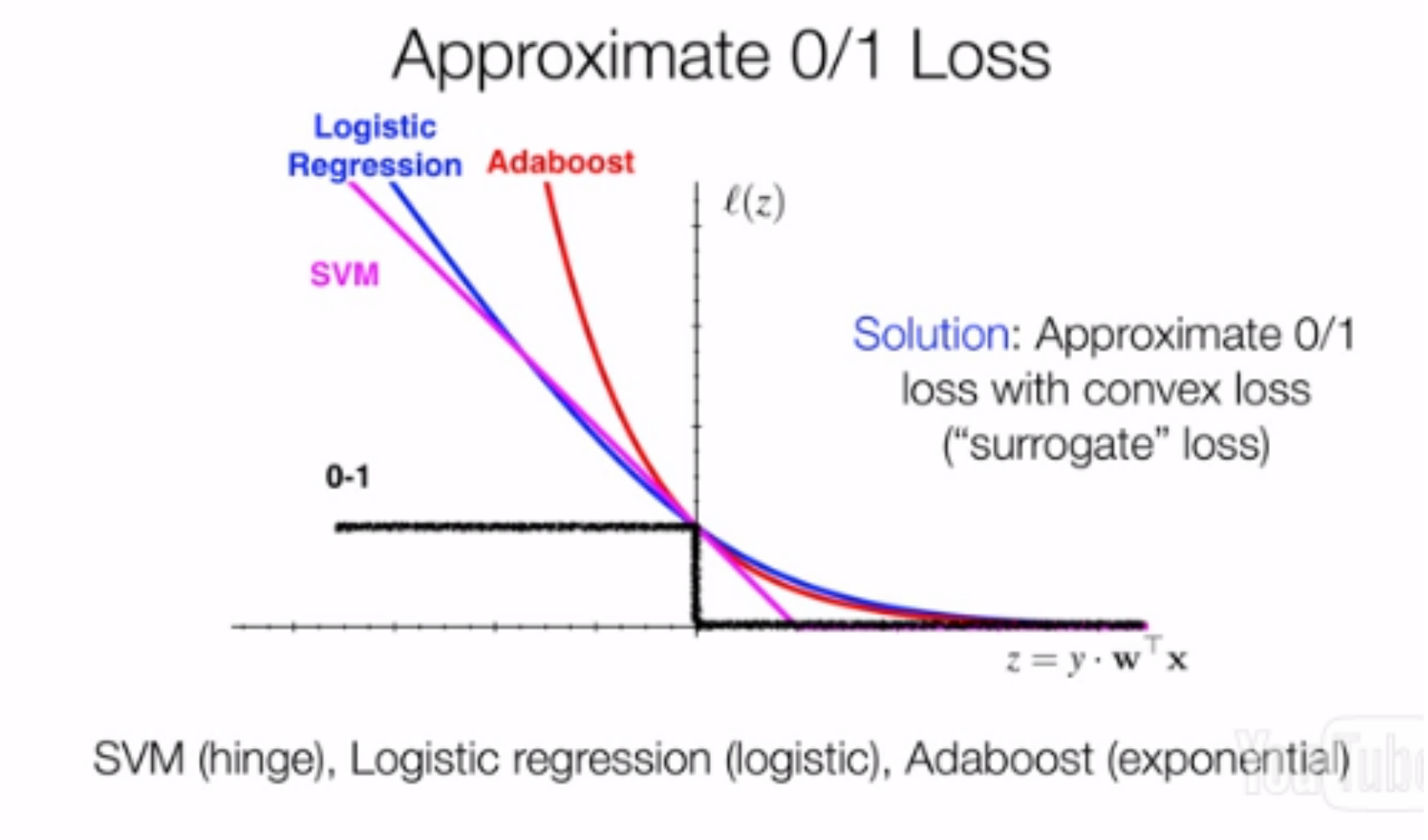
- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N}\sum_{i=1}^{N} L(\text{sgn}(w \cdot x_i), y_i)$

  L(y',y) = 0 when y'=y, otherwise L(y',y)=1
- Does not produce useful gradient since the surface of J is flat



What can we do?
- Approximate 0/1 loss with a convex loss surrogate
- E.g., Hinge Loss Function

Zero-one loss versus Hinge Loss



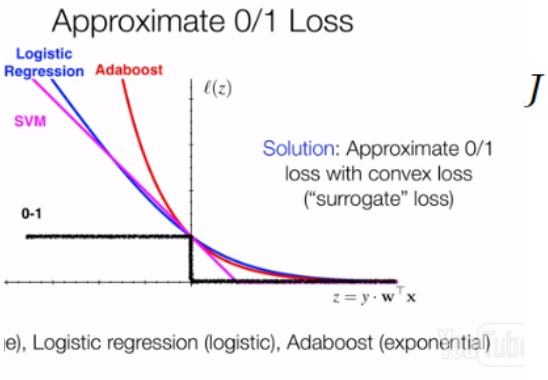
Loss Terms: 5 examples
First: 0-1 loss and Hinge loss

In this section, we use 5 examples to illustrate the loss term. The five example are the Gold Standard(ideal case),Hinge(for soft margin SVM), log(for logistic regression,cross entropy error), squared loss(for linear regression) and Boosting.
First, let's see the "gold standard" loss function. We've implicitly been using it to evaluate our classifiers - we count the number of mixtakes that are made. This is often called "0-1" loss, or L_{01}

$$L_{01}(m) = \begin{cases} 0 & \text{if } m \geq 0 \\ 1 & \text{if } m < 0 \end{cases}$$

Count the number of mistakes

Second, let's look at loss term for soft margin SVM. We use L_{hinge} to represent the hinge loss.



Reg. Term Loss Term

$$J(w) = \frac{1}{2} \parallel w \parallel^2 + \sum_i \max(0, 1 - y^i w^T x^i) \quad (14.5)$$

$$= \frac{1}{2} \parallel w \parallel^2 + \sum_i \max(0, 1 - m_i(w)) \quad (14.6)$$

$$= R_2(w) + \sum_i L_{hinge}(m_i) \quad (14.7)$$

http://www.ics.uci.edu/~dramanan/teaching/ics273a_winter08/lectures/lecture14.pdf

Logistic Regression assumes a parametric form for P(Y|X)

Logistic Regression assumes a parametric form for the distribution $P(Y|X)$, then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is boolean is:

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)} \quad (16)$$

and

$$P(Y=0|X) = \frac{\exp(w_0+\sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)} \quad (17)$$

Equivalently, we can work with the log of the conditional likelihood:

$$W \leftarrow \arg\max_W \sum_l \ln P(Y^l|X^l,W)$$

https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

This conditional data log likelihood, which we will denote $l(W)$ can be written as

$$l(W) = \sum_l Y^l \ln P(Y^l=1|X^l,W) + (1-Y^l)\ln P(Y^l=0|X^l,W)$$

Maximize Loss term L(M)

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1+\exp(w_0+\sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l=1|X^l,W))$$

Maximum Likelihood Estimator for Logistic Regression

Cost Function

- Model output $\hat{y}_k = P(y_k = 1 | \varphi_k)$
- Likelihood contribution $l_{\theta,k} = \hat{y}_k^{\,y_k} (1 - \hat{y}_k)^{1-y_k}$
- Likelihood function $L_\theta = \prod_{k=1}^{N} l_{\theta,k}$
- Log-likelihood function $\ln L_\theta = \sum_{k=1}^{N} (y_k \ln \hat{y}_k + (1 - y_k) \ln(1 - \hat{y}_k))$

Maximum Likelihood Criterion

$$\max_\theta \ln L_\theta \Leftrightarrow \min_\theta -2 \ln L_\theta$$

Third, let's see log loss, which is equivalent to the cross entropy loss function used to train a logistic regression model

LR Simplified

$$J(w) = \lambda \parallel w \parallel^2 + \sum_i y^i \log g_w(x^{(i)}) + (1 - y^i)(\log 1 - g(x^{(i)})), y^i \in (0, 1) \tag{14.8}$$

$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$f_w(x^{(i)}) = w^T x^{(i)}$$

See next slide for derivation

Because

$$\begin{aligned} 1 - g(x^{(i)})) &= 1 - \frac{1}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{e^{-f_w(x^{(i)})}}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{1}{1 + e^{f_w(x^{(i)})}} \end{aligned}$$

So, we can transform the equation into the following form,

$$J(w) = \lambda \parallel w \parallel^2 + \sum_i \log 1 + e^{-y^{\bar{(i)}} f_w(x^{(i)})} \tag{14.9}$$

Loss L(M)

$$L(m) = \log 1 + e^{-m} \tag{14.10}$$

Where m is defined

$$m^i = y^{\bar{(i)}} f_w(x^{(i)})$$

$$y^{\bar{(i)}} = \begin{cases} -1 & \text{if } y^{(i)} = 0 \\ 1 & \text{if } y^{(i)} = 1 \end{cases}$$

Third, let's see log loss, which is equivalent to the cross entropy loss function used to train a logistic regression model

$$J(w) = \lambda \parallel w \parallel^2 - \sum_i y^i \log g_w(x^{(i)}) + (1 - y^i)(\log 1 - g(x^{(i)})), y^i \in (0, 1) \quad (14.8)$$

Minimize log loss

Minimize the **NEG** log joint conditional likelihoo
The conditional likelihood of θ given data x and y is L(θ; y|x) = p(y|x) = f(y|x; θ).

$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$f_w(x^{(i)}) = w^T x^{(i)}$$

Simplify log conditional likelihood to get log a more succinct loss component

$$1 - g(x^{(i)})) = 1 - \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$= \frac{e^{-f_w(x^{(i)})}}{1 + e^{-f_w(x^{(i)})}}$$

$$= \frac{1}{1 + e^{f_w(x^{(i)})}}$$

Log(1/(1+exp(-WX))) = - log(1+exp(-WX))

So, we can transform the equation into the following form,

$$J(w) = \lambda \parallel w \parallel^2 - \sum_i \log(1 + e^{-y^{(i)} f_w(x^{(i)})})$$

Loss L(M)

$$L(m) = \log(1 + e^{-m})$$

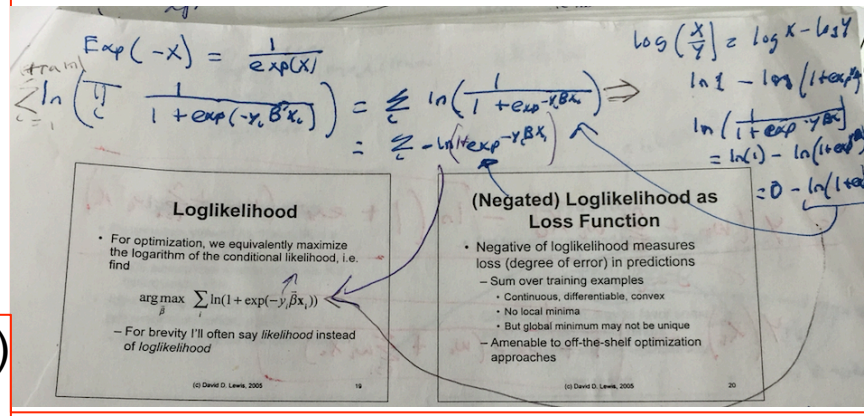
Where m is defined

$$m^i = y^{(i)} f_w(x^{(i)})$$

$$y^{(i)} = \begin{cases} -1 & \text{if } y^{(i)} = 0 \\ 1 & \text{if } y^{(i)} = 1 \end{cases}$$

$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$1 - g(x^{(i)})) = \frac{1}{1 + e^{f_w(x^{(i)})}}$$



Maximize Log loss

This is a maximize version of the log conditional likelihood

$$\hat{\beta} = \text{argmax}_\beta \; LCL - \mu||\beta||_2^2$$

Third, let's see log loss, which is equivalent to the cross entropy loss function used to train a logistic regression model

$$J(w) = \lambda \parallel w \parallel^2 + \sum_i y^i \log g_w(x^{(i)}) + (1 - y^i)(\log 1 - g(x^{(i)})), y^i \in (0, 1) \qquad (14.8)$$

$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$f_w(x^{(i)}) = w^T x^{(i)}$$

Because

$$\begin{aligned} 1 - g(x^{(i)})) &= 1 - \frac{1}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{e^{-f_w(x^{(i)})}}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{1}{1 + e^{f_w(x^{(i)})}} \end{aligned}$$

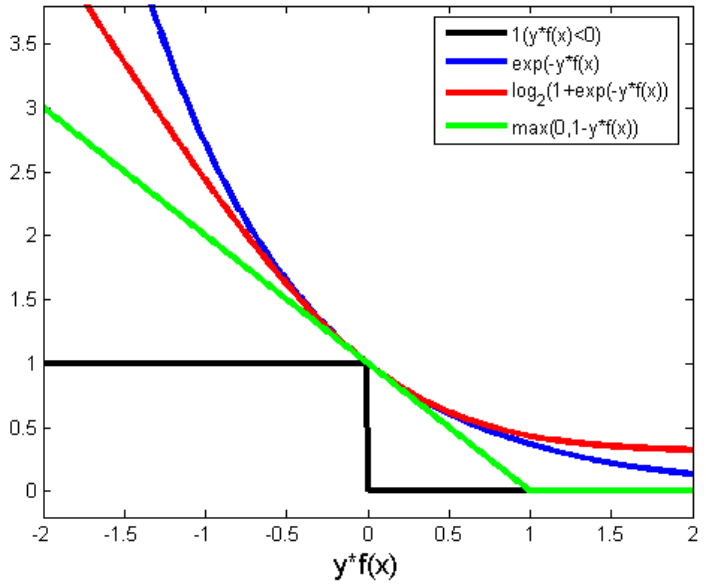
So, we can transform the equation into the following form,

$$J(w) = \lambda \parallel w \parallel^2 + \sum_i \log 1 + e^{-y^{\bar{(i)}} f_w(x^{(i)})}$$

Loss L(M) $L(m) = \log 1 + e^{-m}$

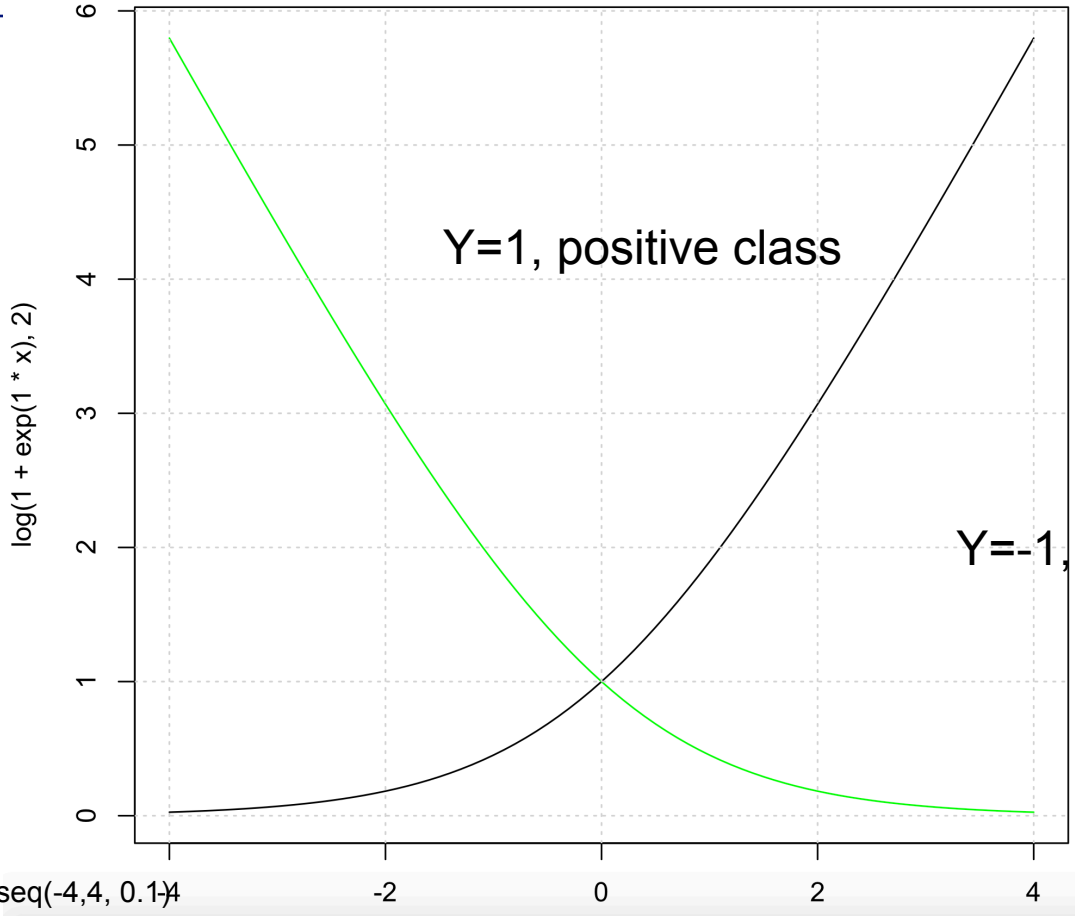
Where m is defined $m^i = y^{\bar{(i)}} f_w(x^{(i)})$

$$y^{\bar{(i)}} = \begin{cases} -1 & \text{if } y^{(i)} = 0 \\ 1 & \text{if } y^{(i)} = 1 \end{cases}$$




```
 log(1 + exp(-1*2),2)   #actual class is positive
[1] 0.1831184           # LOSS

> log(1 + exp(-1*5),2) #actual class is positive
[1] 0.0096882           #Small loss

> log(1 + exp(1*2),2) #actual class is NEG
[1] 3.068508          #BIG loss
> log(1 + exp(1*-2),2) )   #actual class is NEG
[1] 0.1831184           #Small loss
>
```

Logistic loss term




```
> x = seq(-4,4, 0.1)
> x
 [1] -4.0 -3.9 -3.8 -3.7 -3.6 -3.5 -3.4 -3.3 -3.2 -3.1 -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4
[18] -2.3 -2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7
[35] -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0
[52]  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7
[69]  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.0
> plot(x, log(1 + exp(1*x),2))
> plot(x, log(1 + exp(1*x),2), type="l")
> lines(x, log(1 + exp(1*x),2), type="l", col="green")
> grid
```

4: Squared Error loss term; 5: Exponential Loss term

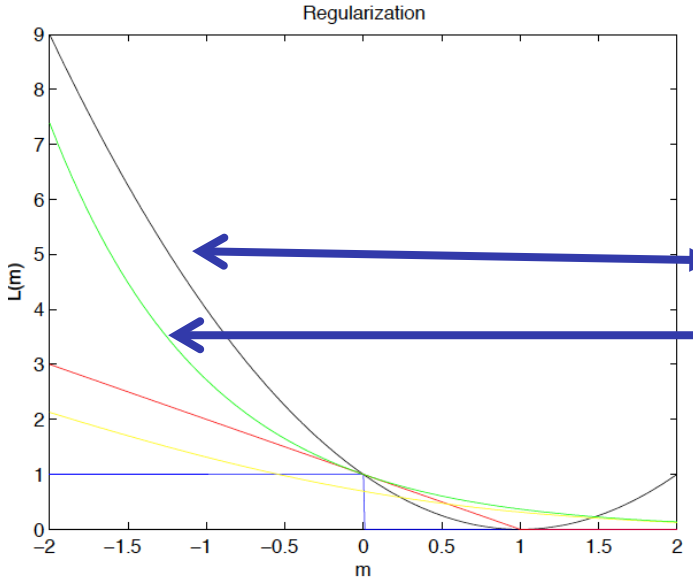
Fourth, let's see Linear Regression, which have a squared loss term. We will use L_2 to scribe the squared loss term. We can see from fig(1) the squared term is much higher than the Gold Stand,so it is a bad function.

$$L_2(m) = (f_w(x) - y)^2 = (m-1)^2 \tag{14.11}$$

Later on, we'll look at a learning algorithm called boosting. It can be seen as a greedy optimization of the exponential loss term,

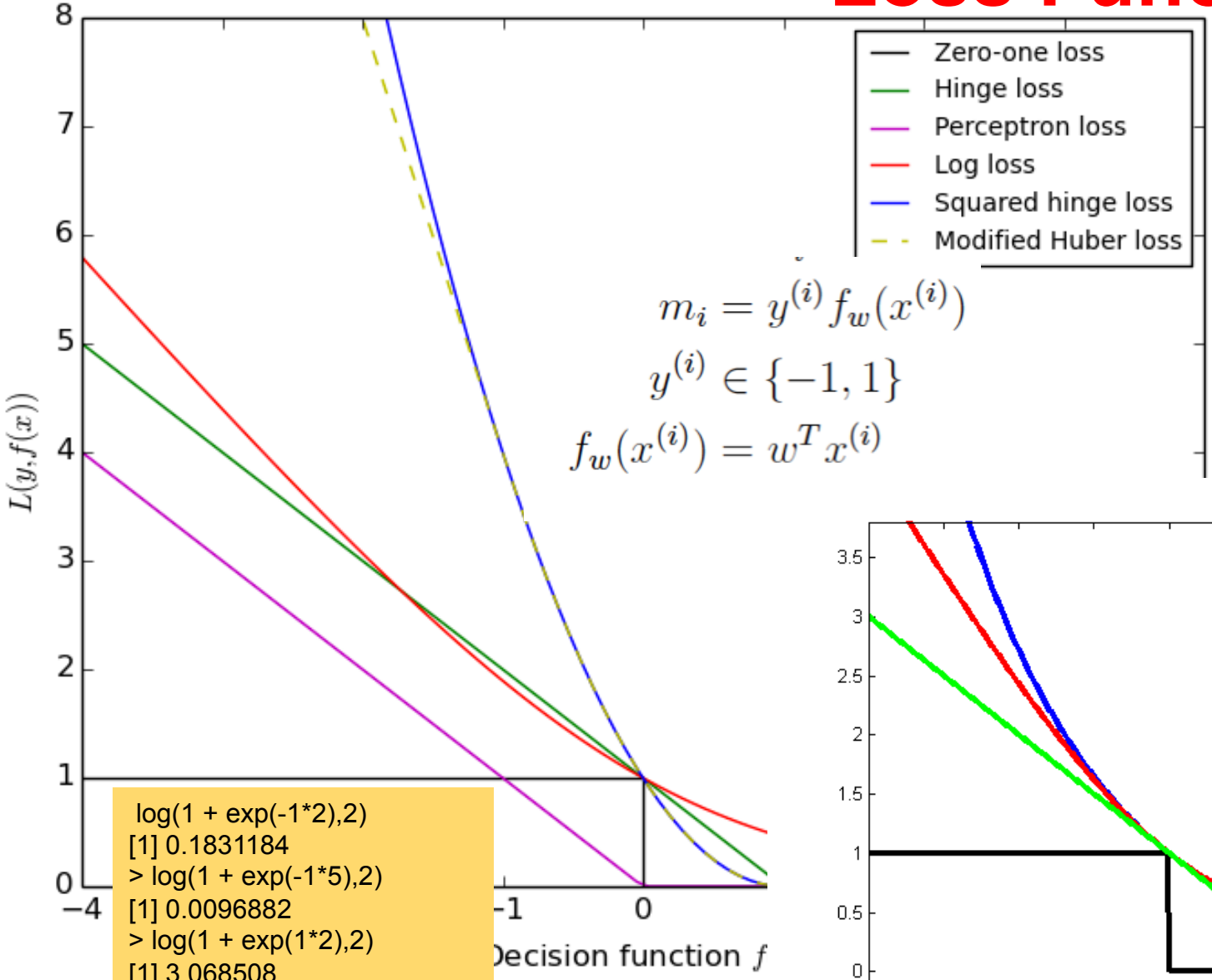
$$J(w) = \lambda R(w) + \sum_i \exp(-y^{(i)} f_w(x^{(i)})) \tag{14.12}$$

$$L_{exp}(m_i) = \exp(-m_i(w)) \tag{14.13}$$

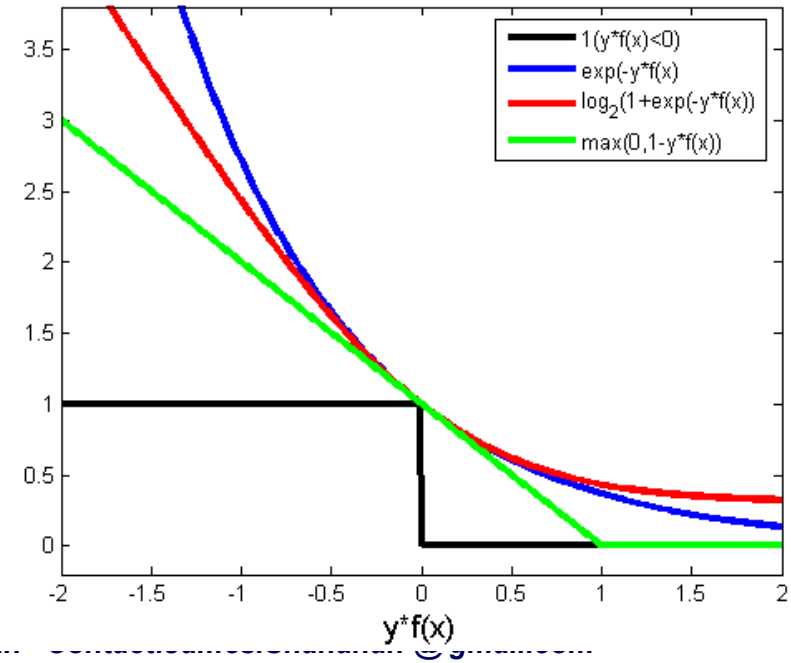


the black line is for squared loss function
the green line is for boosting loss
the red line is for Hinge loss (y*WX)
the yellow line is for logistic loss
the blue line is for 0-1 loss function

Loss Functions

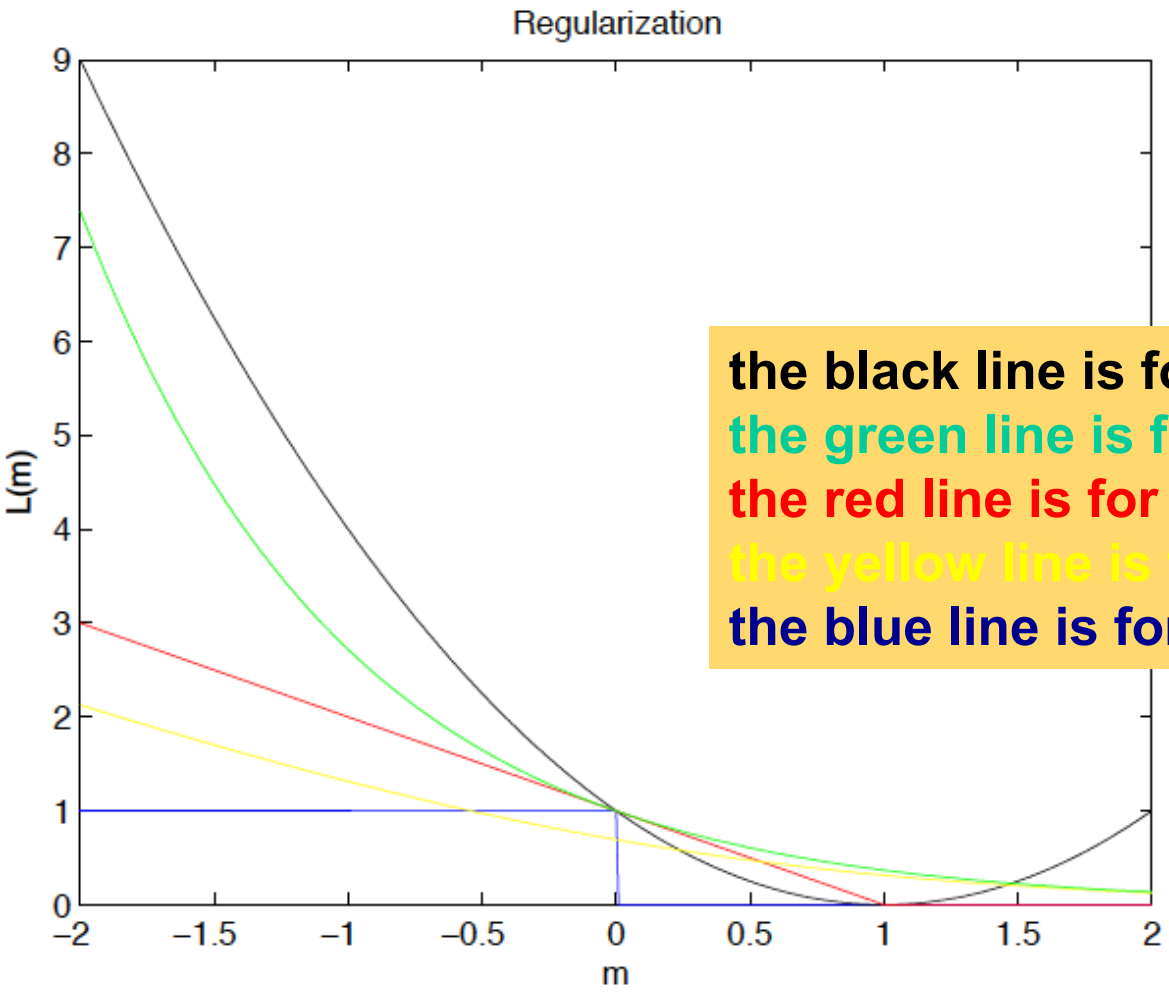


$$m_i = y^{(i)} f_w(x^{(i)})$$
$$y^{(i)} \in \{-1, 1\}$$
$$f_w(x^{(i)}) = w^T x^{(i)}$$




```
 log(1 + exp(-1*2),2)
[1] 0.1831184
> log(1 + exp(-1*5),2)
[1] 0.0096882
> log(1 + exp(1*2),2)
[1] 3.068508
> log(1 + exp(1*-2),2)
[1] 0.1831184
>
```

Loss Functions



Let W = (0,0,....)

Repeat

For j in 0..n #each variable

For i in 1..m #each example

$$W_{j,t+1} = W_{j,t} + \alpha * (y - p) X_j$$

until convergence (i.e., no big changes in W or error)

the black line is for squared loss function
the green line is for boosting loss
the red line is for Hinge loss (y*WX)
the yellow line is for logistic loss
the blue line is for 0-1 loss function

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term ($R(w)$ expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

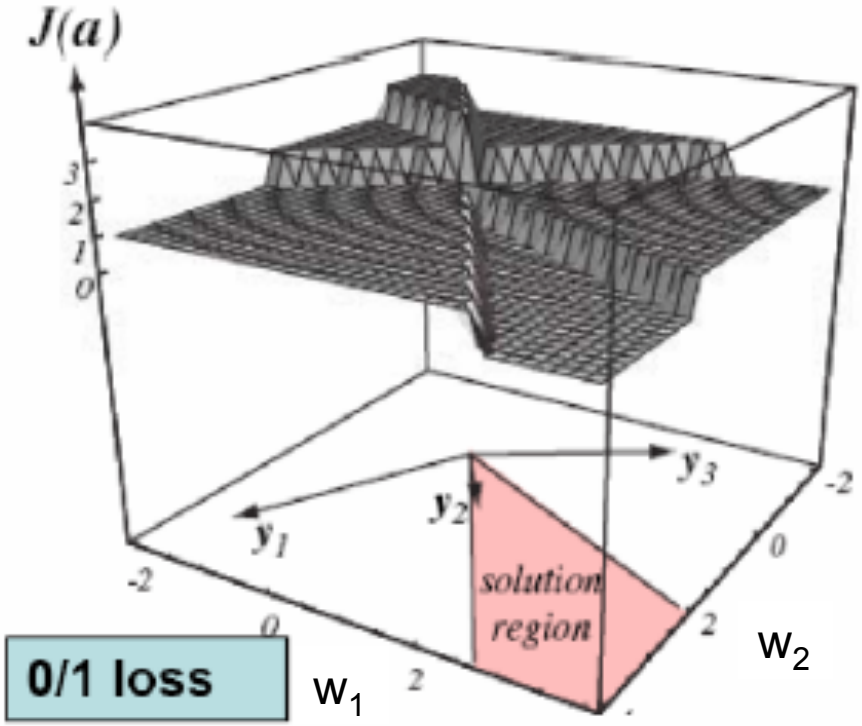
$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

0-1 Loss function for Classification

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N}\sum_{i=1}^{N} L(\mathrm{sgn}(w \cdot x_i), y_i)$

  L(y',y) = 0 when y'=y, otherwise L(y',y)=1
- Does not produce useful gradient since the surface of J is flat

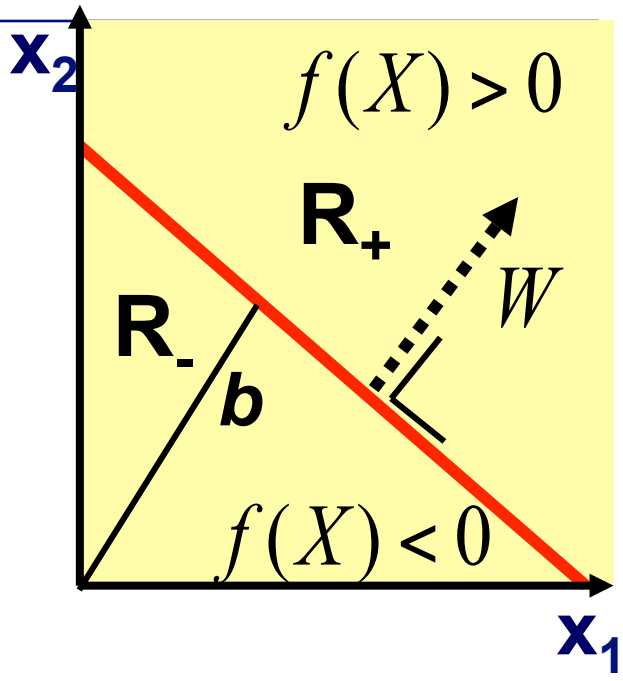


Count the number of mistakes

Slide Improvisation

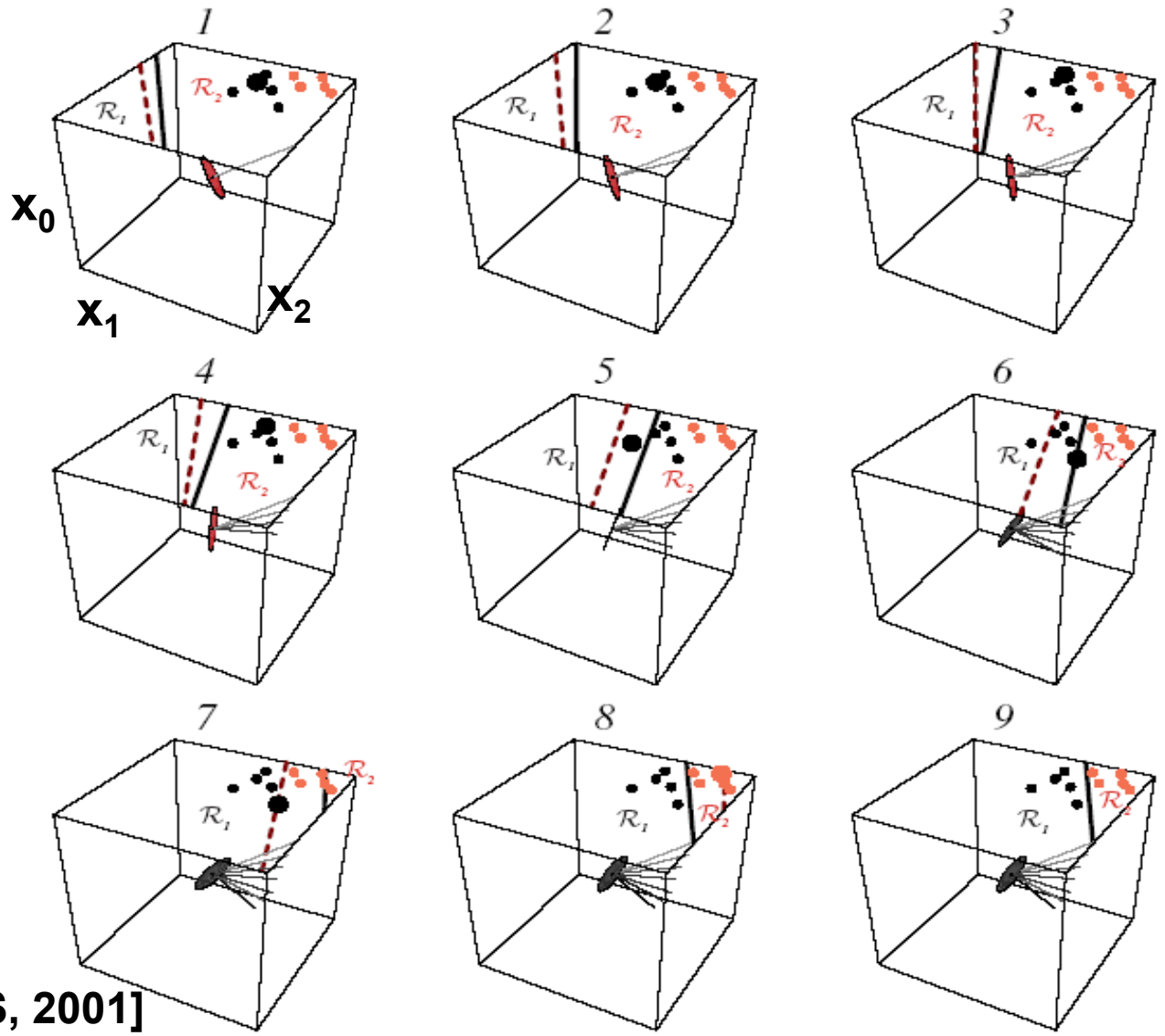
Summary: Linear Discriminants

$$Class(X) = sign(< W, X > + b)$$



- A linear discriminant function is linear in the components of *X*
- A linear discriminant function divides the feature space by a hyperplane decision surface H_0
- The orientation of the surface H_0 is determined by the normal vector *W* and H_0's location is determined by the *bias*
- Talk about polynomial/generalized discriminant functions later

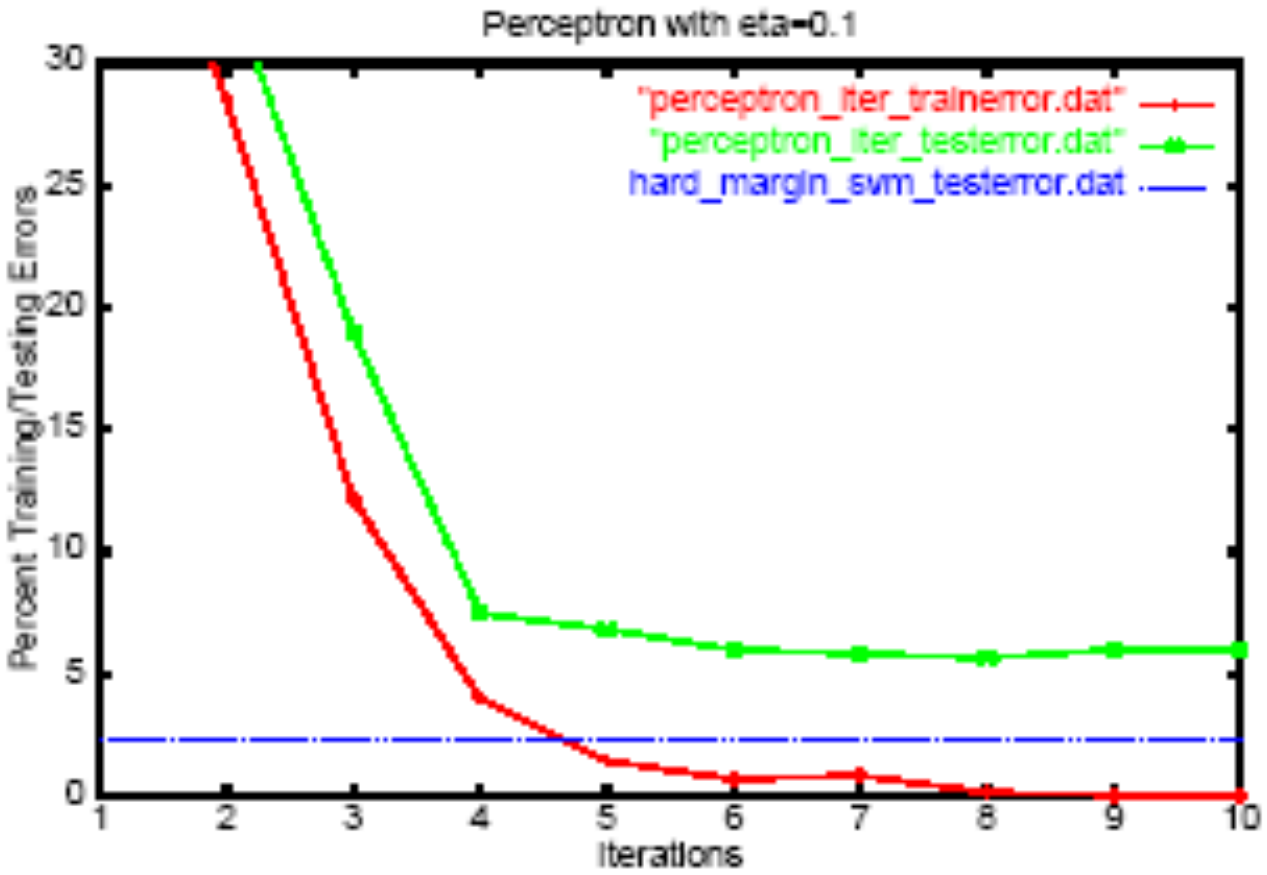
Two-class Perceptron Learning



Source: [DHS, 2001]

Perceptron Learning: Text Example

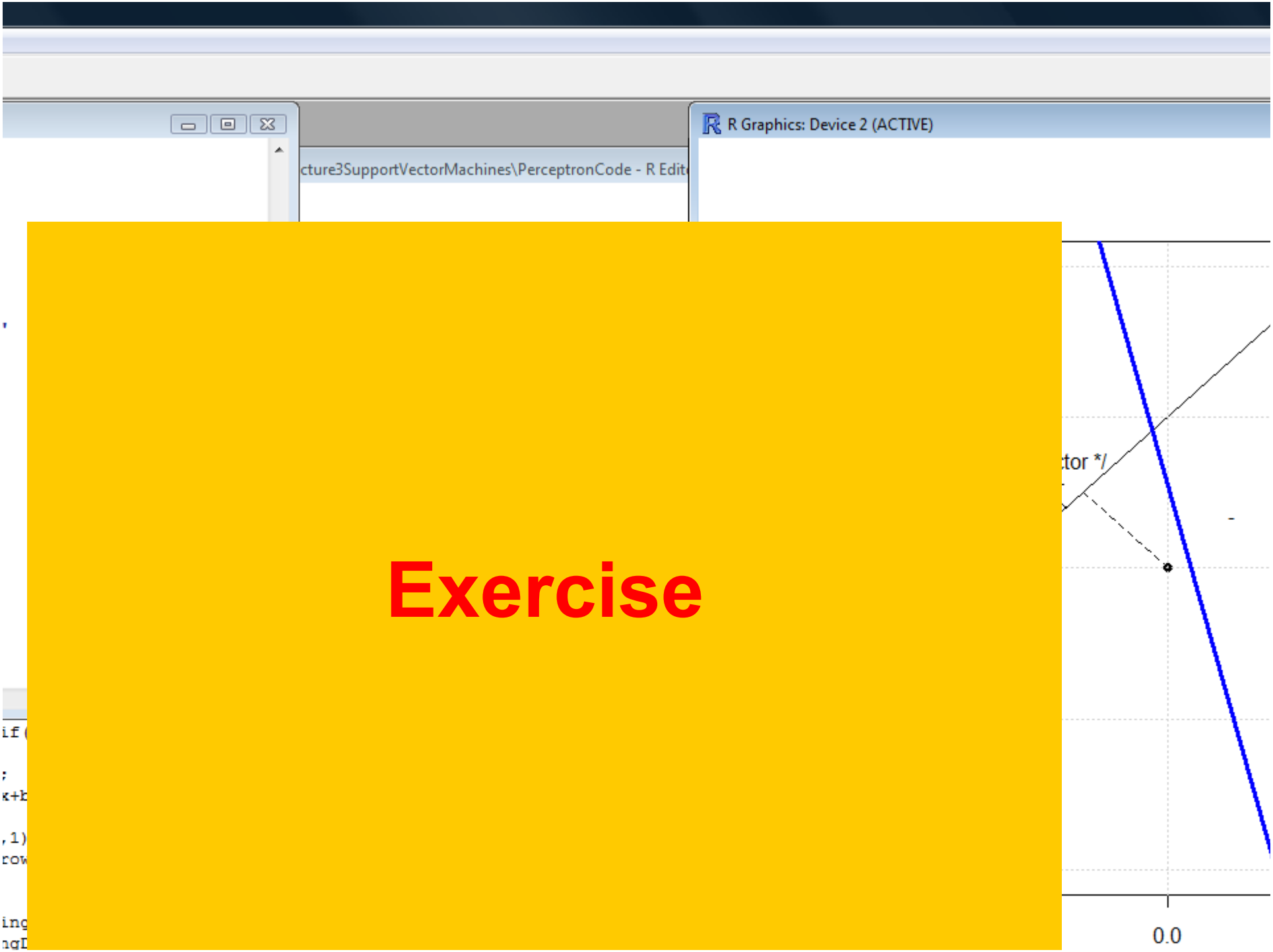
Experiment: Perceptron for Text Classification



Train on 1000 pos / 1000 neg examples for "acq" (Reuters-21578).

[Source: http://www.cs.cornell.edu/Courses/CS678/2003sp/slides/perceptron_4up.pdf]

Exercise


R Graphics: Device 2 (ACTIVE)
cture3SupportVectorMachines\PerceptronCode - R Edit
tor */
0.0

- **Tahat was the perceptron**

- **Next look at distribution verion of this sequential alogothm**

- **Distributed Perceptron**

Supervised Machine Learning Outline (Part 2)

- **Loss Functions (10)**
- **General framework for gradient descent Notebook (10) [May merge section 1 and 2 for 10 minutes overall**
- **Logistic Regression at scale (15)**
  - Introduction (10)
  - Distributed Logistic Regression
  - Related approaches and extensions and metrics (5) → (10)
- **Perceptron (20)**
  - Distributed Perceptron
  - Case study
- **SVMs**
  - SVMs in primal and dual form
  - SVMs (15)
  - Stochastic Gradient descent based SVM at scale(10)
  - Adatron 10
  - Distributed adatron enabling SVMs at scale (10)
- **Case Studies (Ng paper on multi-cores)**
- **Sync time**

V2

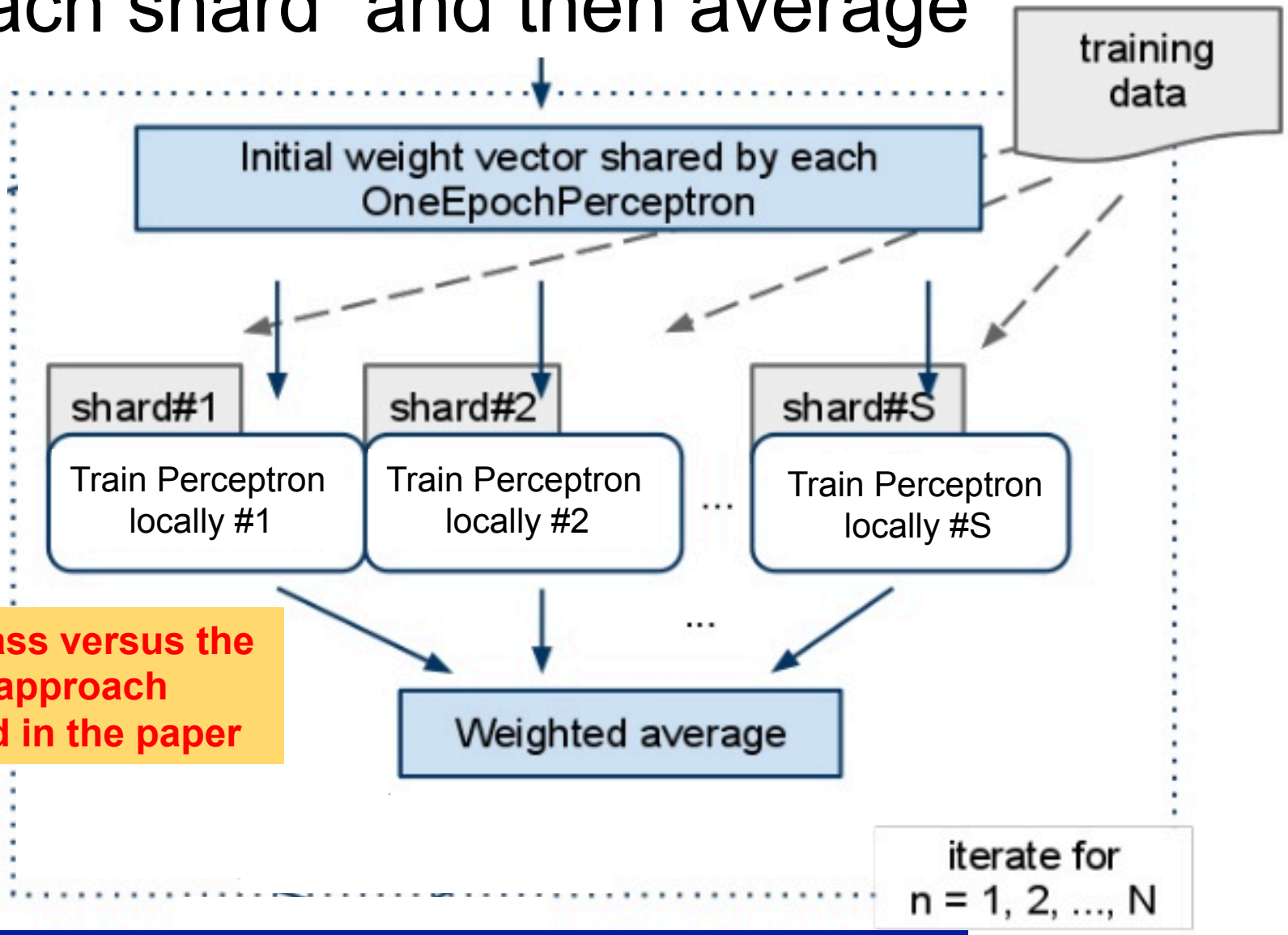
Distributed Perceptron

- The distributed Perceptron training algorithm preserves convergence properties, thus guaranties same accuracy performances as the serial Perceptron (mistake driven over all data).
- The execution of the modules applied to specific NLP tasks can be demonstrated and tested via an interactive web interface that allows the user to inspect the status and structure of the cluster and interact with the MapReduce jobs.

Distributed Perceptron References

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

Hadoop Perceptron: learn locally on each shard and then average



Single pass versus the iterative approach proposed in the paper

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

Hadoop Perceptron: learn locally on each shard and then average

Parameter mixing (averaging) fails (1/6)

- Parameter mixing:
  Train *S* perceptrons with *S* shards of the training data,
  Take a weighted average of their weights

PerceptronParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)

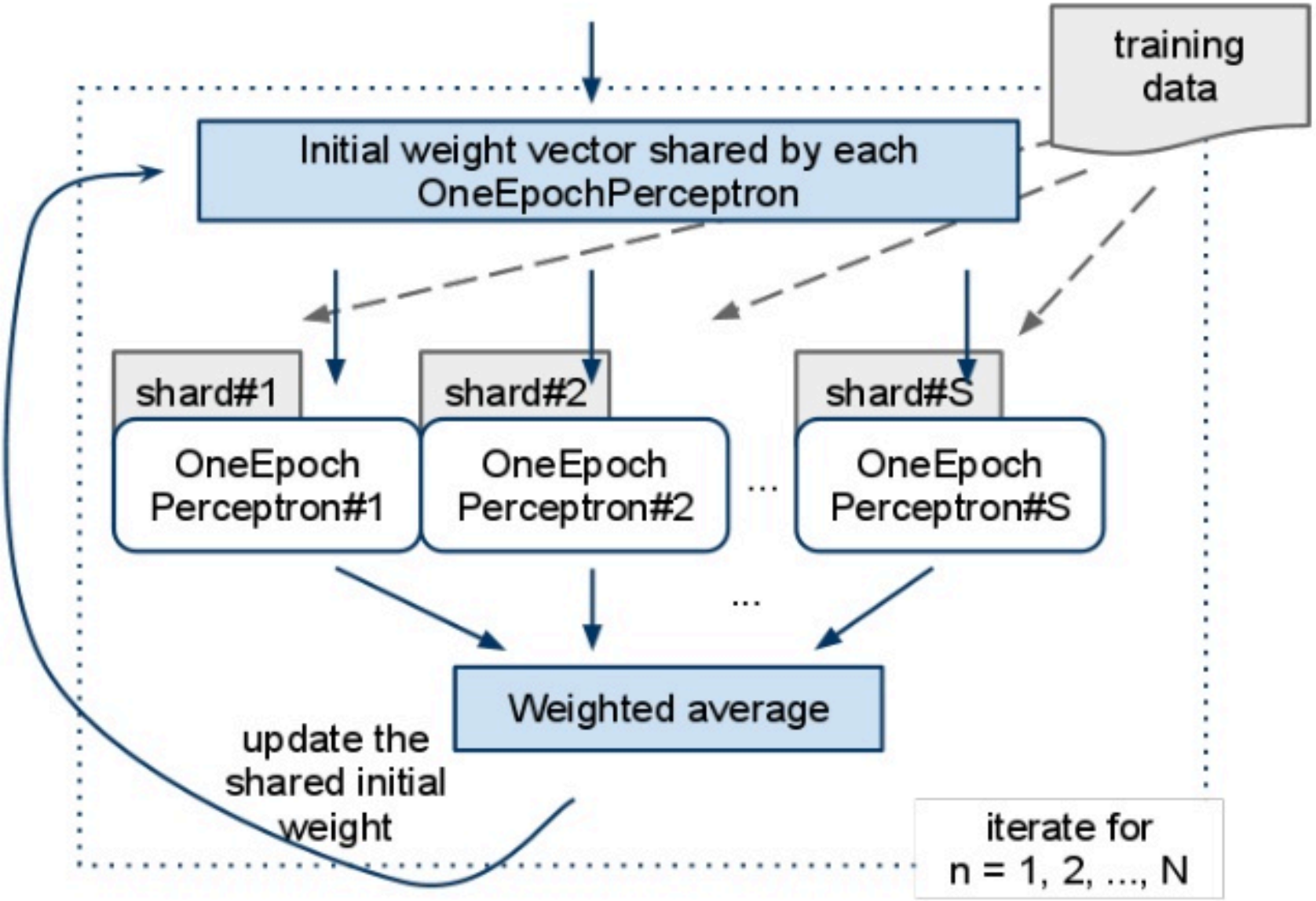
1. Shard $\mathcal{T}$ into S pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$
2. $\mathbf{w}^{(i)} = \text{Perceptron}(\mathcal{T}_i)$ †
3. $\mathbf{w} = \sum_i \mu_i \mathbf{w}^{(i)}$ ‡
4. return $\mathbf{w}$

Figure 2: Distributed perceptron using a parameter mixing strategy. † Each $\mathbf{w}^{(i)}$ is computed in parallel. ‡ $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_S\}, \forall \mu_i \in \boldsymbol{\mu} : \mu_i \geq 0$ and $\sum_i \mu_i = 1$.

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

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

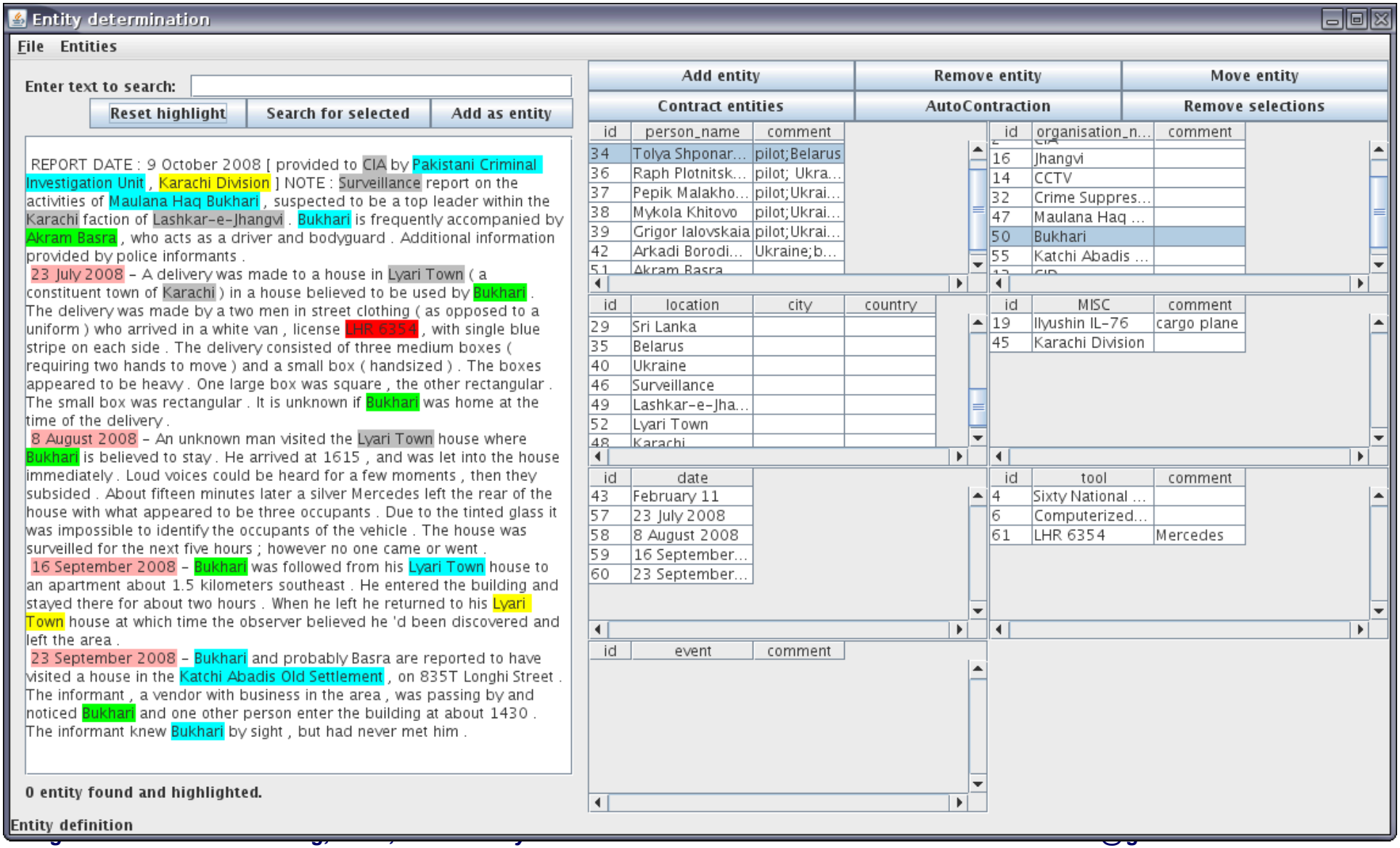
Iterative parameter mixing



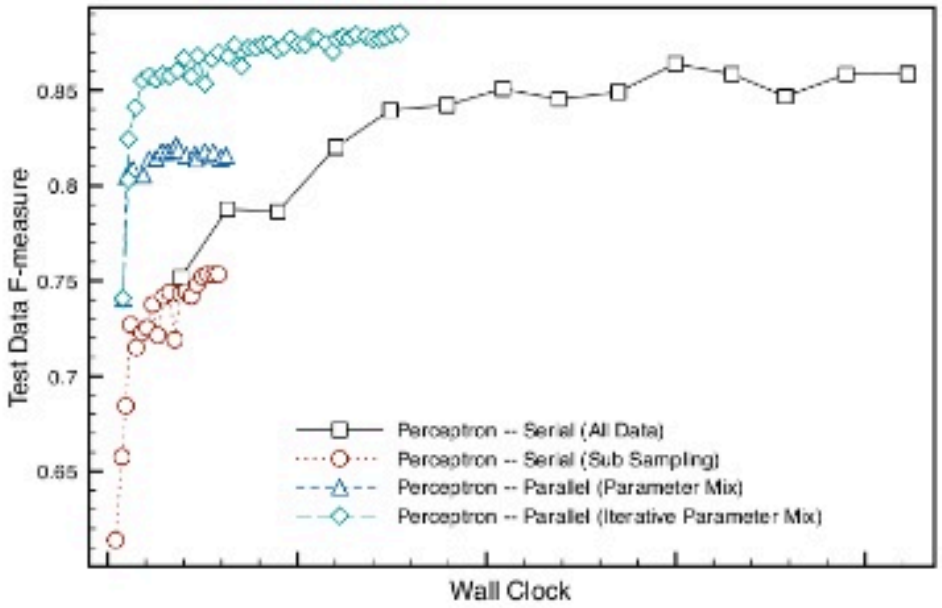
Compare 4 systems

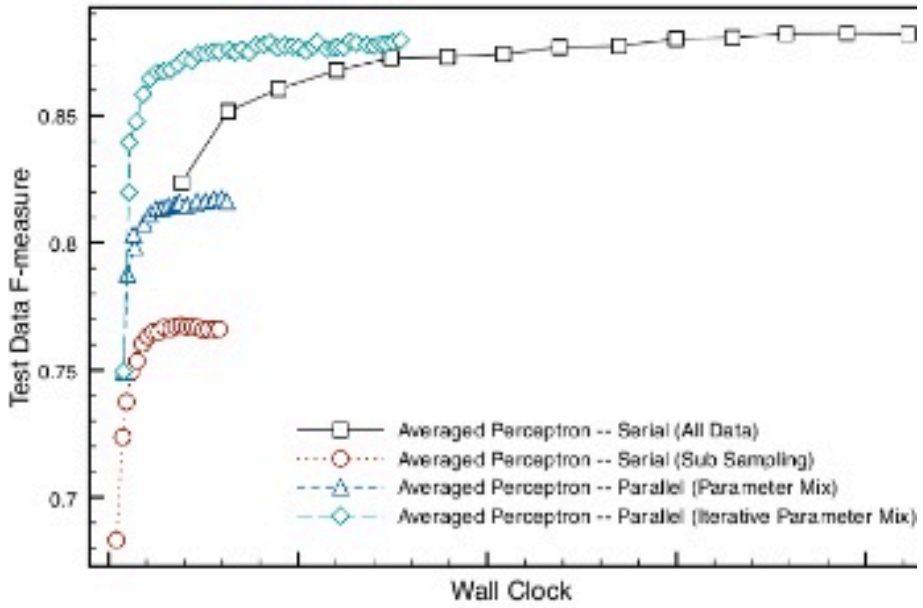
- **1. Serial (All Data): This is the classifier returned if trained serially on all the available data.**
  - On a single computer for example (Mistake driven)
- **2. Serial (Sub Sampling): Shard the data, select one shard randomly and train serially.**
- **3. Parallel (Parameter Mix): Learn locally on each shard: Parallel strategy with uniform mixing.**
- **4. Parallel (Iterative Parameter Mix): Parallel strategy discussed in Section 4.2 with uniform mixing (Section 5.1 looks at mixing strategies).**

Named entity recognition (NER)



NER experiments: faster & better, close to averaged perceptrons

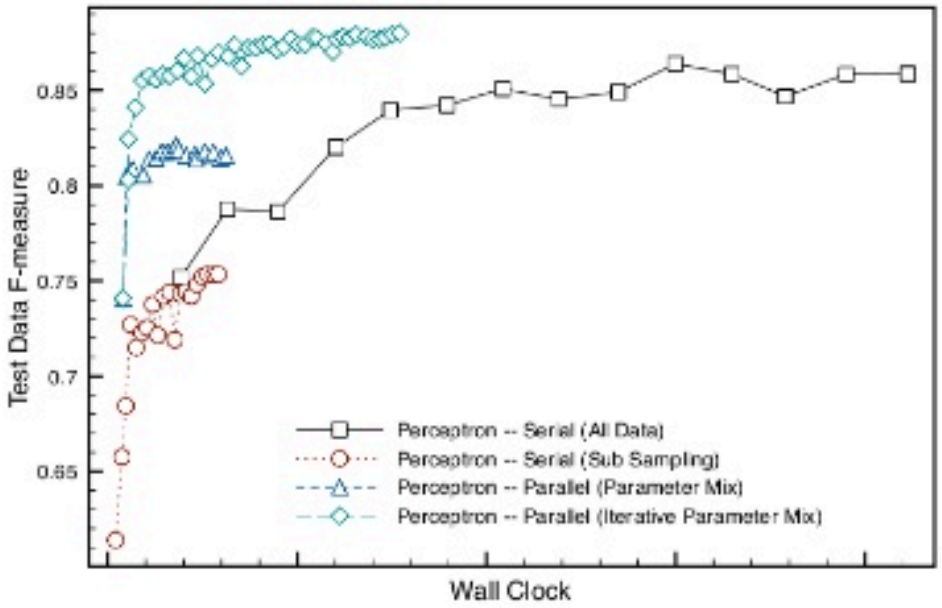


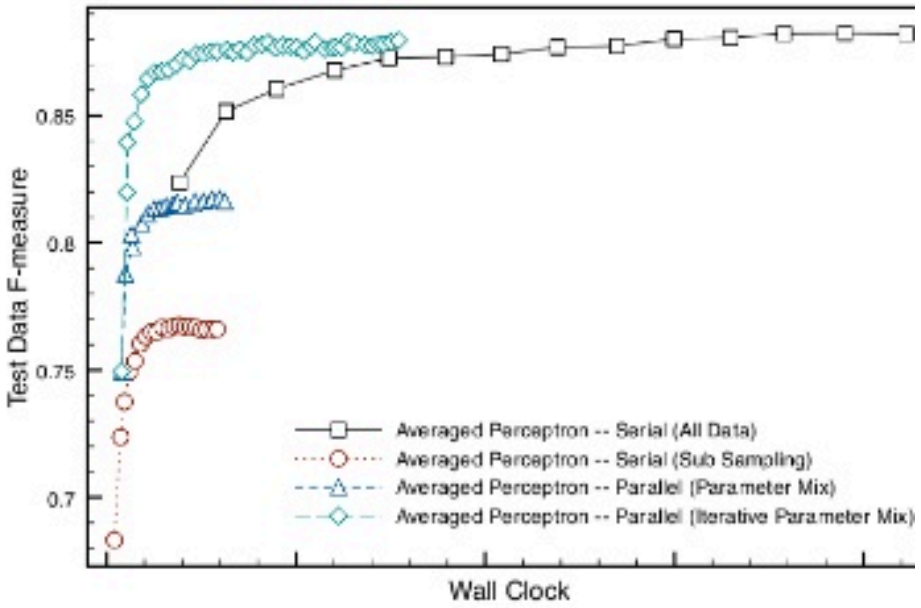


	Reg. Perceptron F-measure	Avg. Perceptron F-measure
Serial (All Data)	85.8	88.2
Serial (Sub Sampling)	75.3	76.6
Parallel (Parameter Mix)	81.5	81.6
Parallel (Iterative Parameter Mix)	87.9	88.1

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

NER experiments: faster & better, close to averaged perceptrons





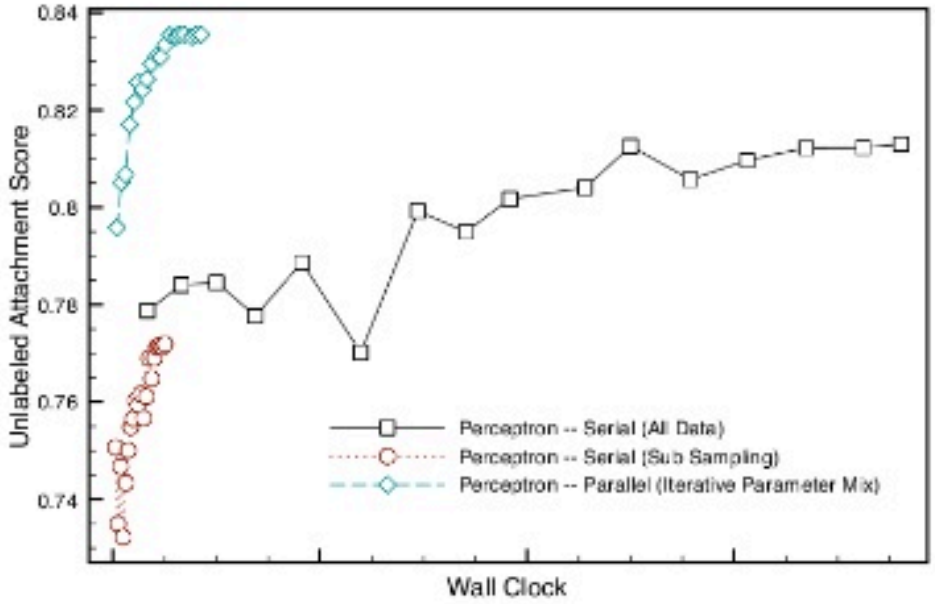
Iterative mixing is **faster and more accurate** than serial. (non-averaged case)

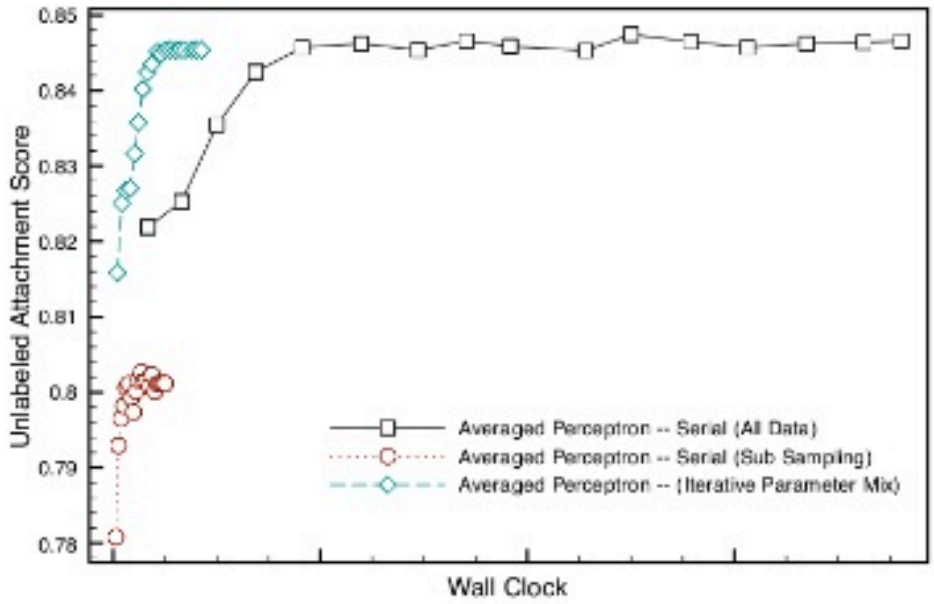
Iterative mixing is **faster** and similarly accurate to serial. (averaged case)

Serial (Sub Sampling)	
Parallel (Parameter Mix)	
Parallel (Iterative Parameter Mix)	87.9	88.1

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

Dependency parsing experiments: similar improvements





	Reg. Perceptron Unlabeled Attachment Score	Avg. Perceptron Unlabeled Attachment Score
Serial (All Data)	81.3	84.7
Serial (Sub Sampling)	77.2	80.1
Parallel (Iterative Parameter Mix)	83.5	84.5

Figure 5: Dependency Parsing experiments. Upper figures plot test data unlabeled attachment score versus wall

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

Different shard size: the more shards, the slower convergence

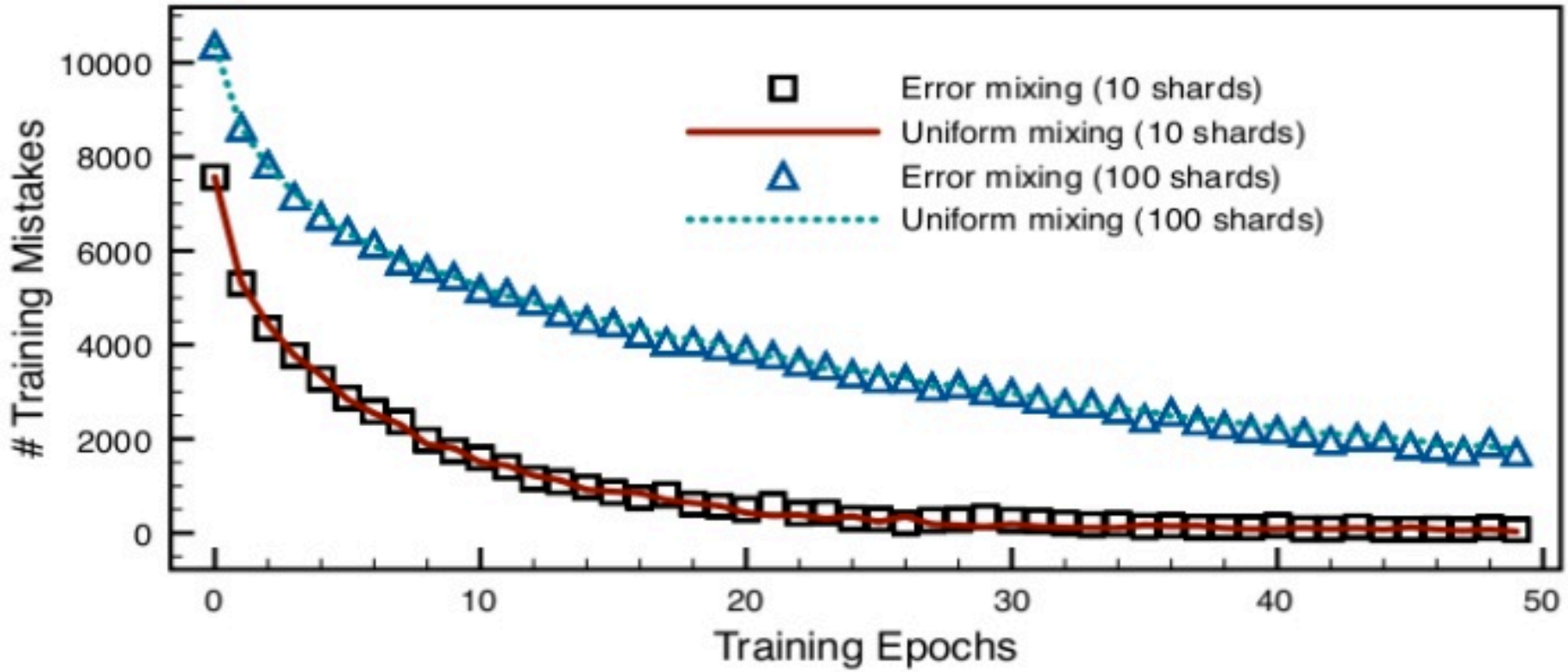


Figure 6: Training errors per epoch for different shard size and parameter mixing strategies.

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

Different shard size: the more shards, the slower convergence

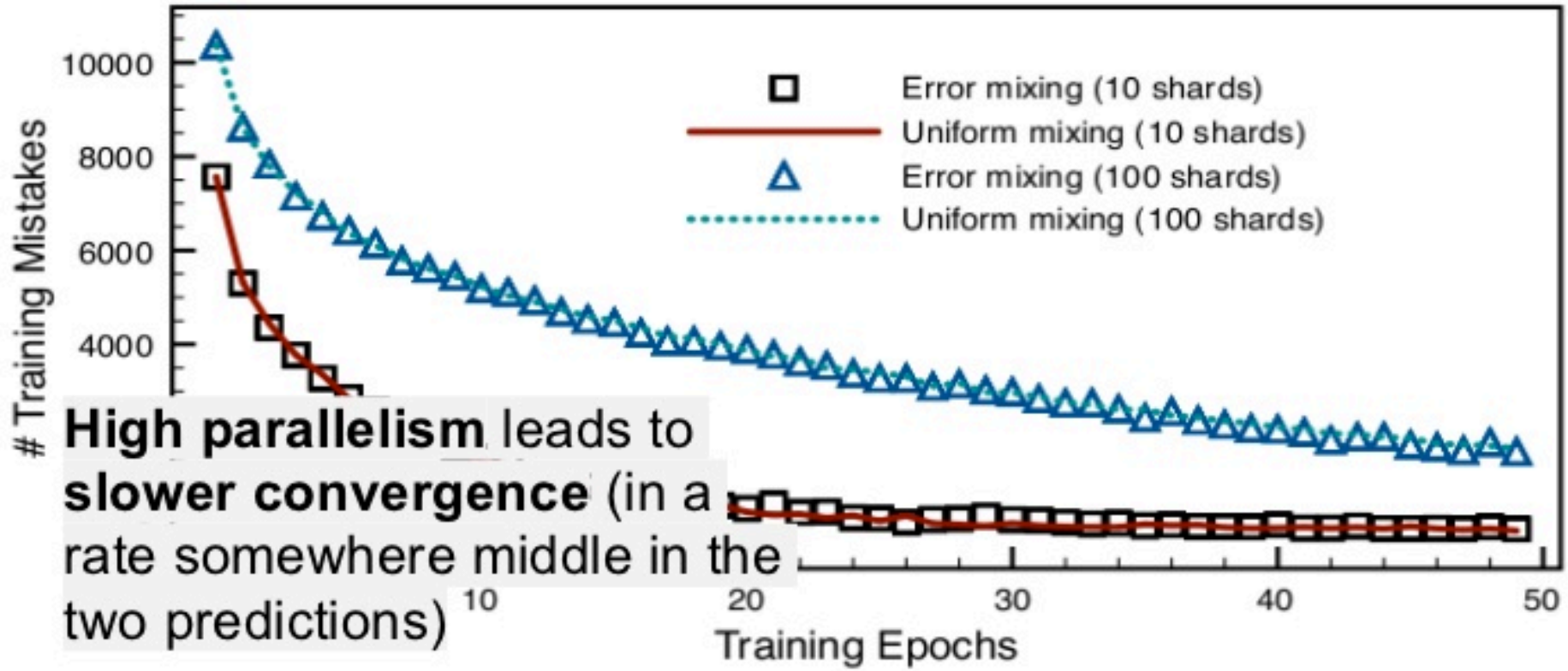


Figure 6: Training errors per epoch for different shard size and parameter mixing strategies.

Distributed Training Strategies for the Structured Perceptron
by R. McDonald, K. Hall & G. Mann, 2010

Conclusions

- Distributed training of the structured perceptron via simple parameter mixing strategies
  - Guaranteed to converge and separate the data (if separable)
  - Results in fast and accurate classifiers
- Trade-off between high parallelism and slow convergence

- (+ applicable to online passive-aggressive algorithm)

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term ($R(w)$ expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \qquad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \qquad (14.2)$$

f_w

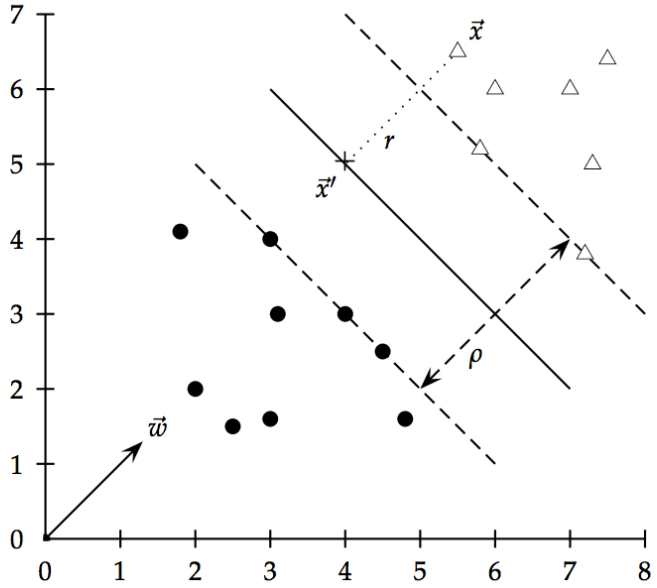
Focus only on the regularization term
E.g., HARD SVMs

Derive Perpendicular Distance from a point to a hyperplane

What is the Euclidean distance from a point $\vec{x}$ to the decision boundary? In Figure 15.3, we denote by r this distance. We know that the shortest distance between a point and a hyperplane is perpendicular to the plane, and hence, parallel to $\vec{w}$. A unit vector in this direction is $\vec{w}/|\vec{w}|$. The dotted line in the diagram is then a translation of the vector $r\vec{w}/|\vec{w}|$. Let us label the point on the hyperplane closest to $\vec{x}$ as $\vec{x}'$. Then:

$$\vec{x}' = \vec{x} - yr\frac{\vec{w}}{|\vec{w}|}$$

where multiplying by y just changes the sign for the two cases of $\vec{x}$ being on either side of the decision surface. Moreover, $\vec{x}'$ lies on the decision boundary



► **Figure 15.3** The geometric margin of a point (r) and a decision boundary (ρ).

and so satisfies $\vec{w}^{\mathrm{T}}\vec{x}' + b = 0$. Hence:

$$\vec{w}^{\mathrm{T}}\left(\vec{x} - yr\frac{\vec{w}}{|\vec{w}|}\right) + b = 0 \tag{15.3}$$

Solving for r gives:[2]

$$r = y\frac{\vec{w}^{\mathrm{T}}\vec{x} + b}{|\vec{w}|} \tag{15.4}$$

2. Recall that $|\vec{w}| = \sqrt{\vec{w}^{\mathrm{T}}\vec{w}}$.

http://nlp.stanford.edu/IR-book/pdf/15svm.pdf

Margins

16.2.1 Classification margin

As discussed in Section 5.3.3, the following quantity represents the signed distance between instance x and the separating hyperplane for a linear threshold model:

$$\delta_{\mathbf{w}}(x) = \frac{\mathbf{w} \circ \mathbf{a}(x)}{\|\mathbf{w}_{1:n}\|} \tag{16.2}$$

with the sign indicating whether the point represented by $\mathbf{a}_{1:n}(x)$ lies on the positive or negative side of the hyperplane. The following slightly modified form of this distance, negated for instances of class 0 by introducing the $c\_(x)$ multiplier:

$$\gamma_{\mathbf{w}}(x) = c\_(x)\frac{\mathbf{w} \circ \mathbf{a}(x)}{\|\mathbf{w}_{1:n}\|} \tag{16.3}$$

is called the *geometric margin* of the parameter vector $\mathbf{w}$ with respect to instance x. This is positive if x lies on the correct side of the decision hyperplane (is classified correctly), negative if it lies on the wrong side of the decision hyperplane, and 0 if it lies exactly on the decision hyperplane. With the normalizing denominator skipped, it is referred to as the *functional margin* of the parameter vector $\mathbf{w}$ with respect to instance x:

$$\hat{\gamma}_{\mathbf{w}}(x) = c\_(x)\mathbf{w} \circ \mathbf{a}(x) = \|\mathbf{w}_{1:n}\|\gamma_{\mathbf{w}}(x) \tag{16.4}$$

The minimum values of the geometric and functional margins of the parameter vector $\mathbf{w}$ with respect to all training instances are called, respectively, the geometric and functional margins of $\mathbf{w}$ with respect to the training set:

$$\gamma_{\mathbf{w}}(T) = \min_{x \in T} \gamma_{\mathbf{w}}(x) \tag{16.5}$$

$$\hat{\gamma}_{\mathbf{w}}(T) = \min_{x \in T} \hat{\gamma}_{\mathbf{w}}(x) \tag{16.6}$$

If the hyperplane represented by $\mathbf{w}$ correctly separates all training instances of different classes, we have $\gamma_{\mathbf{w}}(T) > 0$ and $\hat{\gamma}_{\mathbf{w}}(T) > 0$.

Geometric margin (i.e., Class times perpendicular distance)

Functional margin (dot product between W and X I,e., <W,X>

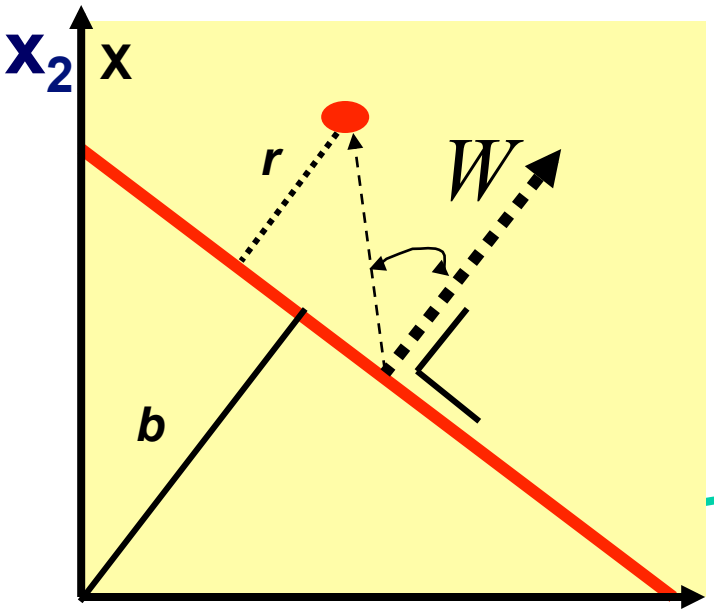
[Data Mining Algorithms Explained Using R, Page 457, By Pawel Cichosz]

Functional Margin

Take any linear separator (e.g., a logistic regression or perceptron model)

- **Functional margin of an example *(X,y)* WRT a hyperplane *(W,b)***

$$\langle W, X \rangle + b$$



$$w_1 x_1 + w_2 x_2 + b = 0$$

$$\left(\sum_{i=1}^{n} w_i x_i \right) + b = 0$$

$$\langle W, X \rangle + b = 0$$

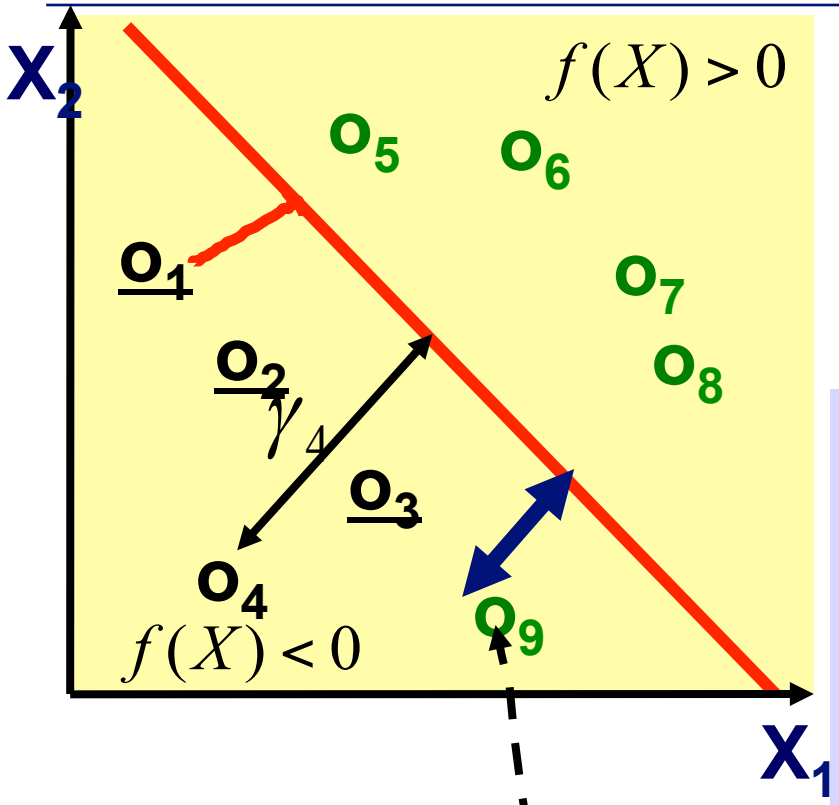
Functional Margin for a Training Set

- Define the margin of an example *i* with respect to a hyperplane *(W, b)* to be the quantity $\gamma_{fun,i}$:

$$\mathbf{w}\cdot\mathbf{x}_i + b \geq +\gamma_{Fun} \quad \forall\ y_i = +1,$$
$$\mathbf{w}\cdot\mathbf{x}_i + b \leq -\gamma_{Fun} \quad \forall\ y_i = -1,$$
$$\text{or equivalently, } y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq \gamma_{Fun} \quad \forall\ i = 1,\ldots,L.$$

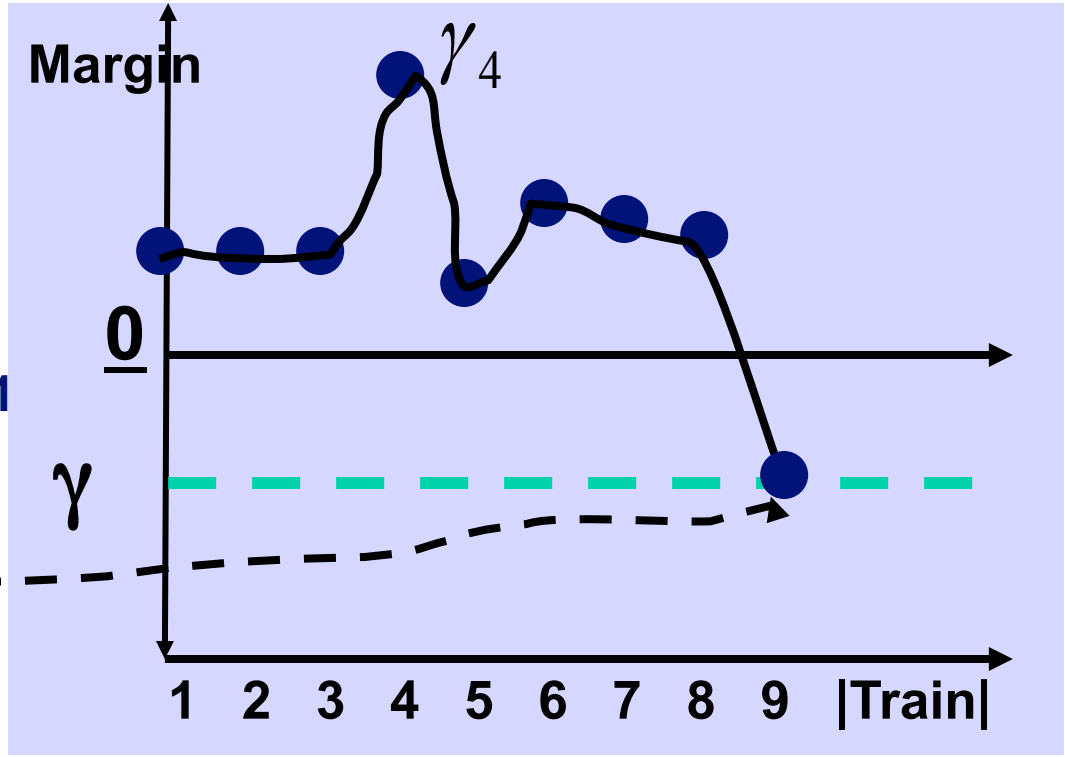
- If $\gamma_{fun,i} > 0$ then correct classification of example (X_i, y_i).
- The margin distribution is the distribution of margins of the examples in a training dataset S.
- The minimum of the margin distribution corresponds to the margin of the hyperplane (W, b) w.r.t S.

Margin and Margin Distribution



Take any linear separator (e.g., a logistic regression or perceptron model)

$$\gamma_{W,b} = \min(y_i(\mathbf{w} \cdot \mathbf{x}_i + b)) \ \forall \ i = 1,\ldots,L.$$



Negative Margin for example o9

- **SVMs: finding the maximal margin hyperplane**

Maximum Margin Classifier: Take 1

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**

$$\underset{\gamma_{Fun},W,b}{\text{Maximize}}\ \gamma_{Fun}$$

$$\text{subject to} \quad \mathbf{w}\cdot\mathbf{x}_i + b \geq +\gamma_{Fun} \quad \forall\ y_i = +1,$$

$$\mathbf{w}\cdot\mathbf{x}_i + b \leq -\gamma_{Fun} \quad \forall\ y_i = -1,$$

$$\text{or equivalently,}\ y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq \gamma_{Fun} \quad \forall\ i = 1,\ldots,L.$$

Hyperplane (W,b) can take any value and makes the margin γ_{fun} unbounded as W is unbounded E.g.,

$$y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) - 1 = \gamma_{Fun,i}$$

$$y_i\left(10 * \mathbf{w}\cdot\mathbf{x}_i + 10 * b\right) - 1 = \gamma_{Fun,i}$$

Geometric Margin

- **Functional margin has a scaling problem**

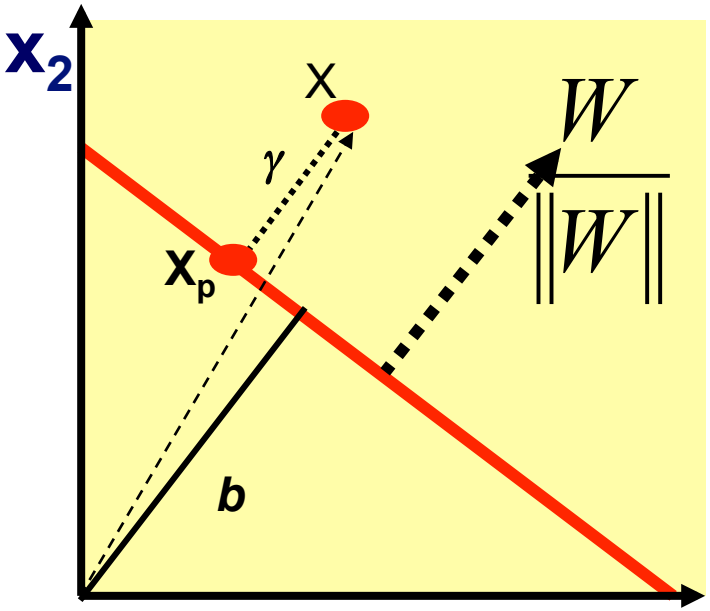
$$y_i\left(\mathbf{w}\cdot\mathbf{x}_i+b\right)=\gamma_{Fun,i}$$

$$y_i\left(10*\mathbf{w}\cdot\mathbf{x}_i+10*b\right)=10*\gamma_{Fun,i}$$

- **Avoid this by normalizing the hyperplane**

$$y_i\left(\frac{\mathbf{w}\cdot\mathbf{x}_i}{\|W\|}+\frac{b}{\|W\|}\right)=\gamma_{Geo,i}$$

Derive Geometric Margin



$$X_P = X - \gamma \frac{W}{\|W\|}$$

$W^T X_P + b = 0$ (X_p is the projection X on H)

$$W^T \left(X - \gamma \frac{W}{\|W\|} \right) + b = 0$$

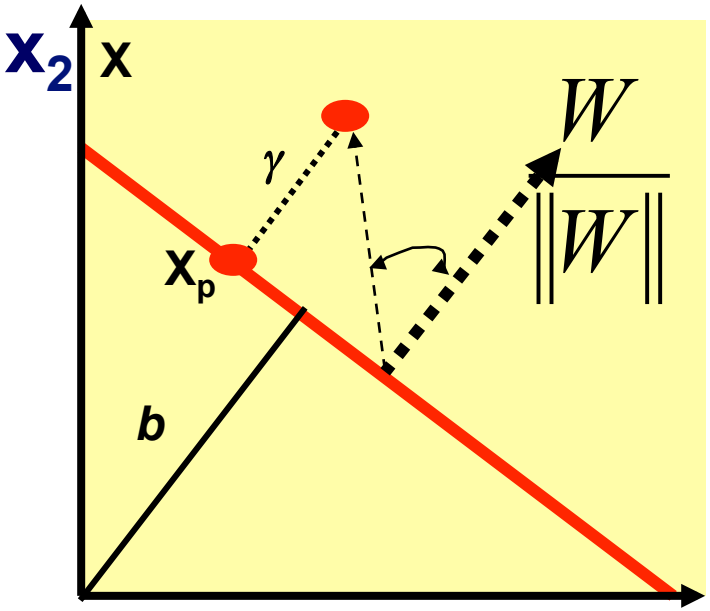
solve for γ

$$\gamma \frac{W^T W}{\|W\|} = W^T X + b$$

$$\gamma \|W\| = W^T X + b$$

$$\gamma = \frac{W^T X}{\|W\|} + \frac{b}{\|W\|}$$

Functional Margin vs Geometric



$$if\ (\|W\| = 1)\ \text{then}\ \gamma_{Fun} = \gamma_{Geo}$$

$$else\ \gamma_{geo} = \frac{\gamma_{fun}}{\|W\|}$$

Maximum Margin Classifier: take 2

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**
- **Previously:**

$$\underset{\gamma, W, b}{\text{Maximize}}\ \gamma_{Fun}$$

$$\text{subject to}\quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq \gamma_{Fun}\ \ \forall\ i = 1,\ldots,L.$$

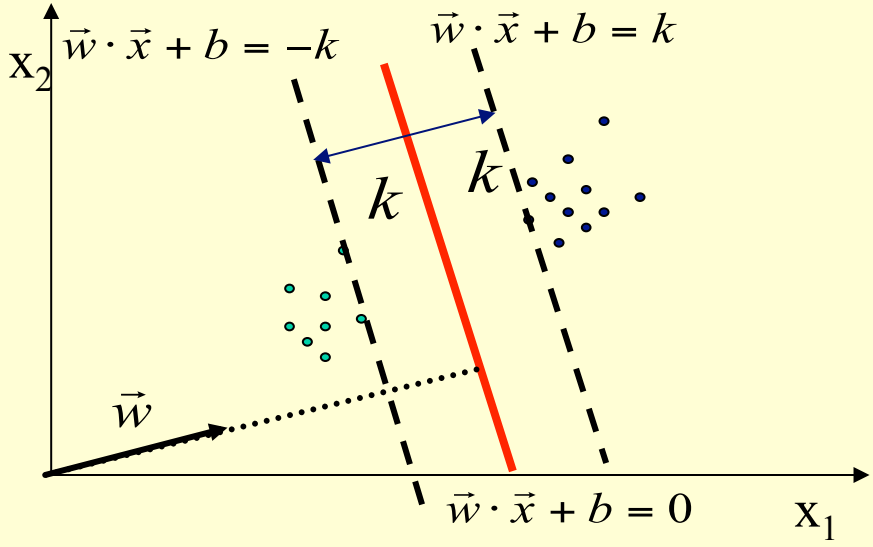
Versus

Better; constrain γ
But the objective
is non-concave

$$\underset{\gamma, W, b}{\text{Maximize}}\ \frac{\gamma_{fun}}{\|W\|}$$

$$\text{subject to}\quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq \gamma_{fun}\ \forall\ i = 1,\ldots,L$$

Setting Up the Optimization Problem



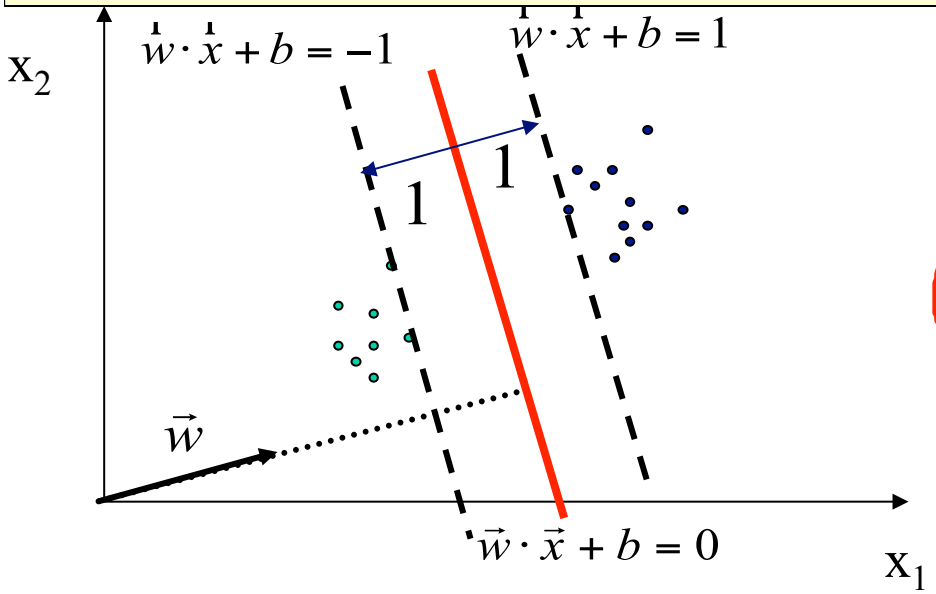
The width of the margin is: $\gamma = \frac{2k}{\|\mathbf{w}\|}$

$$\text{Max } \frac{2k}{\|\mathbf{w}\|^2} \text{ subject to constraints}$$

$$\mathbf{w}\cdot\mathbf{x}_i + b \geq +K \quad \forall\ y_i = +1,$$

$$\mathbf{w}\cdot\mathbf{x}_i + b \leq -K \quad \forall\ y_i = -1,$$

$$\text{or equivalently, } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 \geq 0 \ \ \forall\ \ i = 1,\ldots,L.$$



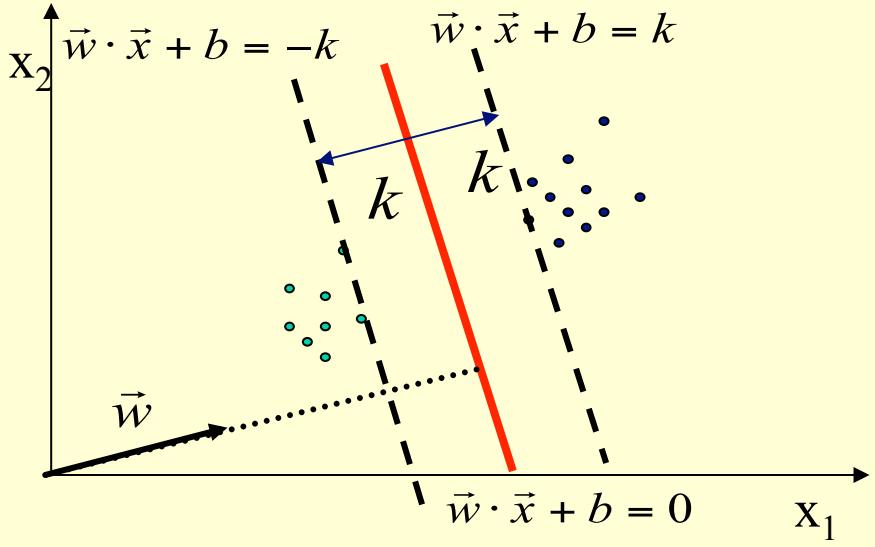
There is a scale and unit for data so that *k=1*. Then problem becomes:

$$\text{Max } \frac{2}{\|\mathbf{w}\|^2}$$

subject to constraints

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 \geq 0 \ \ \forall\ \ i = 1,\ldots,L.$$

Setting Up the Optimization Problem



The width of the margin is: $\gamma = \frac{2k}{\|\mathbf{w}\|}$

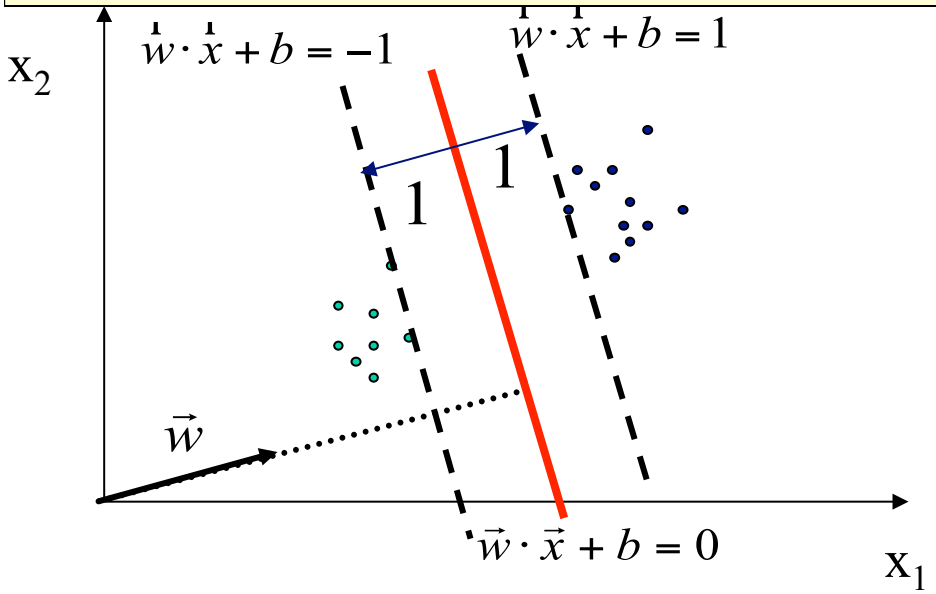
$\text{Max } \frac{2k}{\|\mathbf{w}\|^2}$ subject to constraints

k is Undbounded

$\mathbf{w}\cdot\mathbf{x}_i + b \geq +K \quad \forall\ y_i = +1,$

$\mathbf{w}\cdot\mathbf{x}_i + b \leq -K \quad \forall\ y_i = -1,$

or equivalently, $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 \geq 0 \ \ \forall\ i = 1,\ldots,L.$



There is a scale and unit for data so that *k=1*. Then problem becomes:

$\text{Maximize } \frac{2}{\|\mathbf{w}\|^2} \Leftrightarrow \text{Minimize } \frac{\|\mathbf{W}\|^2}{2}$

subject to constraints

$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 \geq 0 \ \ \forall\ i = 1,\ldots,L.$

Maximum Margin Classifier: take 3

- **The classifier that produces the maximum margin (over the training data)**
- **I.e., the hyperplane that is furthest from the data**

Better!
But the objective is non-convex

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{\gamma_{fun}}{\|W\|}$$

$$\text{subject to} \quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq \gamma_{fun} \;\forall\; i = 1,\ldots,L$$

Versus

Hard SVMs will give an error
For non-separable training dat

Best!
Convex objective

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{1}{\|W\|^2}$$

$$\text{subject to} \quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq 1 \;\forall\; i = 1,\ldots,L$$

$$\gamma_{Fun} = 1$$

$$\text{Scaling constraint: } \gamma_{Fun} \text{ of (W, b) WRT } S \text{ is set to 1}$$

Maximum Margin Classifier

- **Optimal maximum margin (over the training data)**
- **Can be solved using quadratic programming or gradient descent?**

$$\underset{\gamma,W,b}{\text{Maximize}} \frac{1}{\|W\|}$$
$$\text{subject to} \quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq 1 \;\; \forall \;\; i = 1,\ldots,L$$

$$\underset{\gamma,W,b}{\text{Minimize}} \frac{1}{2}\|W\|^2$$
$$\text{subject to} \quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) \geq 1 \;\; \forall \;\; i = 1,\ldots,L$$

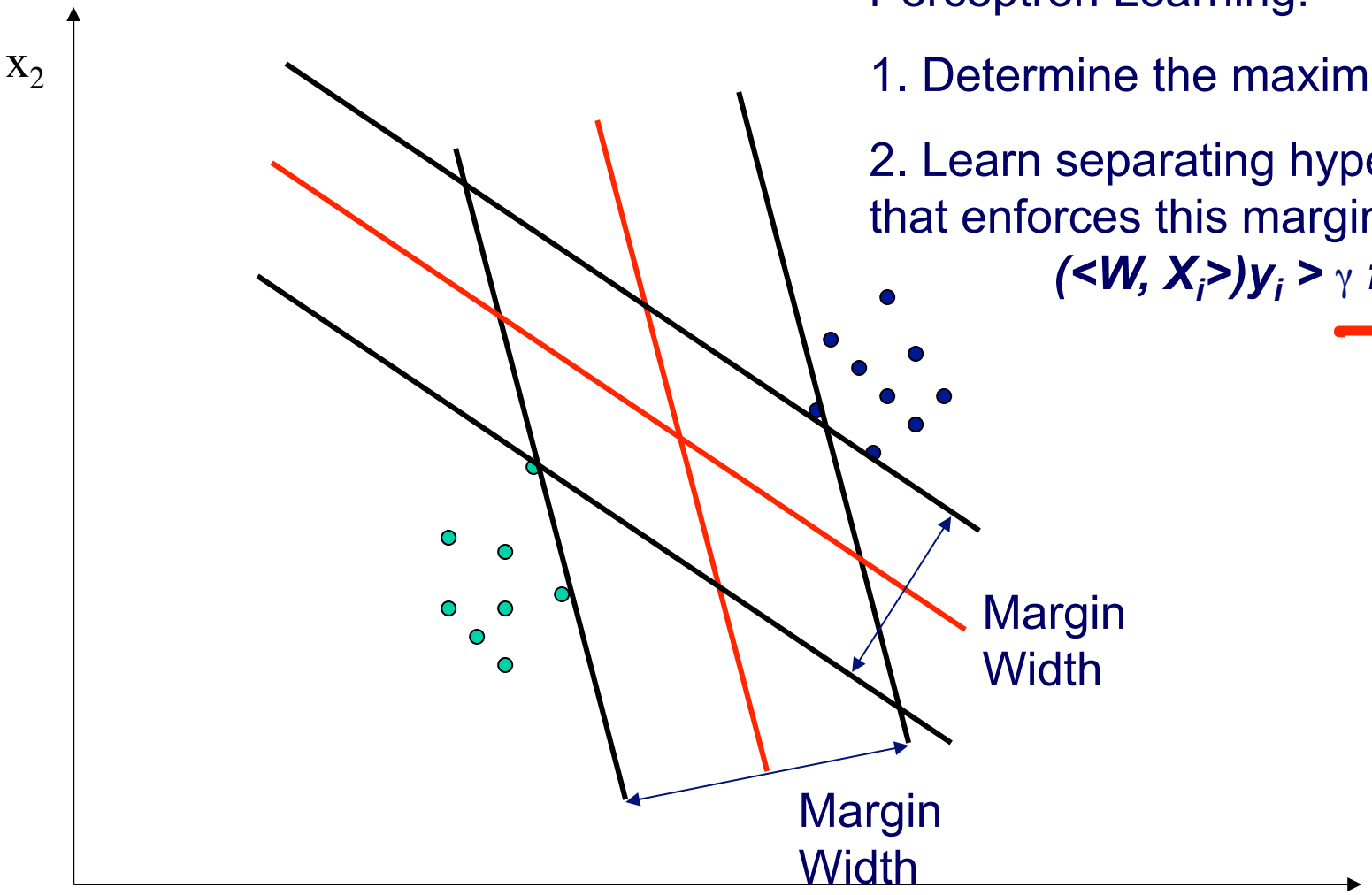
Maximizing the Margin

Chicken-Egg Conundrum for Perceptron Learning:

1. Determine the maximum margin γ

2. Learn separating hyperplane, W, that enforces this margin

$$(\langle W, X_i \rangle) y_i > \gamma \text{ for } I = 1, \ldots, L$$



Learning the Maximal Margin Classier

- **To construct optimal hyperplane**
- **Minimize**
  - Subject to

$$\Phi(W) = \min\left(\frac{\|W\|^2}{2}\right)$$

$$y_i\left((W \cdot X_i) + b\right) \geq 1, i = 1,\ldots,l$$

- **Constrained Optimization problem with Lagrangian**

$$L(w,b,\alpha) = \tfrac{1}{2}\|W\|^2 - \sum_{i=1}^{l} \alpha_i (y_i \cdot ((X_i \cdot W) + b) - 1)$$

Add eqn for each constraint

$$L(w,b,\alpha) = \tfrac{1}{2}\|W\|^2 - \sum_{i=1}^{l} \alpha_i (y_i \cdot ((X_i \cdot W) + b)) + \sum_{i=1}^{l} \alpha_i$$

Refactor α

Learning as Optimization

- **The perceptron algorithm is basically a gradient descent procedure for solving simultaneous linear inequalities (where we specify the required margin)**
- **Represent hyperplane learning as a Lagrange optimization task (learn the maximum margin)**
  - Exploit linear programming/quadratic techniques
    - procedures for maximizing or minimizing linear/quadratic functions subject linear equality or inequality constraints
  - Standard and Slack formulations
  - Lagrange optimization algorithms
    - Simplex Algorithm, ellipsoid algorithm, interior point algorithms
    - Non-LP/QP-based Closed form (SMO, SMOK1, SMOK2)

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term ($R(w)$ expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

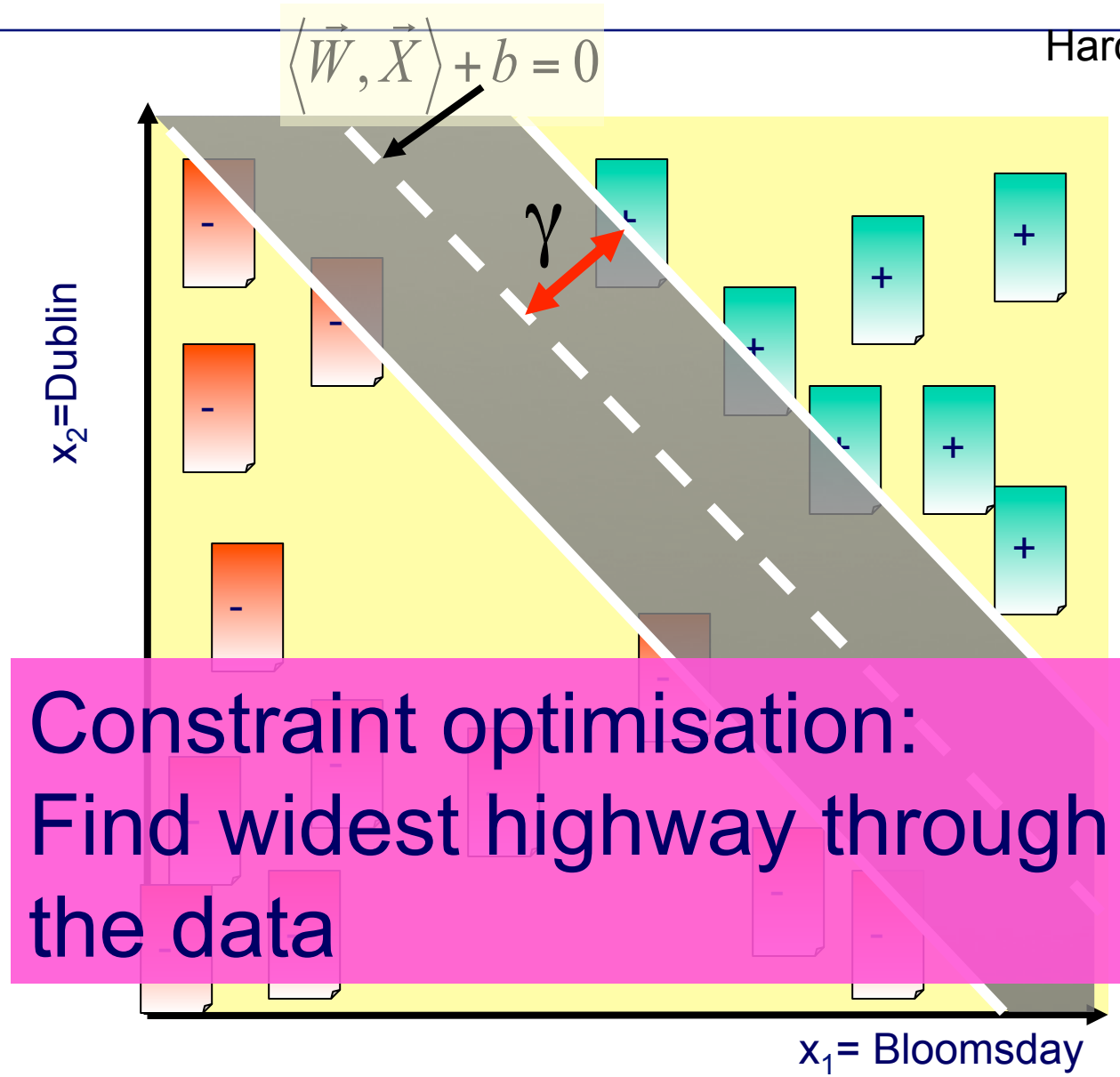
$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

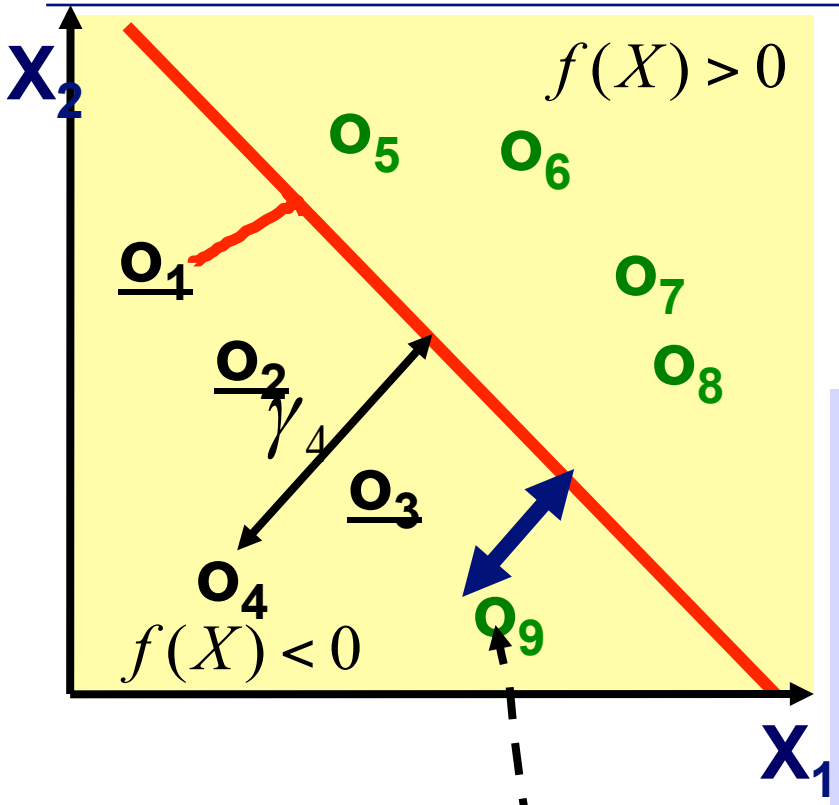
Movie Classifcation

Hard SVMs



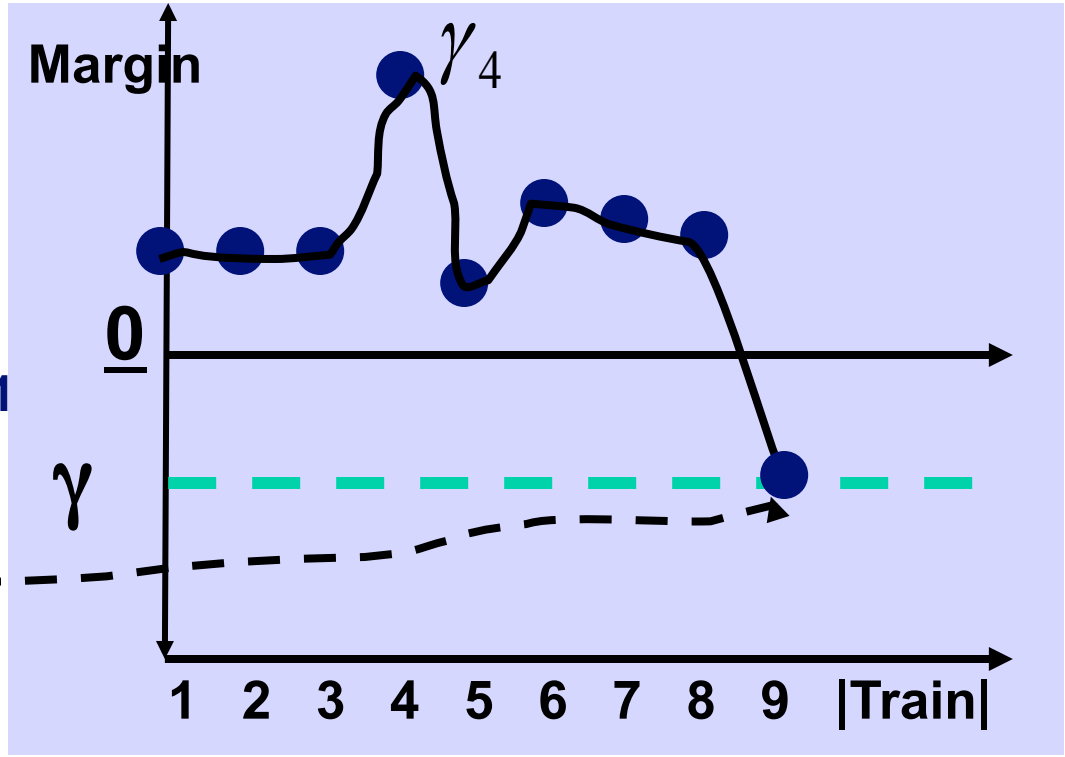
Constraint optimisation:
Find widest highway through the data

Margin and Margin Distribution



Take any linear separator (e.g., a logistic regression or perceptron model)

$$\gamma_{W,b} = \min(y_i(\mathbf{w}\cdot\mathbf{x}_i + b)) \ \forall \ i = 1,\ldots,L.$$



Negative Margin for example o9

Movie Filter

$H_1: \langle \vec{W}, \vec{X} \rangle + b = 1$ $\quad \langle \vec{W}, \vec{X} \rangle + b = 0$ $\quad H_2: \langle \vec{W}, \vec{X} \rangle + b = -1$

Learning can be viewed as optimisation

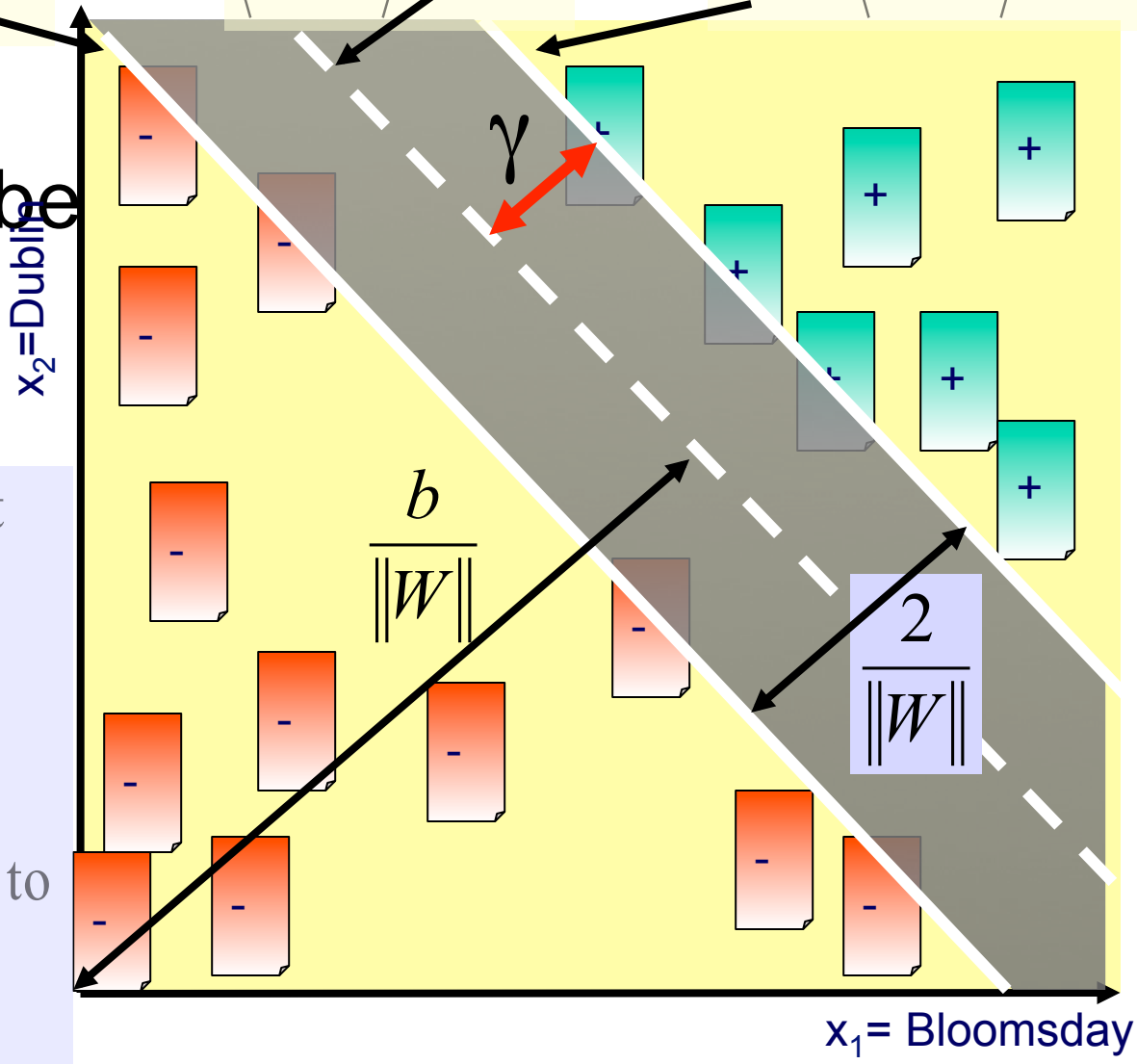
Find H_1 and H_2 such that

$Dist(H_1, H_2)$ is MAX

Or

Minimize $\frac{\|\vec{W}\|^2}{2}$ subject to

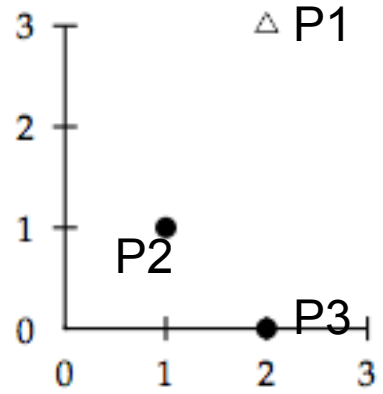
$y_i(< \vec{W}, \vec{X}_i > + b) \geq 1$



Lecture 11 Quiz

- **SVM (geometric) Margin calculation by hand**

Quiz 11.7.1 SVMs

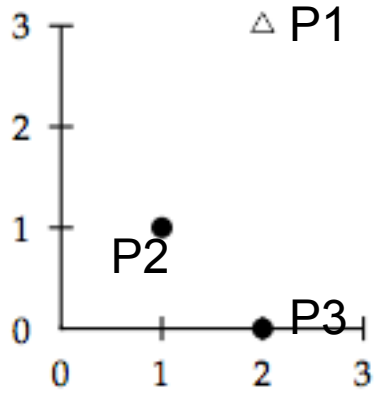


A tiny 3 data point training set for an SVM.

Find the geometric margin of the SVM for this dataset.
Remember the functional margin is 1

- **Consider building an SVM over the (very little) data set shown in the graph here. Working geometrically, for an example like this, the maximum margin weight vector will be parallel to the shortest line connecting points of the two classes, that is, the line between (1, 1) and (2, 3), giving a weight vector of (1, 2). The optimal decision surface is orthogonal to that line and intersects it at the halfway point. Therefore, it passes through (halfway point). So, the SVM decision boundary is y = <A>x_1 + <B>$2x_2$ + <Constant> :**
  - A: halfway point = (1.5, 2); decision boundary is y = $x_1 + 2x_2 - 5.5$
  - B: halfway point = (1, 2); decision boundary is y = $x_1 + 2x_2 - 5.5$
  - C: halfway point = (1.5, 2.5); decision boundary is y = $x_1 + 2.5x_2 - 5.5$
  - D: None of the above

Quiz 11.7.1 SVMs with SOLUTION

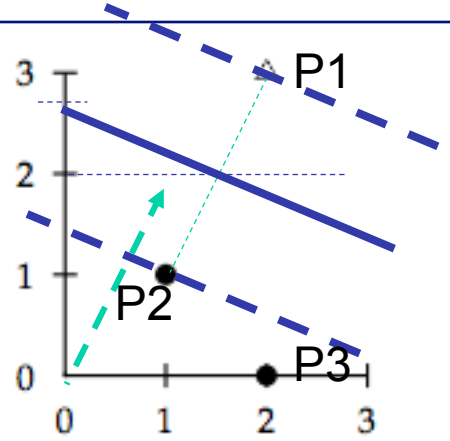


A tiny 3 data point training set for an SVM.

SOLUTION

- **Consider building an SVM over the (very little) data set shown in the graph here. Working geometrically, for an example like this, the maximum margin weight vector will be parallel to the shortest line connecting points of the two classes, that is, the line between (1, 1) and (2, 3), giving a weight vector of (1, 2). The optimal decision surface is orthogonal to that line and intersects it at the halfway point. Therefore, it passes through (halfway point). So, the SVM decision boundary is y = <A>x_1 + <B>$2x_2$ + <Constant> :**
  - A: halfway point = (1.5, 2); decision boundary is y = $x_1 + 2x_2 - 5.5$ CORRECT Answer
  - B: halfway point = (1, 2); decision boundary is y = $x_1 + 2x_2 - 5.5$
  - C: halfway point = (1.5, 2.5); decision boundary is y = $x_1 + 2.5x_2 - 5.5$
  - D: None of the above

Margin calculation by hand



A tiny 3 data point training set for an SVM

Example 15.1: Consider building an SVM over the (very little) data set shown in Figure 15.4. Working geometrically, for an example like this, the maximum margin weight vector will be parallel to the shortest line connecting points of the two classes, that is, the line between $(1,1)$ and $(2,3)$, giving a weight vector of $(1,2)$. The optimal decision surface is orthogonal to that line and intersects it at the halfway point.

Therefore, it passes through $(1.5,2)$. So, the SVM decision boundary is:

$$y = x_1 + 2x_2 - 5.5$$

Working algebraically, with the standard constraint that $\text{sign}(y_i(\vec{w}^T\vec{x}_i + b)) \geq 1$, we seek to minimize $|\vec{w}|$. This happens when this constraint is satisfied with equality by the two support vectors. Further we know that the solution is $\vec{w} = (a, 2a)$ for some a. So we have that:

$$\begin{aligned} a + 2a + b &= -1 \\ 2a + 6a + b &= 1 \end{aligned}$$

Therefore, $a = 2/5$ and $b = -11/5$. So the optimal hyperplane is given by $\vec{w} = (2/5, 4/5)$ and $b = -11/5$.

The margin ρ is $2/|\vec{w}| = 2/\sqrt{4/25 + 16/25} = 2/(2\sqrt{5}/5) = \sqrt{5}$. This answer can be confirmed geometrically by examining Figure 15.4.

We know the form of the weight vector (a, 2a)
Green vector above.

$\rho = 2/|\vec{w}|$ is maximized

For all $(\vec{x}_i, y_i) \in \mathbb{D}, y_i(\vec{w}^T\vec{x}_i + b) \geq 1$

Functional Margin of 1 means a geometric margin of $^{2}/_{|W|}$

Functional Margin of 1 means a geometric margin of 2/|W|

Since we can scale the functional margin as we please, for convenience in solving large SVMs, let us choose to require that the functional margin of all data points is at least 1 and that it is equal to 1 for at least one data vector. That is, for all items in the data:

(15.5) $$y_i(\vec{w}^{\mathrm{T}}\vec{x}_i + b) \geq 1$$

and there exist support vectors for which the inequality is an equality. Since each example's distance from the hyperplane is $r_i = y_i(\vec{w}^{\mathrm{T}}\vec{x}_i + b)/|\vec{w}|$, the geometric margin is $\rho = 2/|\vec{w}|$. Our desire is still to maximize this geometric margin. That is, we want to find $\vec{w}$ and b such that:

- $\rho = 2/|\vec{w}|$ is maximized
- For all $(\vec{x}_i, y_i) \in \mathbb{D}$, $y_i(\vec{w}^{\mathrm{T}}\vec{x}_i + b) \geq 1$

Maximizing $2/|\vec{w}|$ is the same as minimizing $|\vec{w}|/2$. This gives the final standard formulation of an SVM as a minimization problem:

(15.6) Find $\vec{w}$ and b such that:

- $\frac{1}{2}\vec{w}^{\mathrm{T}}\vec{w}$ is minimized, and
- for all $\{(\vec{x}_i, y_i)\}$, $y_i(\vec{w}^{\mathrm{T}}\vec{x}_i + b) \geq 1$

SVMs require a Functional Margin of 1 which means a geometric margin of 1/|W|

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

ML Objectives

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w)$$

Objective Function

Almost all machine learning objectives are optimized using this update

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(_{J(w)}\, x_i, y_i)$$

Update weight vector with gradient of the objective function

w is a vector of dimension d

we're trying to find the best w via optimization

Gradient Descent in ML Algorithms

- **OLS**
- **Logistic Regression**
- **Bayesian logistic regression**
- **Probit regression**
- **Perceptron??**
- **SVMs and its many learning algorithms**
  - Linear SVMs
- **Neural Networks**
- **Ensemble models (e.g., gradient boosted decision trees, ADABoost)**

OLS using Gradient Descent

$$J_q(W, X_1^m) = Minimize \frac{1}{2m} \sum_{i=1}^{m} (W^T X_i - y_i)^2$$

OLS Objective Function With decision variables W

$$g(W; X_i, Y_i) = (y^i - W_i X^i) X^i$$

- Intialialize W = vector or zeros
- Repeat until Convergence

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^{n} \left(X_j W^t - y_i \right) X_j \right)$$

OLS Batch Update Rule

- End Repeat

True gradient is approximated by the gradient of the cost function only evaluated at all examples; adjust parameters proportional to this approx. gradient.

Intuitively, drag weight vector closer to the incorrectly predicted examples

Distributed Gradient Descent: E.g., Linear Regression

Master/Slave Process

- **Initialize model parameters, assume a weight vector W=(0, 0, …, 0); Gradient = (0, 0, …, 0)**
- **While not converged**
  - **MASTER: Broadcast model (i.e., weight vector) to the worker nodes**
  - **MASTER launches map-reduce jobs**
    - Mappers (MANY mappers) to compute partial gradients over the respective training data subsets (chunks)
      - Init g=(0, 0,…, 0)
      - For each training example:
        - » Compute partial gradient for each training example
        - » Combine in memory; g = $\Sigma_i(g(W; X_i, Y_i))$
      - Finally Yield the partial gradient g
    - Reducer (single Reducer)
      - Initialize full gradient: G=(0,0,…,0)
      - For each partial gradient g
        - » Aggregate partial gradients: G = $\Sigma_m g_m$
      - Yield full gradient *G*

    Linear regression
    Goal: learn a weight vector W
    Gradient is defined here:

    $$g(W; X_i, Y_i) = (y^i - W_i X^i) X^i$$

  - **MASTER W = W – αG //update the weight vector**
- **End-While**

Maximum vs Minimum

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

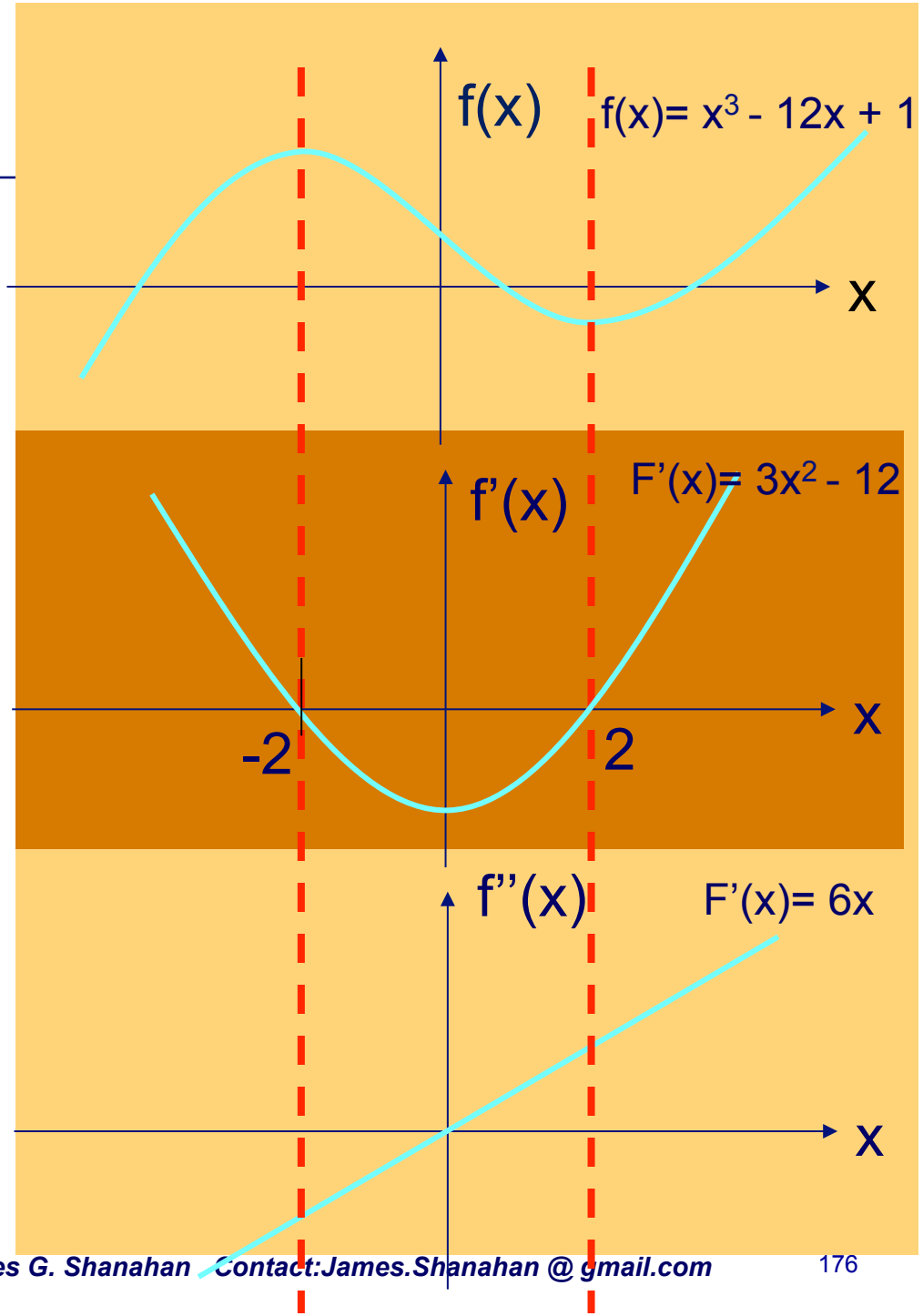
f'(x=x*) =0 at maximum and minimum
These zero points are the roots!

Second derivative $f''(x) = 6x$

Gives the "rate of change of gradient"

SOC

If f''(x=x*) < 0 then maximum
If f''(x=x*) > 0 then minimum
If f''(x=x*) == 0 then ???



Maximum vs Minim

$f(x)= x^3 - 12$

Fi

f

FOC

f'(x=x*) =0 a

These zero p

Second deriv

Gives the "rate of c

SOC

If f''(x=x*) < 0 the

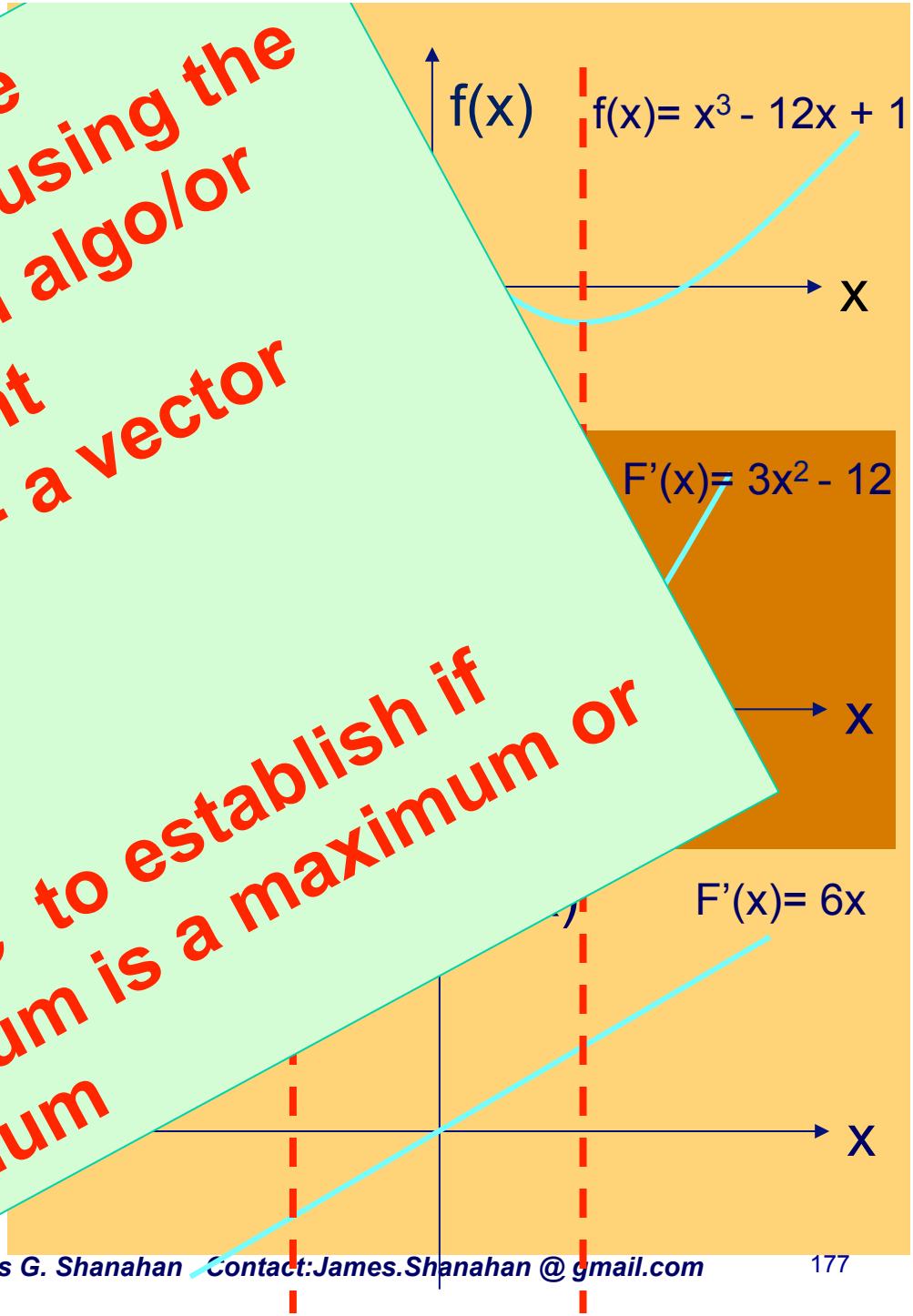
If f''(x=x*) > 0 then

If f''(x=x*) == 0 then

Find the roots of the Gradient function using the Newton-Rhapson algo/or gradient descent

- **FOC ∇ f(x)= a vector zeros**

Use SOC to establish if extremum is a maximum or minimum



Gradient Descent for LR Price~Size Iteration 0

Eqn Line y= aX + b

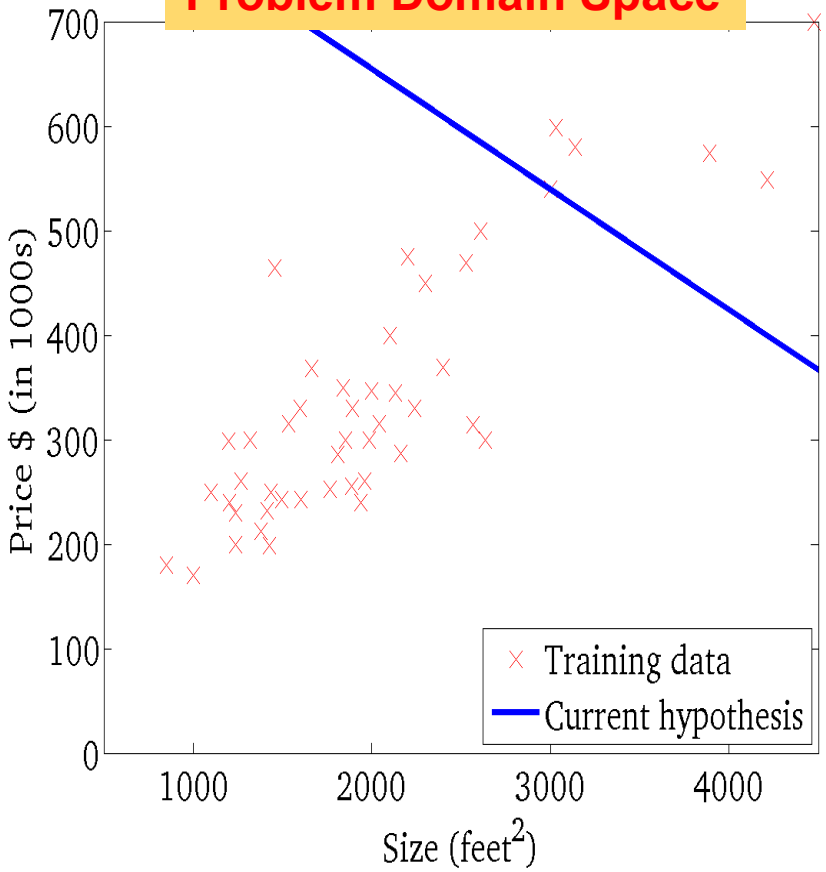
$Y = w_1 X + W_0$

$J(\theta_0, \theta_1)$ $J_q(W, X_1^L) = \frac{1}{m}\sum_{i=1}^{m}(W^T X_i - y_i)^2$

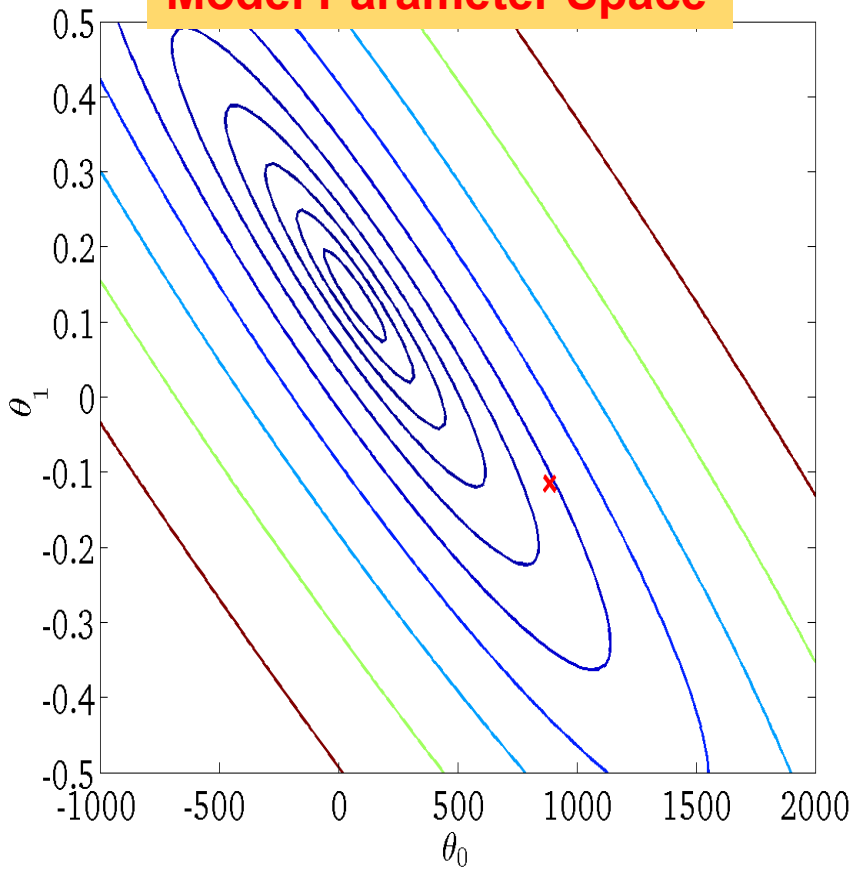
(for fixed θ_0, θ_1 this is a function of x)

(function of the parameters θ_0, θ_1)

Problem Domain Space



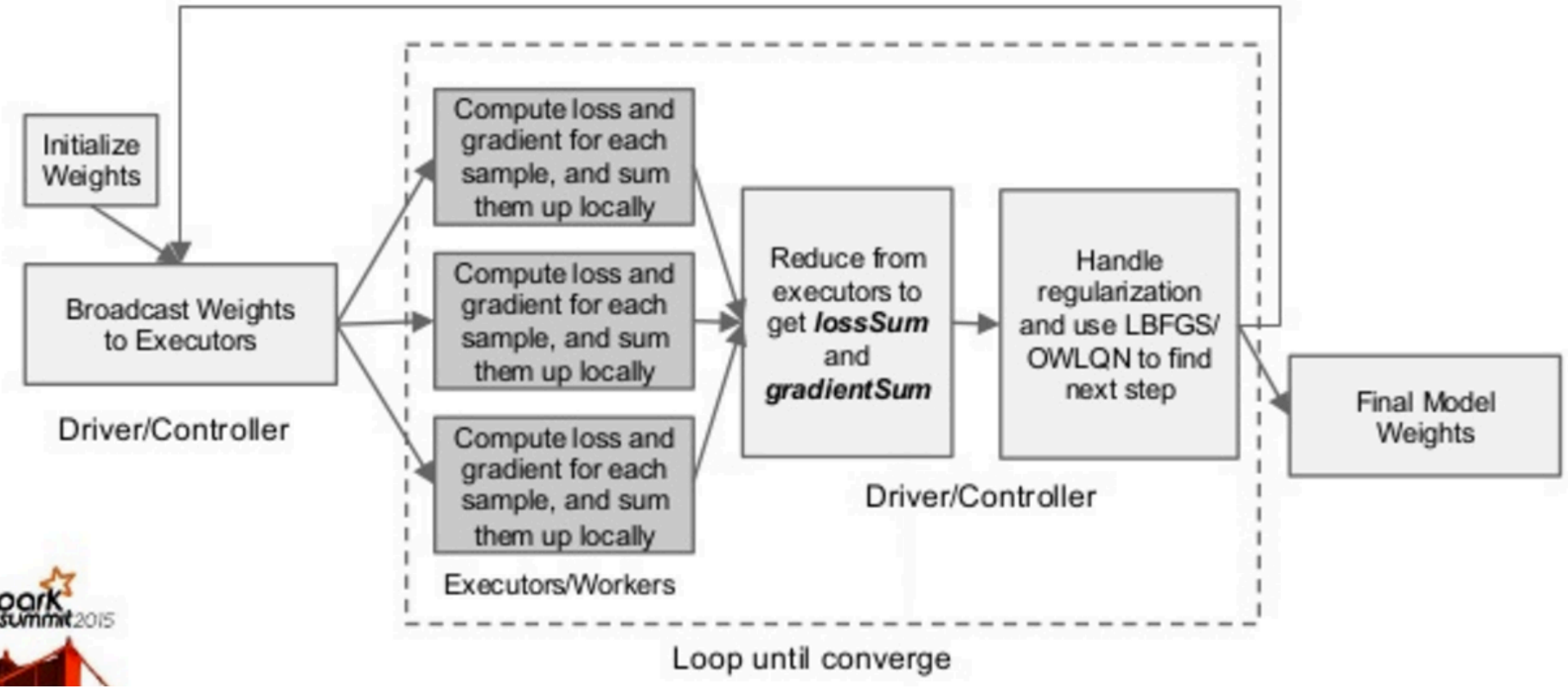
Model Parameter Space



Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

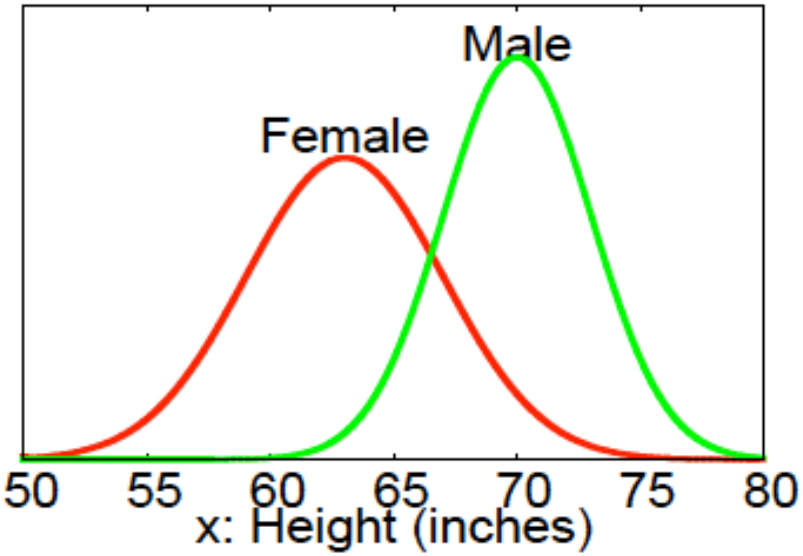
Apache Spark Logistic Regression

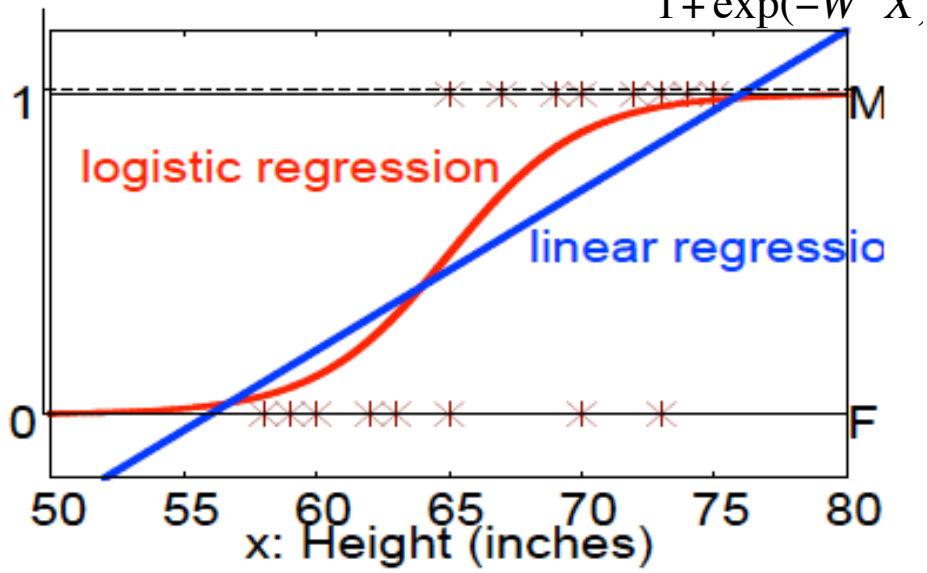


Logistic Regression in One Slide

Example: Predict the *gender* (y=M/F) of a person given their *height* (x=a number)

$$\dot{p} = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1+\exp(-W^T X)}$$





In logReg
Target =ln(p/(1-p))

$$p(y = M \mid x) = \beta_0 + \beta x \quad \text{(linear regression)}$$

$$\ln \frac{p(y = M \mid x)}{p(y = F \mid x)} = \beta_0 + \beta x \quad \text{(logistic regression)}$$

Maximum Likelihood Estimator for Logistic Regression

$$P(y=1|X) = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1+\exp(-W^T X)}$$

Cost Function

- Model output $\hat{y}_k = P(y_k = 1 | \varphi_k)$
- Likelihood contribution $l_{\theta,k} = \hat{y}_k^{\,y_k} (1-\hat{y}_k)^{1-y_k}$
- Likelihood function $L_\theta = \prod_{k=1}^{N} l_{\theta,k}$
- Log-likelihood function $\ln L_\theta = \sum_{k=1}^{N} (y_k \ln \hat{y}_k + (1-y_k) \ln(1-\hat{y}_k))$

Maximum Likelihood Criterion

$$\max_\theta \ln L_\theta \Leftrightarrow \min_\theta -2 \ln L_\theta$$

Logistic Regression assumes a parametric form for P(Y|X)

$$P(y=1 \mid X) = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1+\exp(-W^T X)}$$

Logistic Regression assumes a parametric form for the distribution $P(Y|X)$, then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is boolean is:

$$P(Y=1|X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \quad (16)$$

and

$$P(Y=0|X) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \quad (17)$$

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

Loss term in our loss function

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

Estimating Parameters using Gradient Descent

- **Unfortunately, there is no closed form solution to maximizing l(W) with respect to W.**
  - Therefore, one common approach is to use gradient ascent, in which we work with the gradient, which is the vector of partial derivatives.
  - The ith component of the vector gradient has the form

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Beginning with initial weights of zero, we repeatedly update the weights in the direction of the gradient, changing the *ith* weight according to *this formula,* where η is a small constant (e.g., 0*:01) which determines the step size.*

***Effectively we are p*ulling weight vector closer to the examples where we make mistakes**

Logistic Regression: Gradient Descent

y ={-1, 1}

- **Objective Function (Logloss):**

$$\arg\min_W L(w) = \operatorname{argmin}_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1 + e^{-y_i \cdot w^T \cdot x_i})$$

- **Gradient**

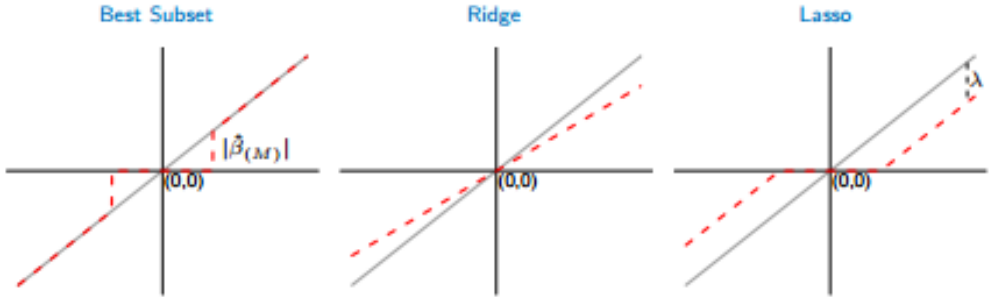
$$\nabla \mathbf{w} = \frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1} (\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1) \cdot y_i \cdot x_i$$

- **Update w till it converges**

Gradient Descent $\mathbf{w} = \mathbf{w} - \eta \cdot \nabla \mathbf{w}$

LogReg Regularization: Constrained optimization

- **Add regularization to prevent over-fitting**
- **Lasso**
  - L1 Norm regularization: **|w|**
- **Ridge**
  - L2 Norm regularization: **w^2**
- **shrinkage of w**



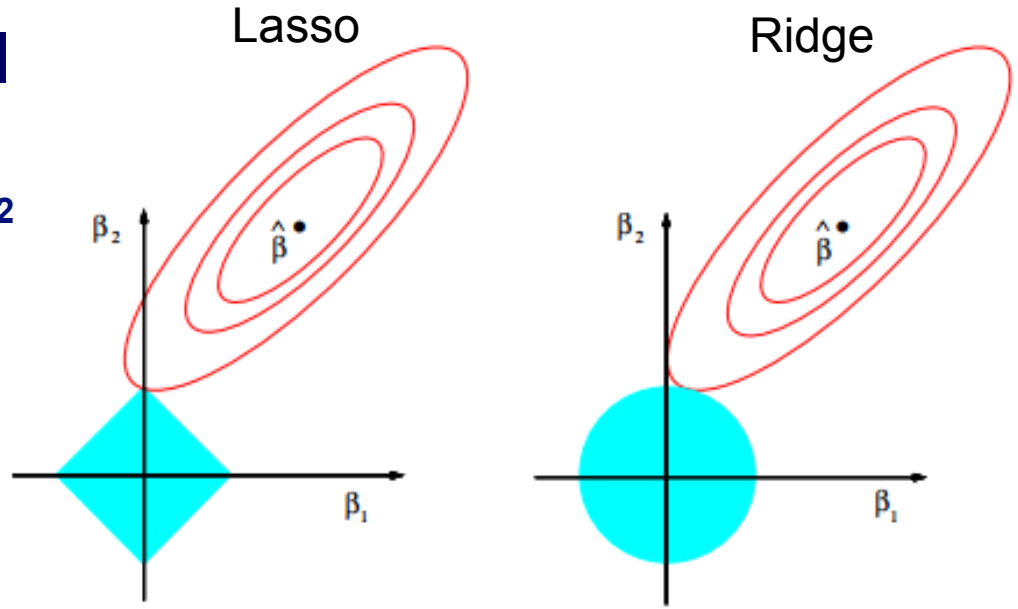


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

From The Elements of Statistical Learning

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/v1fkophu23g3v7e/LogisticRegression-Notebook-Challenge.ipynb

Logistic Regression

Data Generation

```
import numpy as np
import csv
def data_generate(fileName, w=[0,0,0], size = 100):
    size = 100
    np.random.seed(0)
    x1 = np.random.uniform(-4, 4, size)
    x2 = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 3, size)
    v = (x1 * w[0] + x2 * w[1] + w[2] + noise)
    y = (v>0) * 2-1
    data = zip(y, x1, x2)
    with open(fileName,'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

```
w = np.array([8, -3, -1])
data_generate('data.csv', w, 100)
```

```
True
```

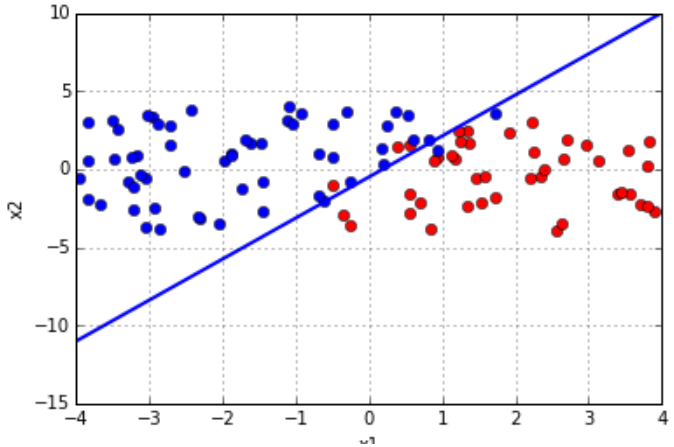
Data format: y, x1,x2

```
1,-6.886661137329684879e-01
-1,1.756760849652599932e+00
-1,7.656676365450297839e-01
1,3.740612089503772886e+00
```

Data Visualization

```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    cols = {'1': 'r', '-1': 'b'}
    with open(file,'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[2]), cols[row[0]]+'o')
    plt.xlabel("x1")
    plt.ylabel("x2")
    x1 = [-4,4]
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, linewidth=2.0)
    plt.grid()
    plt.show()
```

```
dataPlot('data.csv',w)
```



Logistic Regression

Objective Function

$$\arg\min_W L(w) = \operatorname{argmin}_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1+e^{-y_i \cdot w^T \cdot x_i})$$

y ={-1, 1}

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{1+e^{-y_i \cdot w^T \cdot x_i}} - 1\right) \cdot y_i \cdot x_i$$

Gradient descent (no regularization)

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/v1fkophu23g3v7e/LogisticRegression-Notebook-Challenge.ipynb

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0)  #bias term
    return Point(x, float(d[0]))

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) * p.y * np.array(p.x))\
                   .reduce(lambda a, b: a + b)
        w = w - learningRate * gradient / n
    #w = w / np.linalg.norm(w) #normalization
    return w
```

Sum up gradients

Get gradients for each instance

```
data = sc.textFile('data.csv').map(readPoint).cache()
logisticRegressionGD(data)
```

```
array([ 1.35420091, -0.46407845, -1.01559616])
```

Logistic Regression

Objective Function

$$\arg\min_W L(w) = \arg\min_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1+e^{-y_i \cdot w^T \cdot x_i})$$

y ={-1, 1}

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{1+e^{-y_i \cdot w^T \cdot x_i}} - 1\right)\cdot y_i \cdot x_i$$

Gradient descent (no regularization)

Error

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0)  #bias term
    return Point(x, float(d[0]))

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) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b)
    return w
```

Get gradients for each instance

Sum gradients for each instance

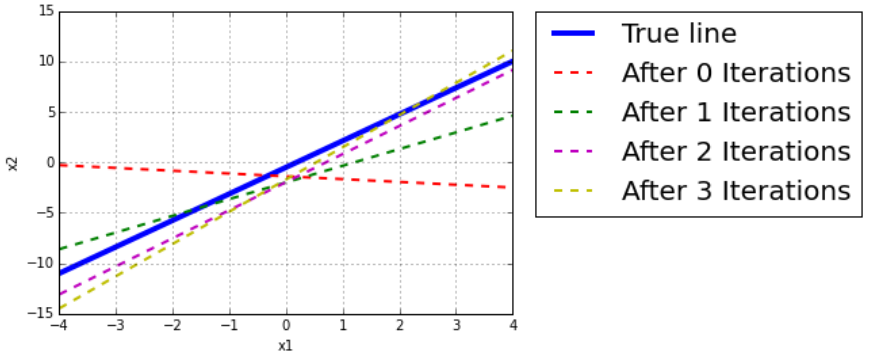
```
data = sc.textFile('data.csv').map(readPoint).cache()
logisticRegressionGD(data)
```

```
array([ 1.35420091, -0.46407845, -1.01559616])
```

Logistic Regression

Plot w in iterations

```
def iterationsPlot(fileName, truew):
    x1 = [-4, 4]

    w = truew
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'b', label="True line", linewidth=4.0)

    np.random.seed(800)
    w = np.random.normal(0,1,3)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'r--', label="After 0 Iterations", linewidth=2.0)

    data = sc.textFile(fileName).map(readPoint).cache()
    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'g--', label="After 1 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'm--', label="After 2 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'y--', label="After 3 Iterations", linewidth=2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxespad=0.)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid()
    plt.show()
```




```
iterationsPlot('data.csv',w)
```

Logistic Regression vs Ridge and Lasso

Objective Function $\arg\min_W L(w) = \arg\min_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1+e^{-y_i \cdot w^T \cdot x_i})$ y ={-1, 1}

Gradient Descent $\frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{1+e^{-y_i \cdot w^T \cdot x_i}} - 1\right) \cdot y_i \cdot x_i$

- **Ridge (L2 regularization)**

Objective Function $\arg\min_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1+e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot w^T \cdot w$ y ={-1, 1}

Gradient Descent $\frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{1+e^{-y_i \cdot w^T \cdot x_i}} - 1\right) \cdot y_i \cdot x_i + \lambda w$

y ={-1, 1}

- **Lasso(L1 regularization)**

Objective Function $\arg\min_W \frac{1}{n}\sum_{i=0}^{n-1} Log(1+e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot |w|$

Vector of size of W of -1 or +

Gradient Descent $\frac{\partial L}{\partial w} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{1+e^{-y_i \cdot w^T \cdot x_i}} - 1\right) \cdot y_i \cdot x_i + \lambda \cdot \mathrm{sgn}(w)$

namedtuple

namedTuple

The standard **tuple** uses numerical indexes to access its members.

```
bob = ('Bob', 30, 'male')
print 'Representation:', bob

jane = ('Jane', 29, 'female')
print '\nField by index:', jane[0]

print '\nFields by index:'
for p in [ bob, jane ]:
    print '%s is a %d year old %s' % p
```

This makes **tuples** convenient containers for simple uses.

```
$ python collections_tuple.py

Representation: ('Bob', 30, 'male')

Field by index: Jane

Fields by index:
Bob is a 30 year old male
Jane is a 29 year old female
```

Examples as namedTuple

```
Point = namedtuple('Point', 'x y')  #Point record with fields x and y, where x will be a array of 2 real numbers
xx= Point([1.2,2.3], 1)
print xx[0], xx[1]
print xx.x, xx.y
```

```
[1.2, 2.3] 1
[1.2, 2.3] 1
```

Challenge: Extend OLS to Regularized OLS

- **Challenge: Extend Logistic Regression to Regularized OLS**

- **Code up Lasso and Ridge based Logistic Regression starting from this notebook**

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/v1fkophu23g3v7e/LogisticRegression-Notebook-Challenge.ipynb

Ridge and Lasso Regression

Linear Regression

$$\min_W L(w) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)^2$$

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)\cdot x_i$$

Ridge Regression (L2 regularization)

Objective Function

$$\min_W L(w) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)^2 + \lambda \cdot w^T \cdot w$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)\cdot x_i + \lambda w$$

Lasso Regression (L1 regularization)

Objective Function

$$\min_W L(w) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)^2 + \lambda \cdot |w|$$

Vector of size of W of -1 or +1

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=0}^{n-1}(y_i - w^T \cdot x_i)\cdot x_i + \lambda \cdot \mathrm{sgn}(w)$$

Hints

- **vector of sign()**

```
In [7]: np.sign(wReg)
Out[7]: array([ 1, -1,  1, -1,  0])
```

```
In [7]: np.sign(wReg)
Out[7]: array([ 1, -1,  1, -1,  0])

In [9]: wReg.shape[0]
Out[9]: 5

In [ ]:
```

Logistic regression with Regularization

Gradient descent (regularization)

```
def logisticRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    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 p: (1 / (1 + np.exp(-p.y*np.dot(wBroadcast.value, p.x)))-1) * p.y * np.array(p.x))\
                    .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 regularization
        elif regType == "Lasso":
            wReg = w * 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(w.shape[0])
        gradient = gradient + regParam * wReg  #gradient:  GD of Sqaured Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/3e1bsvpvqbf97u1/LogisticRegression-Notebook.ipynb

Ridge Regression

```
np.random.seed(400)
logisticRegressionGDReg(data, iterations=50, regParam=0.1, regType="Ridge")
```

```
array([ 0.74835721, -0.17134752, -0.39953122])
```

Lasso Regression

Separable Updates

Can be generalized for

» Unconstrained optimization

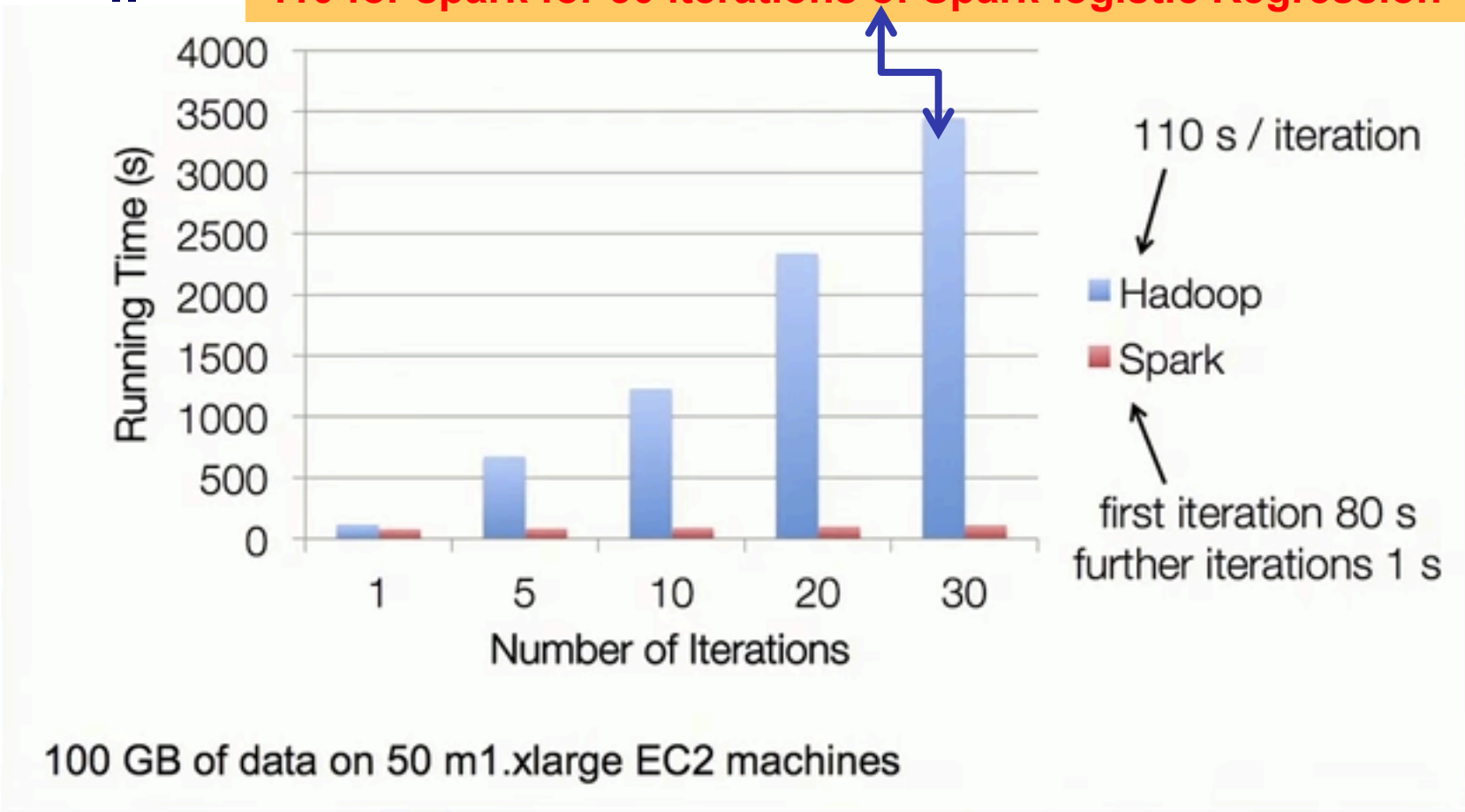
» Smooth or non-smooth

» LBFGS, Conjugate Gradient, Accelerated Gradient methods, …

Logistic Regression: Hadoop vs Spark

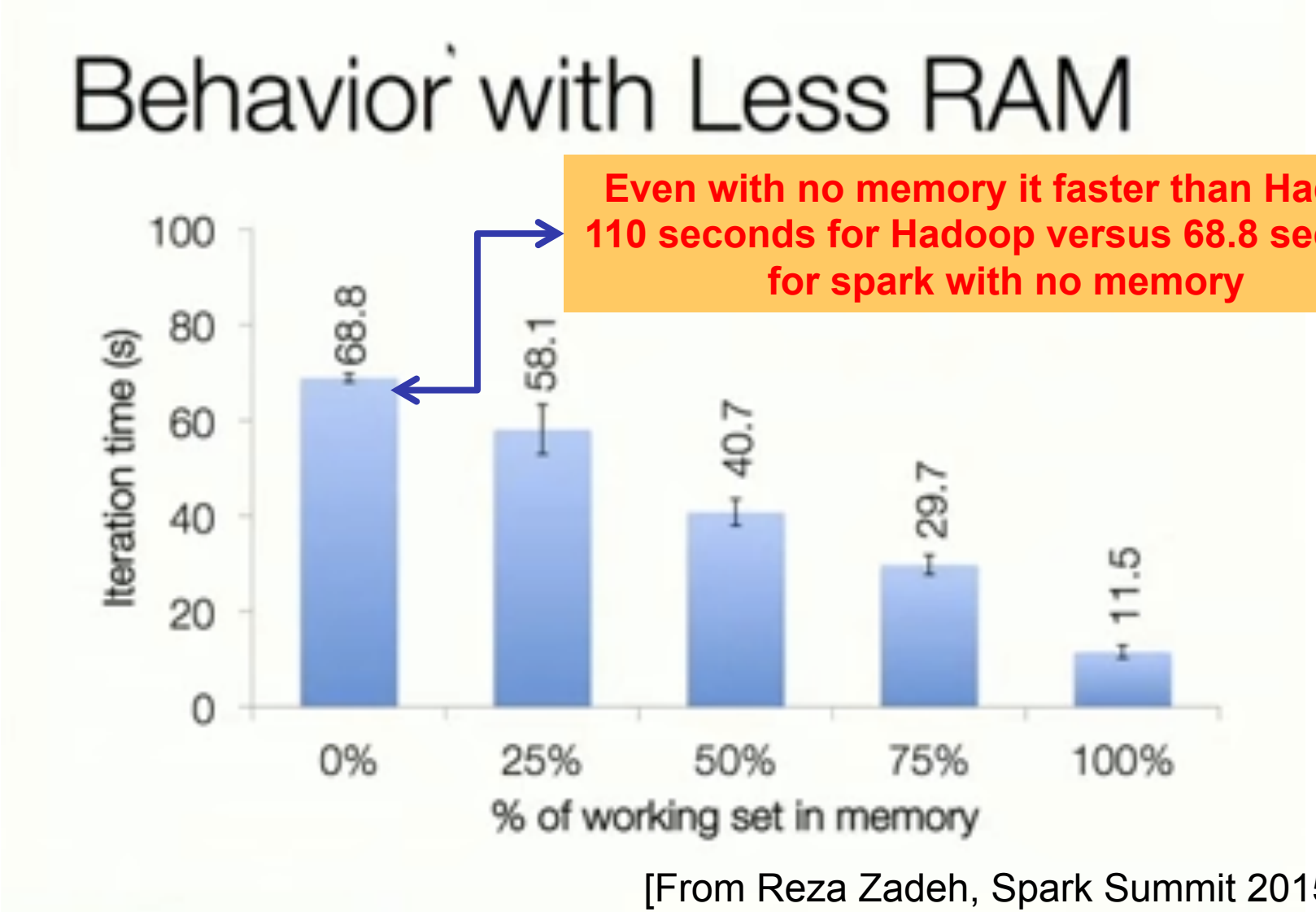
- ..

3,500 for 30 iterations of Hadoop logistic Regression
versus
110 for spark for 30 iterations of Spark logistic Regression



[From Reza Zadeh, Spark Summit 2015]

If the working set does not fit in memory



[From Reza Zadeh, Spark Summit 2015]

Apache Spark Linear Models

- **[SPARK-5253]** Linear Regression with Elastic Net (L1/L2)
  [SPARK-7262] Binary Logistic Regression with Elastic Net
  - Author: DB Tsai, merged in Spark 1.4
  - Internally handle feature scaling to improve convergence and avoid penalizing too much on those features with low variances
  - Solutions exactly match R's glmnet but with scalability
  - For LiR, the intercept is computed using close form like R
  - For LoR, clever initial weights are used for faster convergence
- **[SPARK-5894]** Feature Polynomial Mapping
  - Author: Xusen Yin, merged in Spark 1.4

linear regression with L1, L2, and elastic-net regularization

Regularization

- The loss function becomes

$$l_{total}(\vec{w}, \vec{x}) = l_{model}(\vec{w}, \vec{x}) + l_{reg}(\vec{w})$$

- The loss function of regularization doesn't depend on data.

- Common regularizations are
  - L2 Regularization: $l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} w_i^2$
  - L1 Regularization: $l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} |w_i|$
  - Elastic-Net Regularization: $l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} \left(\frac{\alpha}{2} w_i^2 + (1-\alpha)|w_i|\right)$

Loss functions

http://spark.apache.org/docs/latest/mllib-linear-methods.html#logistic-regression

The following table summarizes the loss functions and their gradients or sub-gradients for the methods MLlib supports:

	loss function $L(\mathbf{w}; \mathbf{x}, y)$	gradient or sub-gradient
hinge loss	$\max\{0, 1 - y\mathbf{w}^T\mathbf{x}\}, \quad y \in \{-1, +1\}$	$\begin{cases} -y \cdot \mathbf{x} & \text{if } y\mathbf{w}^T\mathbf{x} < 1, \\ 0 & \text{otherwise.} \end{cases}$
logistic loss	$\log(1 + \exp(-y\mathbf{w}^T\mathbf{x})), \quad y \in \{-1, +1\}$	$-y\left(1 - \frac{1}{1+\exp(-y\mathbf{w}^T\mathbf{x})}\right) \cdot \mathbf{x}$
squared loss	$\frac{1}{2}(\mathbf{w}^T\mathbf{x} - y)^2, \quad y \in \mathbb{R}$	$(\mathbf{w}^T\mathbf{x} - y) \cdot \mathbf{x}$

Regularizers

The purpose of the regularizer is to encourage simple models and avoid overfitting. We support the following regularizers in MLlib:

	regularizer $R(\mathbf{w})$	gradient or sub-gradient
zero (unregularized)	0	$\mathbf{0}$
L2	$\frac{1}{2}\|\mathbf{w}\|_2^2$	$\mathbf{w}$
L1	$\|\mathbf{w}\|_1$	$\text{sign}(\mathbf{w})$
elastic net	$\alpha\|\mathbf{w}\|_1 + (1-\alpha)\frac{1}{2}\|\mathbf{w}\|_2^2$	$\alpha\text{sign}(\mathbf{w}) + (1-\alpha)\mathbf{w}$

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Metrics are key

- Metrics are key
- Diagnostics

Challenge

- **Decipher the following diagnostics from a learnt logistic regression model:**
  - See next slide

LR Model Summary

- **summary(mod.mroz.glm)**
- Call:

glm(formula = lfp ~ k5 + k618 + age + wc + hc + lwg + inc, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.1062	-1.0900	0.5978	0.9709	2.1893

Residual distribution in logit space
Use summary(predict(mod.mroz.glm, Mroz)) to recover the probabilities

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	3.182140	0.644375	4.938	7.88e-07	***
k5	-1.462913	0.197001	-7.426	1.12e-13	***
k618	-0.064571	0.068001	-0.950	0.342337	
age	-0.062871	0.012783	-4.918	8.73e-07	***
wc	0.807274	0.229980	3.510	0.000448	***
hc	0.111734	0.206040	0.542	0.587618	
lwg	0.604693	0.150818	4.009	6.09e-05	***
inc	-0.034446	0.008208	-4.196	2.71e-05	***

Feature Significance

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 1029.75 on 752 degrees of freedom Null Deviance where phat is #successes/#events

Residual deviance: 905.27 on 745 degrees of freedom Residual Deviance= -2*LogLikelihood

AIC: 921.27

AIC = 905+ 2* 8 #7 variables + bias term

Number of Fisher Scoring iterations: 4

Classification Rule: Logistic Regression

- **To classify any given X we generally want to assign the value y that maximizes P(Y = y |X). Put another way, we assign the label y = 0 if the following condition holds:**

$$y = \begin{cases} 0 & if \quad 1 < \dfrac{P(Y^i = 0 \mid X^i; W)}{P(Y^i = 1 \mid X^i; W)} \\ 1 & otherwise \end{cases} \backslash$$

simplifies to

$$y = 0 \quad \text{if } 1 < \exp(W^T X^i)$$

taking natural log of both sides

$$y = 0 \quad \text{if } 0 < W^T X^i$$

Assign 0 label if $W^T X^i \geq 0$

[http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf]

Logistic Regression is a Linear Classifier in the problem domain space

$$y = \begin{cases} 0 & if \quad 1 < \dfrac{P(Y^i = 0 \mid X^i; W)}{P(Y^i = 1 \mid X^i; W)} \\ 1 & otherwise \end{cases}$$

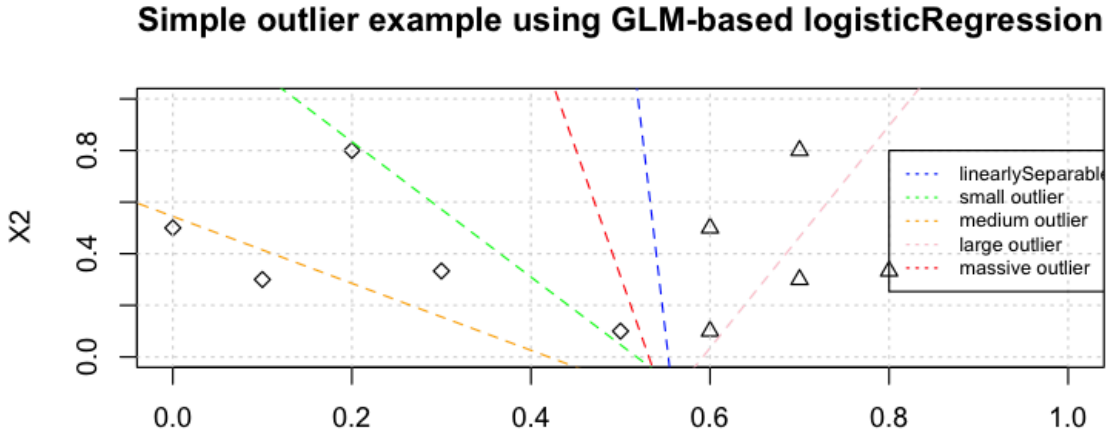
simplifies to

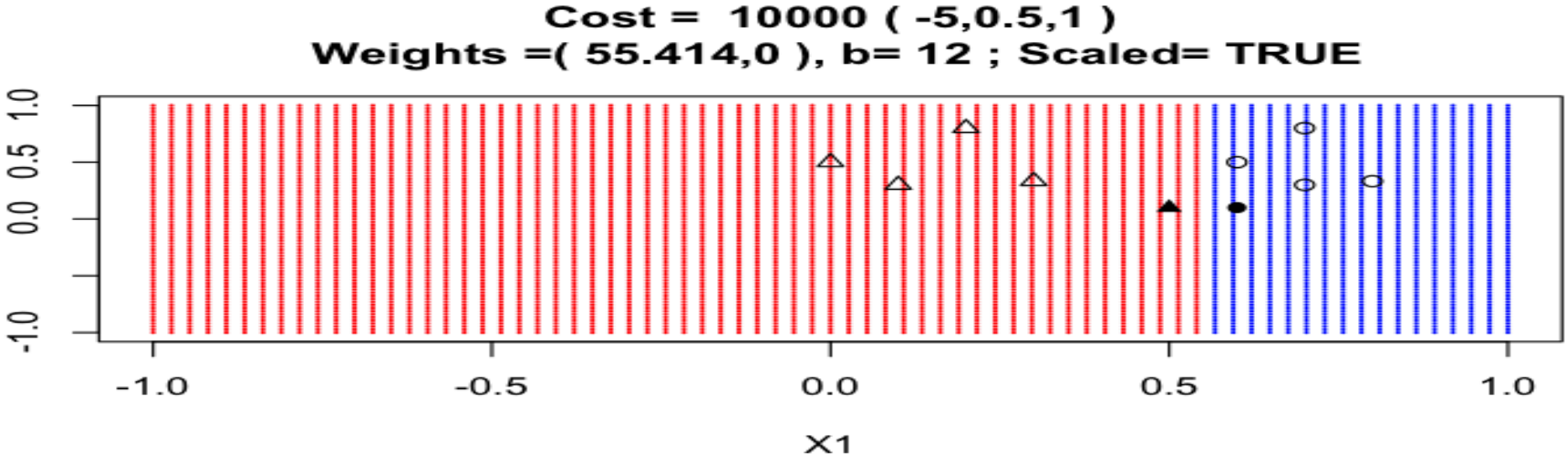
$y = 0$ if $1 < \exp(W^T X^i)$

taking natural log of both sides

$y = 0$ if $0 < W^T X^i$

Assign 0 label if $W^T X^i \geq 0$





Assessing Logistic Regression Models

- **Residual Deviance (-2*l(W)) [lower is better]**
  - The lower the better
- **Akaike information criterion (AIC) [lower is better]**
  - - 2 * logLikelihood + 2 * number of parameters.
  - AIC = −2LL + 2q,
    - with q being the number of parameters (here, q = 3).
  - AIC has some truly devoted adherents, especially among nonstatisticians
- **Bayes information criterion (BIC) [lower is better]**
  - - 2 * log-likelihood + ln(number of observations) * number of parameters
- **Blind Test:**
  - Classify each example based on the 0.5 threshold rule
  - Use Accuracy, precision, recall

LOSS FUNCTION

Topics: loss function for classification

- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
  - we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy

Negative Log Likelihood of Learnt Model W

- **Negative log-likelihood of our model**

$$\hat{\beta} = \text{argmin}_\beta \sum_{i=1}^{n} -\log p(y_i|x_i;\beta) + \mu \sum_{j=0}^{d} \beta_j^2.$$

Minimize $-2l(W) = -2\sum_{i=1}^{m} y^i \log p - (1-y^i)\log(1-p)$

Actual Predicted
-1 * log(0.5)
-1 * -0.7 = 0.7 Error
Bigger error

vs

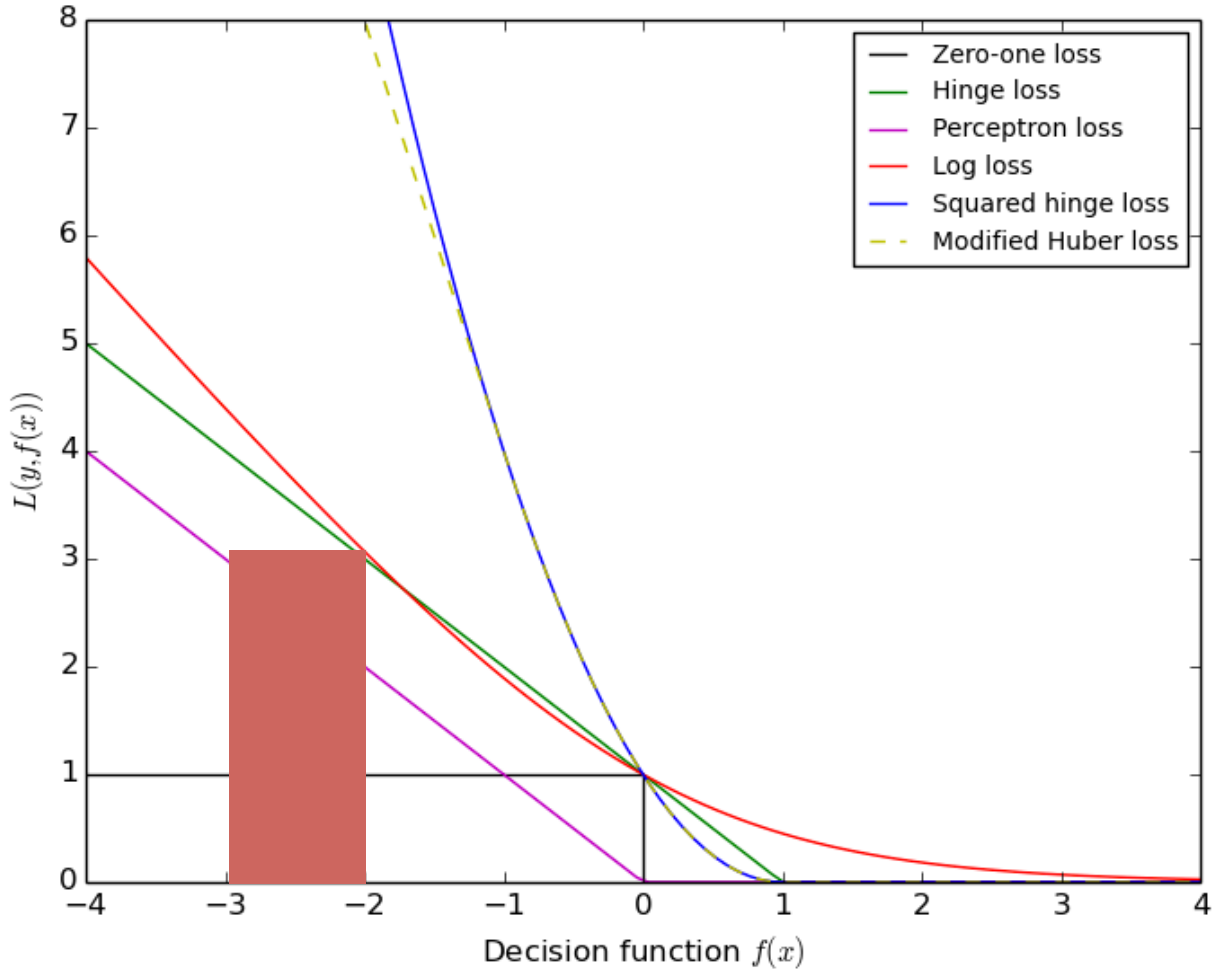
-1 * log(0.9)
-1*-0.1 = 0.1 Error

- **AKA Residual Deviance**
- **R Code for negative log likelihood for MROZ data**

```
library("car") #Mroz
attach(Mroz)
mod.mroz.glm = glm(lfp ~ k5 + k618 + age +wc + hc + lwg + inc,
    family=binomial))
 coefficients(mod.mroz.glm)
 phat=mod.mroz.glm$fitted.values
 y= ifelse(lfp=='yes', 1,0)
minusTwoTimesLogLik = -2 * sum(y*log(phat) + (1-y)*log(1-phat) )
```

```
mod.mroz.glm  =glm(lfp ~ k5 + k618 + age, family=binomial)
phat=mod.mroz.glm$fitted.values
 y= ifelse(lfp=='yes', 1,0)
minusTwoTimesLogLik = -2 * sum(y*log(phat) + (1-y)*log(1-phat
> minusTwoTimesLogLik
[1] 960.7074
```

905.266 is the minusTwoTimesLogLikelihood of the data given the learnt model (lower is better!)


Zero-one loss
Hinge loss
Perceptron loss
Log loss
Squared hinge loss
Modified Huber loss
$L(y, f(x))$
Decision function $f(x)$
8
7
6
5
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4

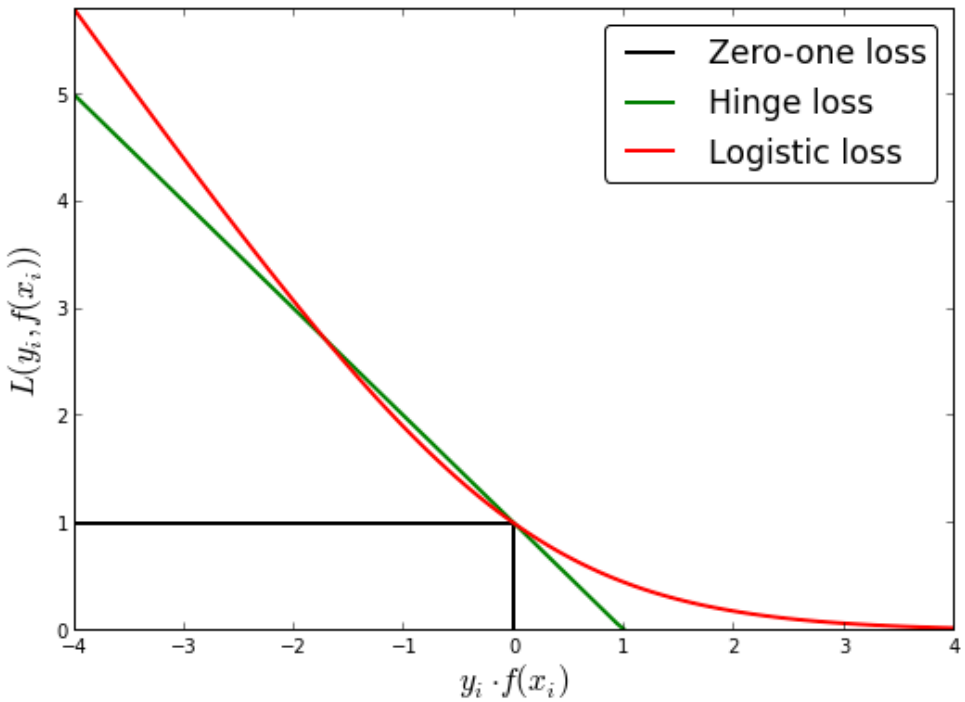
In machine learning it is common to formulate the classification task as a regression loss function minimization problem over a given loss function. Given data input data $(x_1, \ldots, x_n)$ and associated labels $(y_1, \ldots, y_n), y_i \in \{-1, 1\}$, the problem becomes to find a function $f(x)$ that minimizes

$$L(x, y) = \sum_i^n \text{loss}(f(x_i), y_i)$$

where loss is any loss function. These are usually functions that become close to zero when $f(x_i)$ agrees in sign with y_i and have a non-negative value when $f(x_i)$ have opposite signs. Common choices of loss functions are:

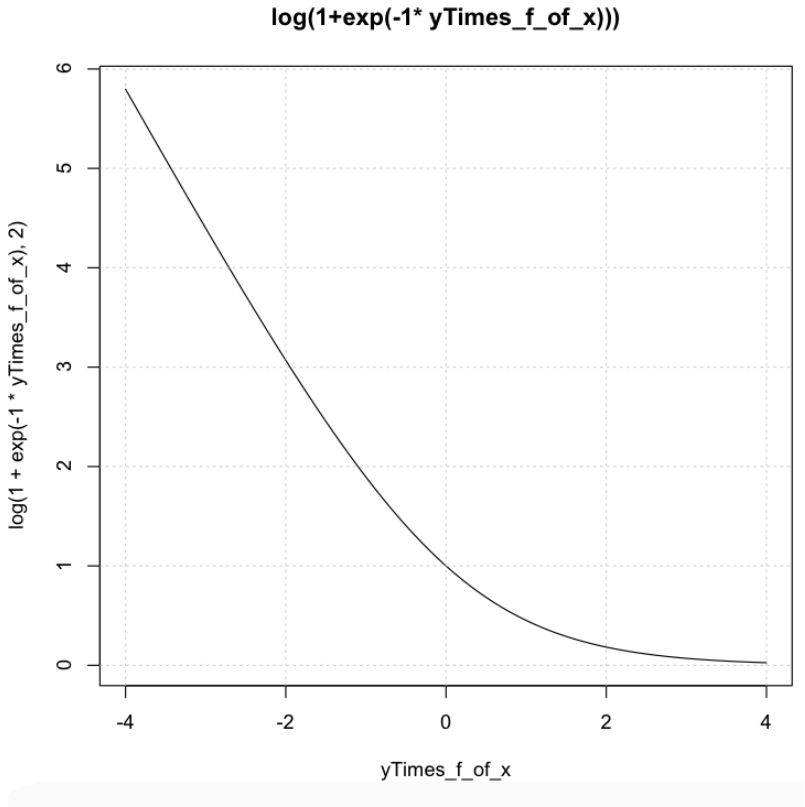
- Zero-one loss, $I(f(x_i) = y_i)$, where I is the indicator function.
- Hinge loss, $\max(0, 1 - f(x_i)y_i)$
- Logistic loss, $\log(1 + \exp f(x_i)y_i)$

Logit space -2sigma, 2sigma



If y =1 P,
If y =0 1-P

Log loss function (base 2)



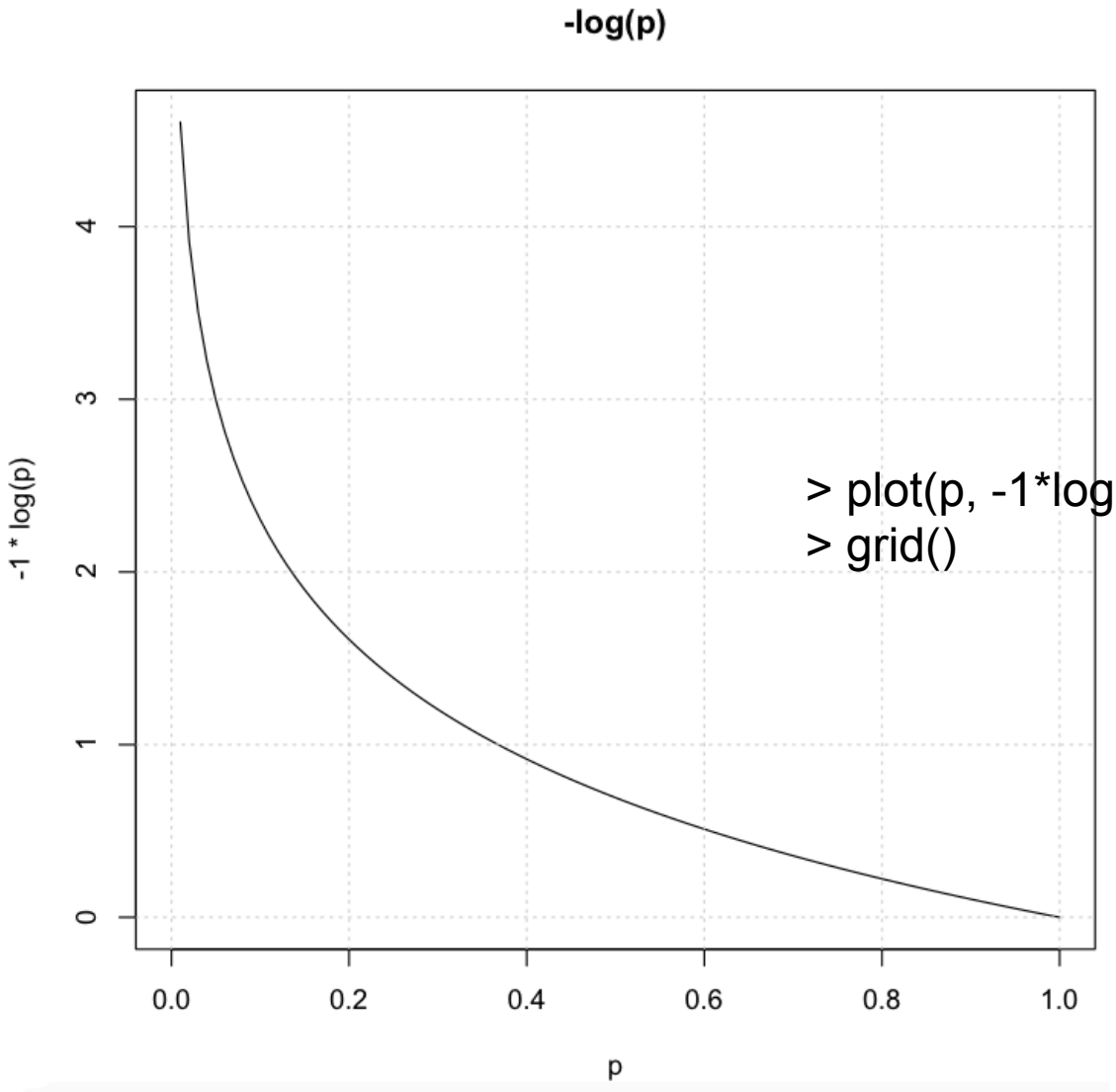
Convex so the loss function is also convex (sum of convex functions)

```
> yTimes_f_of_x = seq(-4, 4, 0.1)
> plot(yTimes_f_of_x, log(1+exp(-1* yTimes_f_of_x),2), main="log(1+exp(-1* yTimes_f_of_x)))",
type="l")
> grid()
>
```

logLoss Function in R

```
> pmax(5:1, pi) #->  5  numbers
[1] 5.000000 4.000000 3.141593 3.141593 3.141593
```

```
LogLoss <- function(actual, predicted, eps=0.00001) {
  #bound the probabilities
  predicted <- pmin(pmax(predicted, eps), 1-eps)
  # log loss
  -1/length(actual)*(sum(actual*log(predicted)+(1-actual)*log(1-predicted)))
}
```




```
> plot(p, -1*log(p), type="l", main="-log(p)")
> grid()
```

A convex logistic regression cost function

Regularization Loss

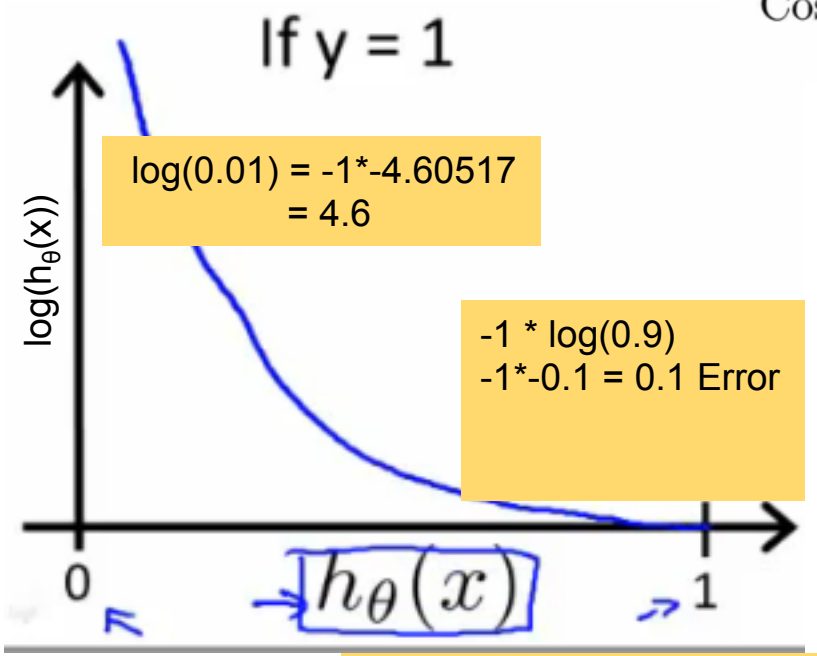
$$J(w) = \lambda \parallel w \parallel^2 - \sum_i \log\left(1 + e^{-y^{(i)} f_w(x^{(i)})}\right)$$

$$-2l(W) = -2\sum_{i=1}^{m} y^i \log p - (1 - y^i)\log(1 - p)$$

If this term close to 1 then 0 loss

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Minimize J(W)



So when we're right, cost function is 0

Else it slowly increases cost function as we become "more" wrong

X axis is what we predict

Y axis is the cost associated with that prediction

If the predicted probability, using β, of the true label yi is close to 1, then the loss is small. But if the predicted probability of yi is close to 0, then the loss is large. Losses are always non-negative; we want to minimize them

http://cseweb.ucsd.edu/~elkan/250B/logreg.pdf

A convex logistic regression cost function

Regularization Loss

$$J(w) = \lambda \parallel w \parallel^2 - \sum_i \log\left(1 + e^{-y^{(i)} f_w(x^{(i)})}\right)$$

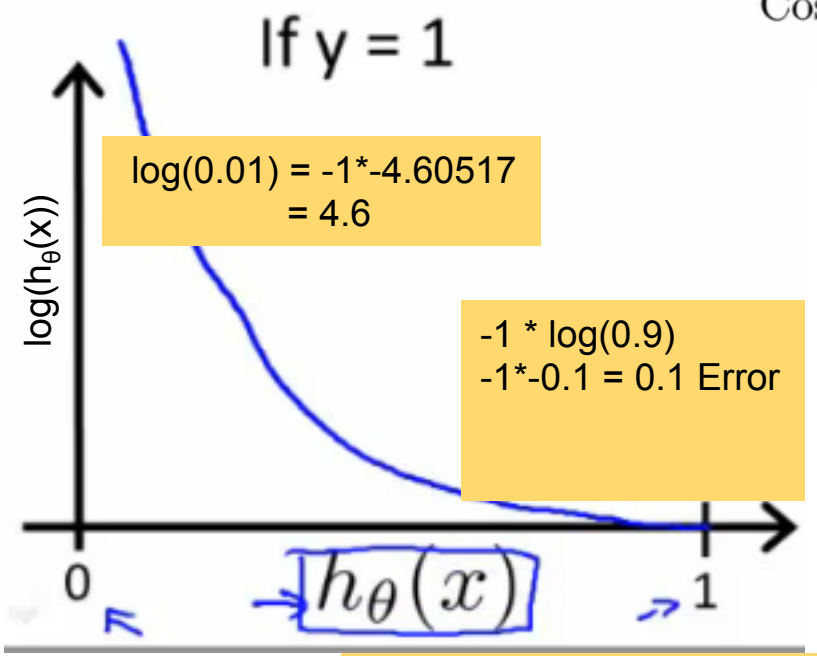
$$-2l(W) = -2\sum_{i=1}^{m} y^i \log p - (1 - y^i)\log(1 - p)$$

If this term close to 1 then 0 loss

Minimize J(W)

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

-log(1-P)



So when we're right, cost function is 0

Else it slowly increases cost function as we become "more" wrong

X axis is what we predict

Y axis is the cost associated with that prediction

Prediction of P=0.1

The actual label y = 0

What is the loss for this example

-log(1-P) = 0.1 loss

If the predicted probability, using β, of the true label yi the loss is small. But if the predicted close to 0, then the loss is large.

s non-negative; we want to minimize them

d.edu/~elkan/250B/logreg.pdf

LR Model Summary

- **summary(mod.mroz.glm)**
- Call:

```
glm(formula = lfp ~ k5 + k618 + age + wc + hc + lwg + inc, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1062  -1.0900   0.5978   0.9709   2.1893

Coefficients:
             Estimate   Std. Error  z value   Pr(>|z|)
(Intercept)  3.182140   0.644375    4.938     7.88e-07 ***
k5          -1.462913   0.197001   -7.426     1.12e-13 ***
k618        -0.064571   0.068001   -0.950     0.342337
age         -0.062871   0.012783   -4.918     8.73e-07 ***
wc           0.807274   0.229980    3.510     0.000448 ***
hc           0.111734   0.206040    0.542     0.587618
lwg          0.604693   0.150818    4.009     6.09e-05 ***
inc         -0.034446   0.008208   -4.196     2.71e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
    Null deviance: 1029.75  on 752  degrees of freedom
Residual deviance:  905.27  on 745  degrees of freedom
AIC: 921.27
Number of Fisher Scoring iterations: 4
```

Residual distribution in logit space
Use summary(predict(mod.mroz.glm, Mroz)) to recover the probabilities

Feature Significance

Null Deviance where phat is #successes/#events
Residual Deviance= -2*LogLikelihood
See later slides for calulcation

AIC = 905+ 2* 8 #7 variables + bias term

Maximum Likelihood Estimator for Logistic Regression

$$P(y=1 \mid X) = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1+\exp(-W^T X)}$$

Cost Function

- Model output $\hat{y}_k = P(y_k = 1 \mid \varphi_k)$
- Likelihood contribution $l_{\theta,k} = \hat{y}_k^{\,y_k} (1-\hat{y}_k)^{1-y_k}$
- Likelihood function $L_\theta = \prod_{k=1}^{N} l_{\theta,k}$
- Log-likelihood function $\ln L_\theta = \sum_{k=1}^{N} (y_k \ln \hat{y}_k + (1-y_k) \ln(1-\hat{y}_k))$

Maximum Likelihood Criterion

$$\max_\theta \ln L_\theta \Leftrightarrow \min_\theta -2 \ln L_\theta$$

Deviance of Null Model W: Set p =proportion successes

- **The deviance of the null model (i.e., no explanatory variables) is defined by**
  - Set p =proportion successes for each training data sample

Note y =1 or 0

$$deviance = -2\sum_{i=1}^{m} y^i \log p - (1 - y^i)\log(1 - p)$$

- **R Code for deviance for MROZ data**

```
mod.mroz.glm <-
glm(lfp ~ k5 + k618 + age +wc + hc + lwg + inc, family=binomial))
coefficients(mod.mroz.glm)
y= lfp
 y= lfp
 phat=(rep(length(lfp[which(lfp==1)])/length(lfp), length(y)))
 nullDeviance= -2 * sum(y*log(phat) + (1-y)*log(1-phat) )
```

1029.746 is the minusTwoTimesLogLikelihood of the data given the null model

Deviance of Learnt Model W

Minimize $-2 * l(W) = -2 \sum_{i=1}^{m} y^i \log p + (1 - y^i) \log(1 - p)$

- **The deviance is defined by**
  - D = -2(L – Lsat)
  - where L is the log-likelihood of our model
  - and Lsat the log-likelihood of the saturated model (with as many variables as observations).
- **R Code for deviance for MROZ data**

```
mod.mroz.glm <-
glm(lfp ~ k5 + k618 + age +wc + hc + lwg + inc, family=binomial))
coefficients(mod.mroz.glm)
phat=mod.mroz.glm$fitted.values
y= lfp
minusTwoTimesLogLik = -2 * sum(y*log(phat) + (1-y)*log(1-phat) )
```

905.266 is the minusTwoTimesLogLikelihood of the data given the learnt model

Metrics are key

- **Metrics are key**
- **Incorporate them into the Gradient Descent**

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Derivations of Logistic Regression and different flavors (i.e., priors)

- **Penalized logistic regression**
  - Laplace (L1) versus gaussian (L2)

Priors on Logistic Regression

- **Penalized logistic regression**
  - Laplace (L1) versus gaussian (L2)

- https://www.dropbox.com/s/he10166scbsmncz/logisticRegression-L1-L2-Laplace-versus-Gaussian.pdf?dl=0

- https://cs.brown.edu/courses/archive/2006-2007/cs195-5/lectures/lecture13.pdf

MAP estimation for logistic regression

- Intuition: similar to the coin toss experiment, we may have some belief about the value of $\mathbf{w}$ before seeing any data.
  - E.g., may prefer smaller values of $\|\mathbf{w}\|$.
- A possible prior that captures that belief:

$$p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}; \mathbf{0}, \sigma^2\mathbf{I}\right).$$

- In the 2D case (again, ignoring w_0) this means

$$p(w_1, w_2) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{w_1^2 + w_2^2}{2\sigma^2}\right).$$

MAP for logistic regression

$$p(w_1, w_2) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{w_1^2 + w_2^2}{2\sigma^2}\right).$$

- Instead of $\ell(X_N; \mathbf{w})$ the objective function (under the Gaussian prior) becomes:

Prior

$$\tilde{\ell}(X_N; \mathbf{w}, \sigma) = \ell(X_N; \mathbf{w}) + \log p(\mathbf{w})$$

$$= \sum_{i=1}^{N} \log p\left(y_i \mid \mathbf{x}_i; \mathbf{w}\right) - \frac{1}{2\sigma^2}(w_1^2 + w_2^2) + \text{const}(\mathbf{w}).$$

- This is a *penalized log-likelihood* (or *log-posterior*).
- Note that $w_1^2 + w_2^2 = \|\mathbf{w}\|^2$.
- Setting σ^2 will affect the penalty we impose for a particular value of $\|\mathbf{w}\|$.

Recall we assume σ^2 will affect the penalty for a particular value of ||W||

Log of a normal distribution: constant and a term that depends on W

- Each w has a probability, $f(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w-\mu)^2}{2\sigma^2}}$
- Likelihood, L is $L_{w_1,w_2,\ldots,w_N}(\mu, \sigma^2)$

$$L_{w_1,\ldots,w_N}(\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w_1-\mu)^2}{2\sigma^2}} * \ldots * \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w_N-\mu)^2}{2\sigma^2}}$$

- The log likelihood, $l = \ln L$, is

Constant

$$l_{w_1,\ldots,w_N}(\mu, \sigma^2) = -\frac{n}{2} \ln 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (w_i - \mu)^2$$

$$p(w_1, w_2) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{w_1^2 + w_2^2}{2\sigma^2}\right).$$

Recall we assume W is centred on zero with deviation σ

Scaled objective

- When N is large, it may be more convenient to work with

$$\tilde{\ell}(X_N; \mathbf{w}, \sigma) \;=\; \frac{1}{N}\sum_{i=1}^{N} \log p\left(y_i \,|\, \mathbf{x}_i; \mathbf{w}\right) \;-\; \frac{1}{2N\sigma^2}\|\mathbf{w}\|^2.$$

- The parameter $\lambda = \frac{1}{2\sigma^2}$ determines the *strength* of regularization.
  - Higher $\lambda \Rightarrow$ more weight on the prior.

L_2 versus L_1 regularization

- We can have a different penalty term. For instance,

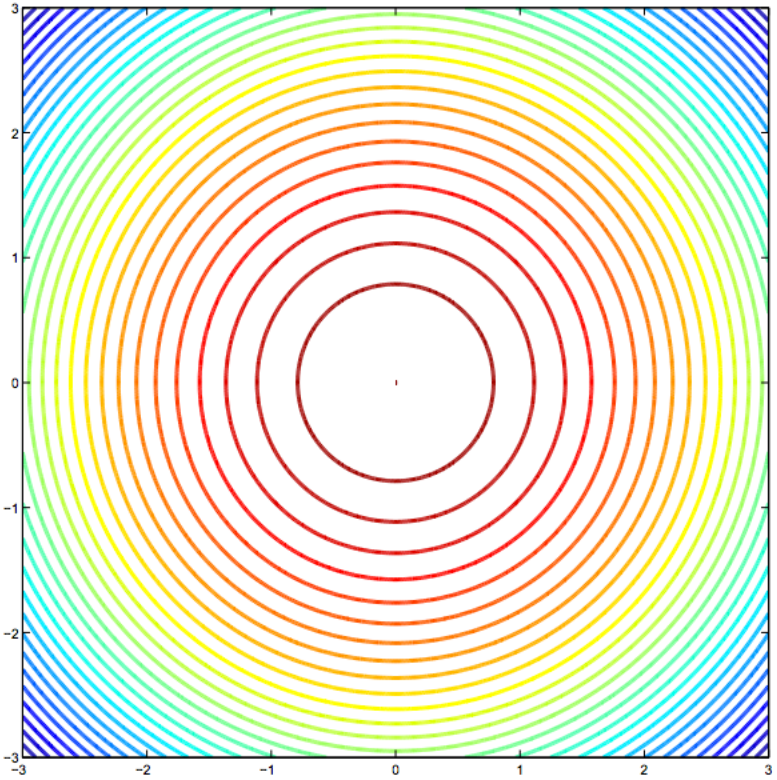
$$\tilde{\ell}(X_N; \mathbf{w}, \sigma) = \sum_{i=1}^{N} \log p\left(y_i \,|\, \mathbf{x}_i; \mathbf{w}\right) - \lambda \sum_{j=1}^{d} |w_j| + \text{const}(\mathbf{w}),$$

the L_1-norm penalty.

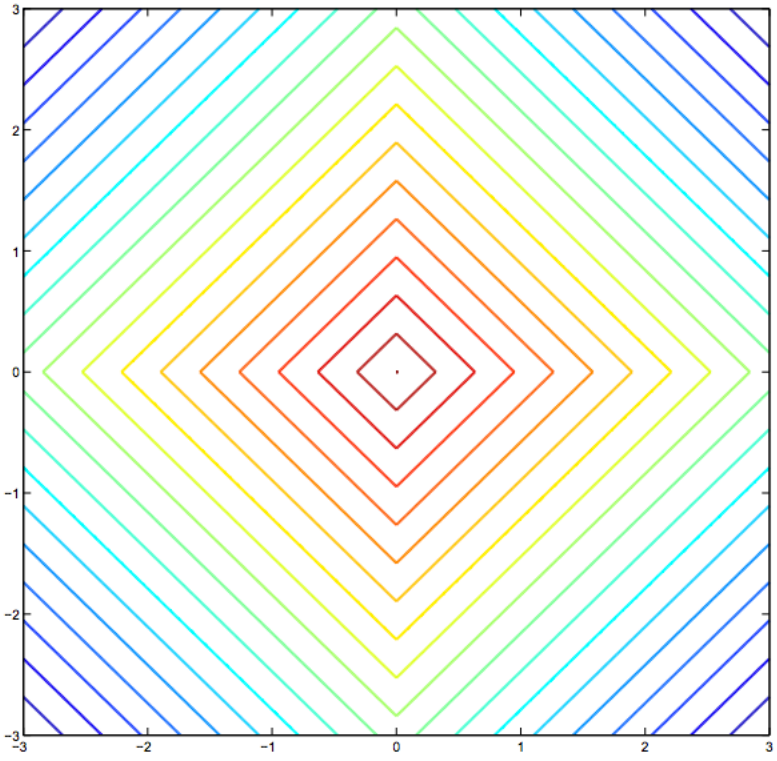
- the L_p norm of $\mathbf{w}$ is $\left(\sum_{j=1}^{d} w_j^p\right)^{1/p}$.
- L_2: Euclidean distance.
- L_1: Manhattan distance.

L_2 versus L_1 regularization

$-\lambda \sum_{j=1}^{d} w_j^2$



$-\lambda \sum_{j=1}^{d} |w_j|$

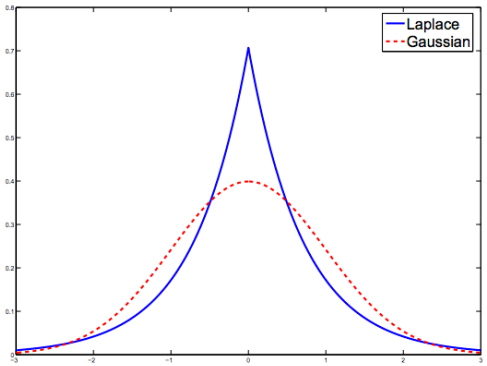


Probabilistic interpretation of L_1 penalty

Lasso Regression:

- Laplacian pdf:

$$p(x; a) = \frac{a}{2} e^{-a|x|}$$



- L_1 drives $\mathbf{w}$ to zero more efficiently.

A random variable has a Laplace(μ, b) distribution if its probability density function is

$$f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$
$$= \frac{1}{2b} \begin{cases} \exp\left(-\frac{\mu - x}{b}\right) & \text{if } x < \mu \\ \exp\left(-\frac{x-\mu}{b}\right) & \text{if } x \geq \mu \end{cases}$$

Here, μ is a location parameter and $b > 0$, which is sometimes referred to as the diversity, is a scale parameter. If $\mu = 0$ and $b = 1$, the positive half-line is exactly an exponential distribution scaled by 1/2.

The probability density function of the Laplace distribution is also reminiscent of the normal distribution; however, whereas the normal distribution is expressed in terms of the squared difference from the mean μ, the Laplace density is expressed in terms of the absolute difference from the mean. Consequently the Laplace distribution has fatter tails than the normal distribution.

Logistic Regression via Distributed Gradient Descent

- **Master/Driver Process**
- **Initialize model parameters, W= vector of zeros**

$$W = (0,0,....)$$

- **While not converged**
  - Broadcast model (e.g., weight vector) to the worker nodes
  - Mapper (MANY mappers)
    - Compute partial gradient for each training example
    - Combine in memory
    - Finally Yield the partial gradient
  - Reducer (single Reducer)
    - Aggregate partial gradients
    - Yield full gradient
  - Check for convergence
- **End-While**

Create the gradient mapper function

And the rest is the same

$$W_{t+1} = W_t + \alpha \sum (y_i - p) X_{i+1}$$

Related Maximum Likelihood Approaches to LR

- **Probit Regression**
- **Avoiding Overfitting via Regularization**

$$W \leftarrow \arg\max_W \sum_l \ln P(Y^l|X^l,W) - \frac{\lambda}{2}||W||^2$$

Penalized log likelihood function

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l,W)) - \lambda w_i$$

Gradient

- **Multiclass LR**

$y_1, y_2, ..y_{k-1}$

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^n w_{ki}X_i)}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

When $Y = y_K$, it is

Y_k

$$P(Y = y_K|X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

- **An extension of the logistic model to sets of interdependent variables is the conditional random field.**

Probit versus Logit

Probit Regression versus Logistic Resression

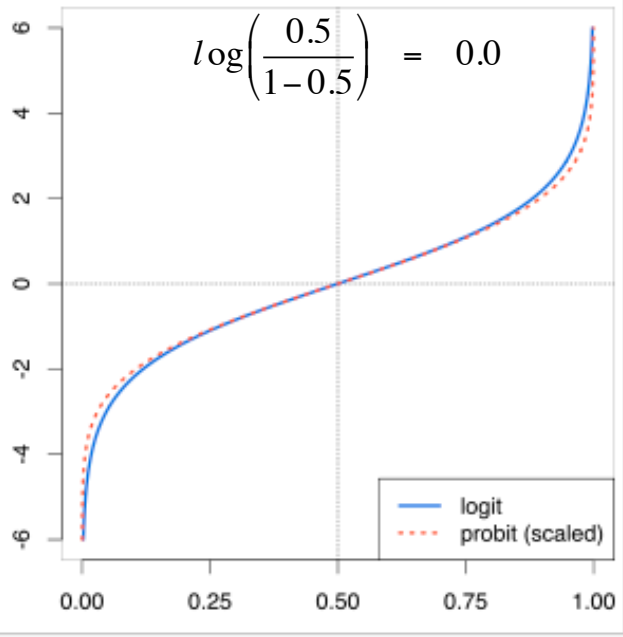
$$\log\left(\frac{p}{1-p}\right) = W^T X$$

Comparison with probit [edit]

Closely related to the logit function (and logit model) are the probit function and probit model. The logit and probit are both sigmoid functions with a domain between 0 and 1, which makes them both quantile functions — i.e. inverses of the cumulative distribution function (CDF) of a probability distribution. In fact, the logit is the quantile function of the logistic distribution, while the probit is the quantile function of the normal distribution. The probit function is denoted $\Phi^{-1}(x)$, where $\Phi(x)$ is the CDF of the normal distribution, as just mentioned:

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx$$

As shown in the graph, the logit and probit functions are extremely similar, particularly when the probit function is scaled so that its slope at y=0 matches the slope of the logit. As a result, probit models are sometimes used in place of logit models because for certain applications (e.g. in Bayesian statistics) the implementation is easier.



Comparison of the logit function with a scaled probit (i.e. the inverse CDF of the normal distribution), comparing $\text{logit}(x)$ vs. $\Phi^{-1}(x)/\sqrt{\frac{\pi}{8}}$, which makes the slopes the same at the y-origin.

See also [edit]

- Discrete choice on binary logit, multinomial logit, conditional logit, nested logit, mixed logit, exploded logit, and ordered logit
- Limited dependent variable
- Daniel McFadden, a Bank of Sweden Prize in Economic Sciences in Memory of Alfred Nobel winner for development of a particular logit model used in economics[2]
- Logit analysis in marketing
- Multinomial logit

Jacobian and Hessian for LR

$$W^{(new)} = W^{(old)} - [\nabla^2 \sigma(W^{(old)})]^{-1} \nabla f(W^{(old)}) \quad (27)$$

Then we can derive

$$\nabla E(W) = \sum_{n=1}^{N} (W^T X_n - Y') X_n \quad (28)$$

$$= X^T X W - X^T Y' \quad (29)$$

$$\nabla^2 E(W) = \sum_{n=1}^{N} X_n X_n^T \quad (30)$$

$$= X^T X \quad (31)$$

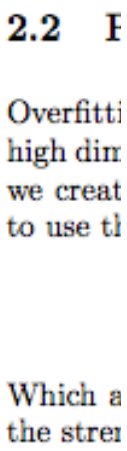
Plug in equation (27), we can derive

$$W^{(new)} = W^{(old)} - (X^T X)^{-1} \{X^T X W^{(old)} - X^T Y\} \quad (32)$$

$$= (X^T X)^{-1} X^T Y \quad (33)$$

Regularization through priors

- **Maximum likelihood estimation**
  - They are typically determined by some sort of optimization procedure, e.g. maximum likelihood estimation, that finds values that best fit the observed data (i.e. that give the most accurate predictions for the data already observed), usually subject to regularization conditions that seek to exclude unlikely values, e.g. extremely large values for any of the regression coefficients.
- **The use of a regularization condition is equivalent to doing maximum a posteriori (MAP) estimation, an extension of maximum likelihood.**
- **Regularization is most commonly done using a squared regularizing function, which is equivalent to placing a zero-mean Gaussian prior distributionon the coefficients, but other regularizers are also possible.)**
- **Find solution using an iterative numerical method must be used, such as gradient descent**


2.2 F

Overfitti
high dim
we creat
to use tl

Which a
the strer

Regularized Logistic Regession

- **Avoiding Overfitting via Regularization**

$$W \leftarrow \arg\max_W \sum_l \ln P(Y^l|X^l,W) - \frac{\lambda}{2}||W||^2$$

Penalized log likelihood function

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l,W)) - \lambda w_i$$

Gradient

Related Maximum Likelihood Approaches to LR

- **Probit Regression**
- **Avoiding Overfitting via Regularization**

$$W \leftarrow \arg\max_W \sum_l \ln P(Y^l|X^l,W) - \frac{\lambda}{2}||W||^2$$

Penalized log likelihood function

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l,W)) - \lambda w_i$$

Gradient

- **Multiclass LR**

$y_1, y_2, ..y_{k-1}$

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^n w_{ki}X_i)}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

When $Y = y_K$, it is

Y_k

$$P(Y = y_K|X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji}X_i)}$$

- **An extension of the logistic model to sets of interdependent variables is the conditional random field.**

Bayesian Logistic Regression

- **Maximum likelihood approaches**
  - Probit Regression
  - An extension of the logistic model to sets of interdependent variables is the conditional random field (sequences of data, e.g., in NLP).

- **Bayesian Logistic Regression**
  - approximation method such as the Metropolis–Hastings algorithm.
  - MCMC methods have sequential dependencies so makes them less attractive for distributed frameworks
  - E.g. in Spark see,
    - http://blog.cloudera.com/blog/2014/08/bayesian-machine-learning-on-apache-spark/

Outline

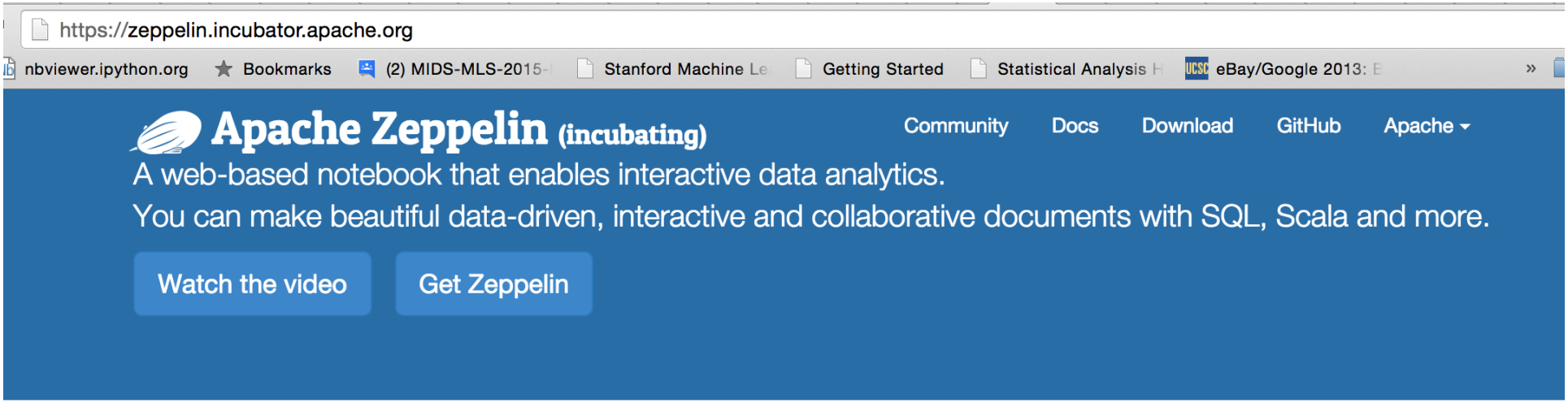
- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**

Zeppelin Notebooks locally and on AWS

James G. Shanahan

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

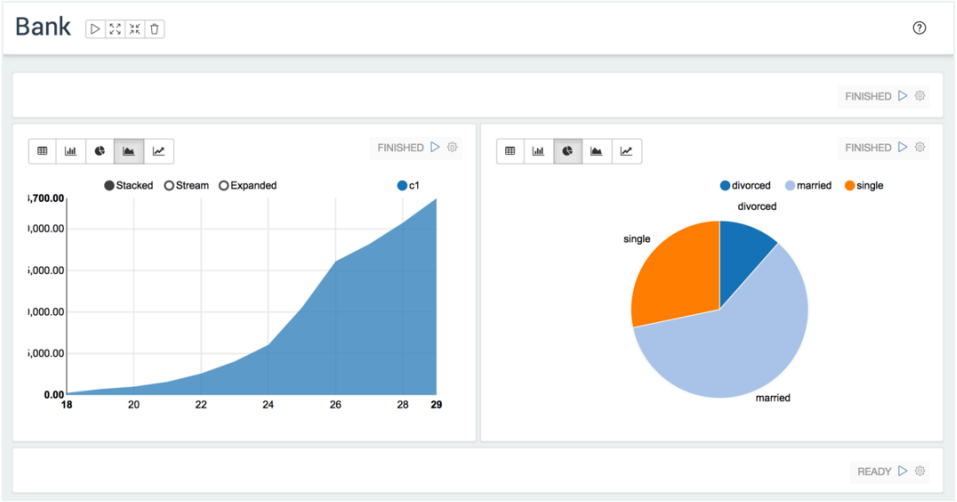
Large Scale Machine Learning
MIDS, UC Berkeley



Multi-purpose Notebook

The Notebook is the place for all your needs

- Data Ingestion
- Data Discovery
- Data Analytics
- Data Visualization & Collaboration



Multiple language backend

Multiple language backend

Zeppelin interpreter concept allows any language/data-processing-backend to be plugged into Zeppelin. Currently Zeppelin supports many interpreters such as Scala(with Apache Spark), Python(with Apache Spark), SparkSQL, Hive, Markdown and Shell.

```
val s = "Scala with built-in Apache Spark Integration"
```

```
s: String = Scala with built-in Apache Spark Integration
```

Took 0 seconds

```
%pyspark
print "Python with built-in Apache Spark Integration"
```

```
Python with built-in Apache Spark Integration
```

Took 0 seconds

```
%sql -- built-in SparkSQL Support
select * from RDD
```

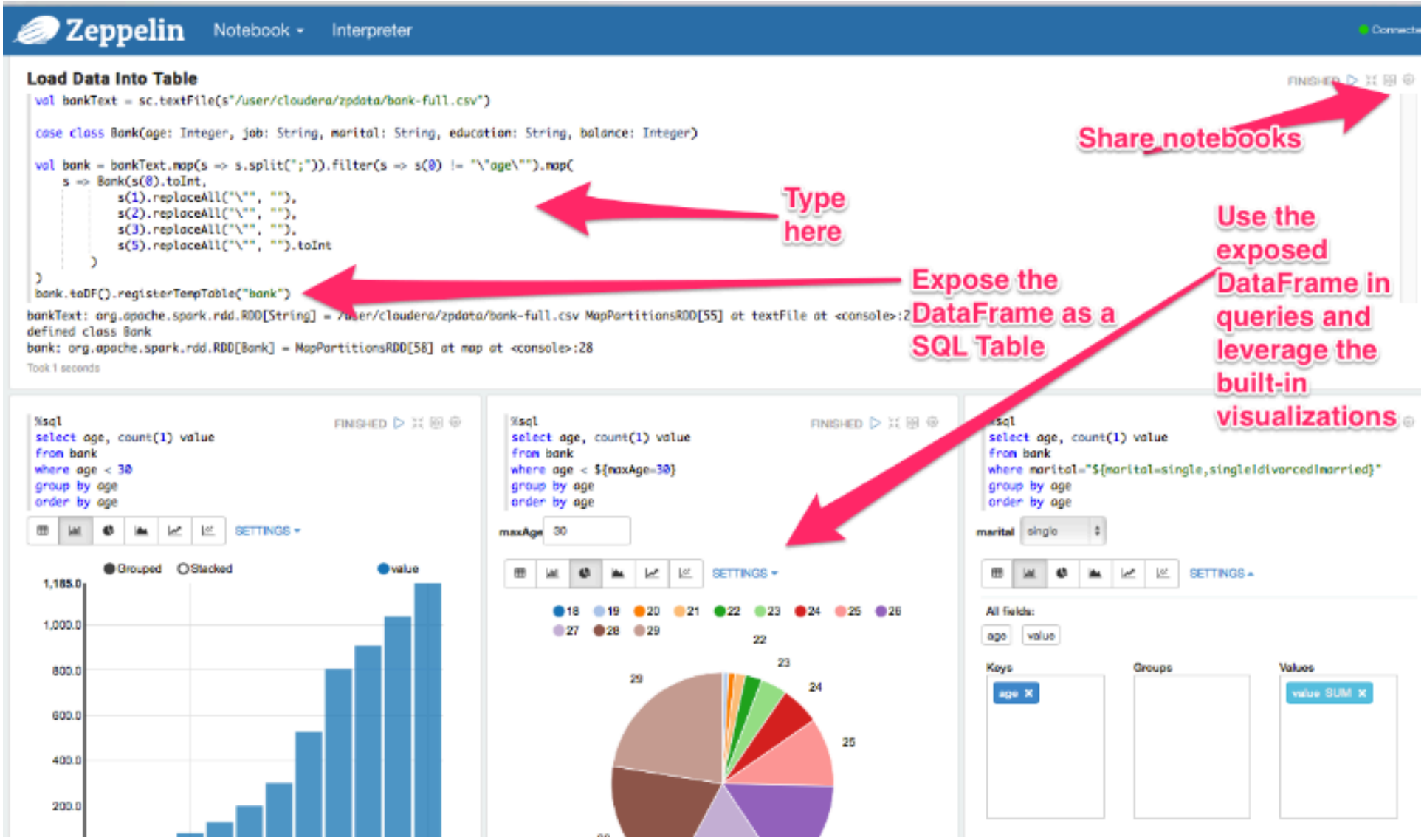
```
%md Markdown
```

Markdown

Took 0 seconds

```
%sh echo "Shell"
```

```
Shell
```

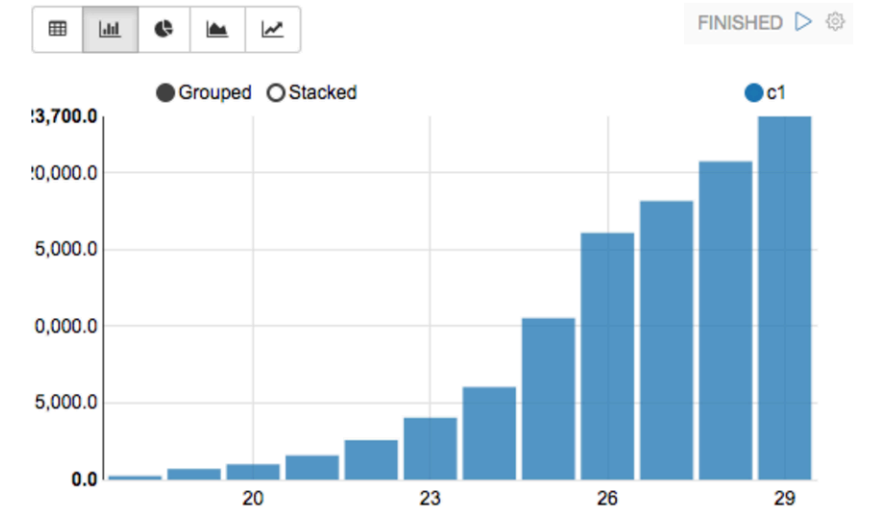

Zeppelin
Notebook
Interpreter
Load Data Into Table
Share notebooks
Type here
Expose the DataFrame as a SQL Table
Use the exposed DataFrame in queries and leverage the built-in visualizations
SETTINGS
Grouped
Stacked
value
maxAge
marital
All fields:
Keys
Groups
Values

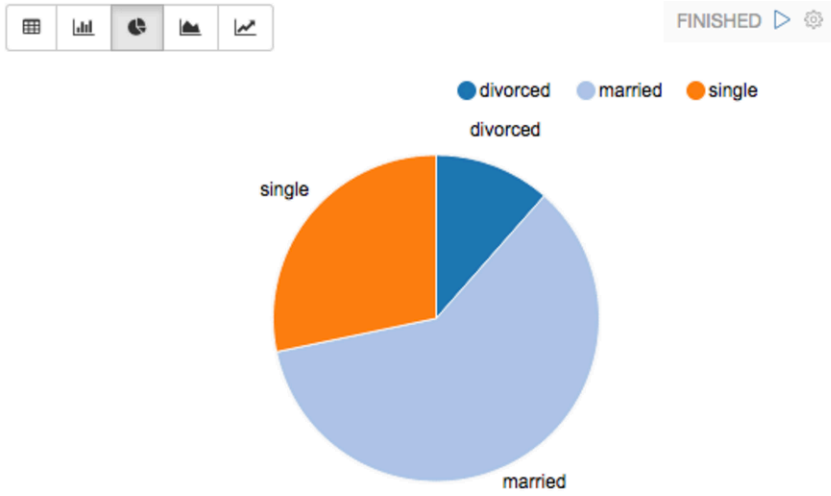
Zeppelin's Spark integration provides

- Automatic SparkContext and SQLContext injection
- Runtime jar dependency loading from local filesystem or maven repository. Learn more about dependency loader.
- Canceling job and displaying its progress

Data visualization

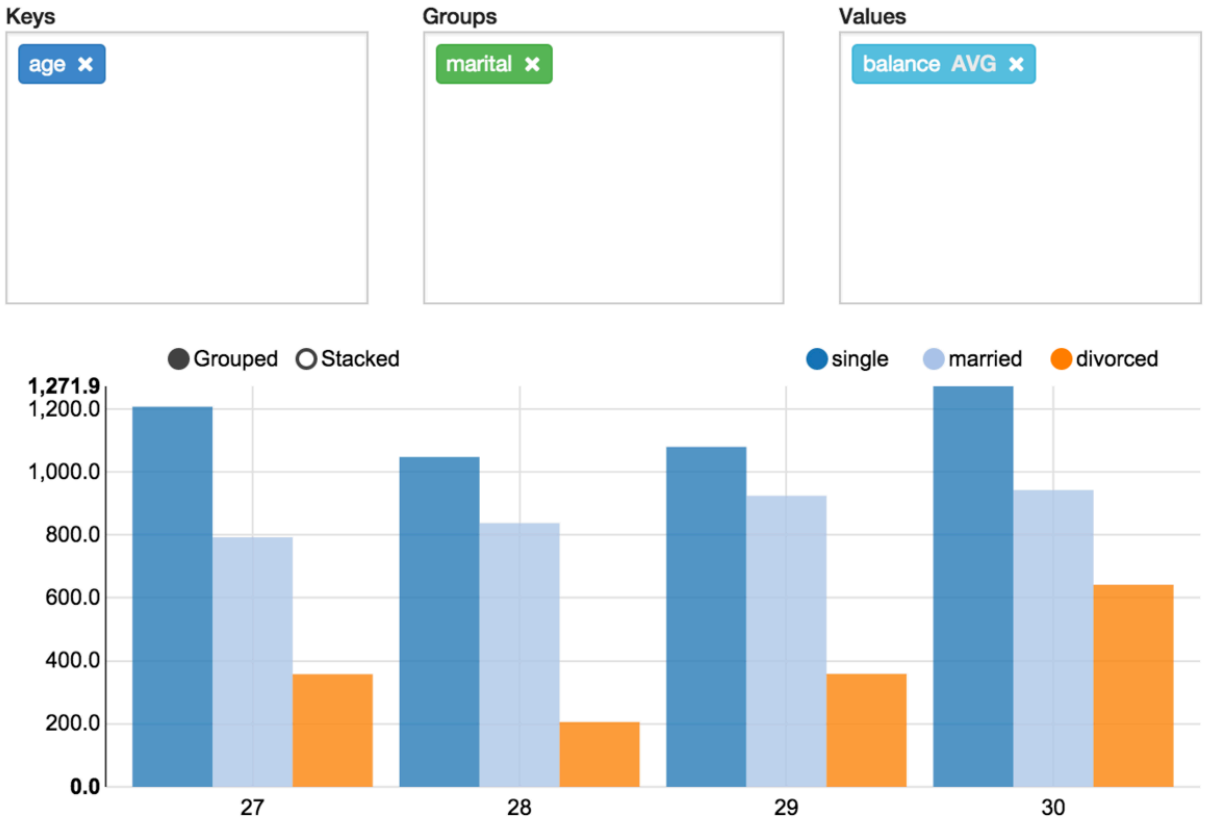
Some basic charts are already included in Zeppelin. Visualizations are not limited to SparkSQL's query, any output from any language backend can be recognized and visualized.





Pivot chart

With simple drag and drop Zeppelin aggeregates the values and display them in pivot chart. You can easily create chart with multiple aggregated values including sum, count, average, min, max.

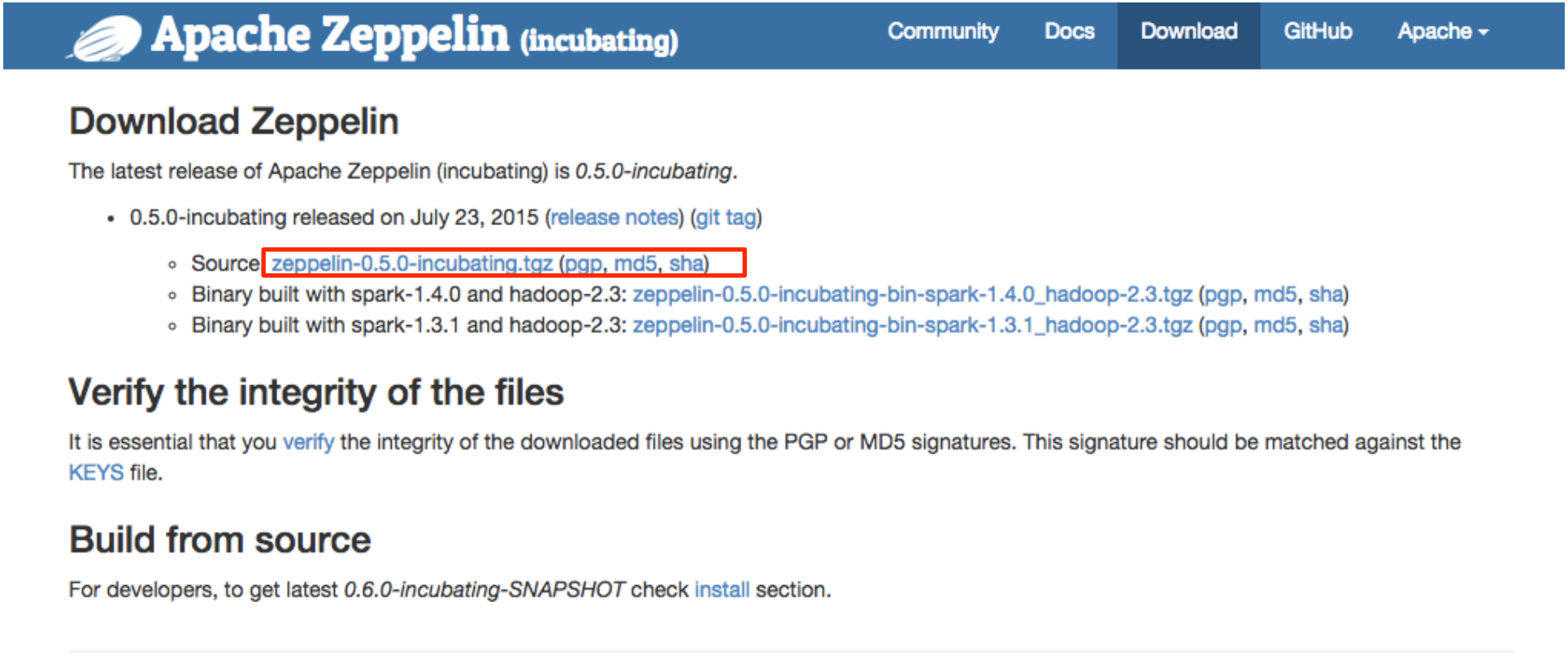


Learn more about Zeppelin's Display system. (text, html, table, angular)

- **Zeppelin on your local computer**

Build from Source

- **Download Source code**
  - https://zeppelin.incubator.apache.org/download.html

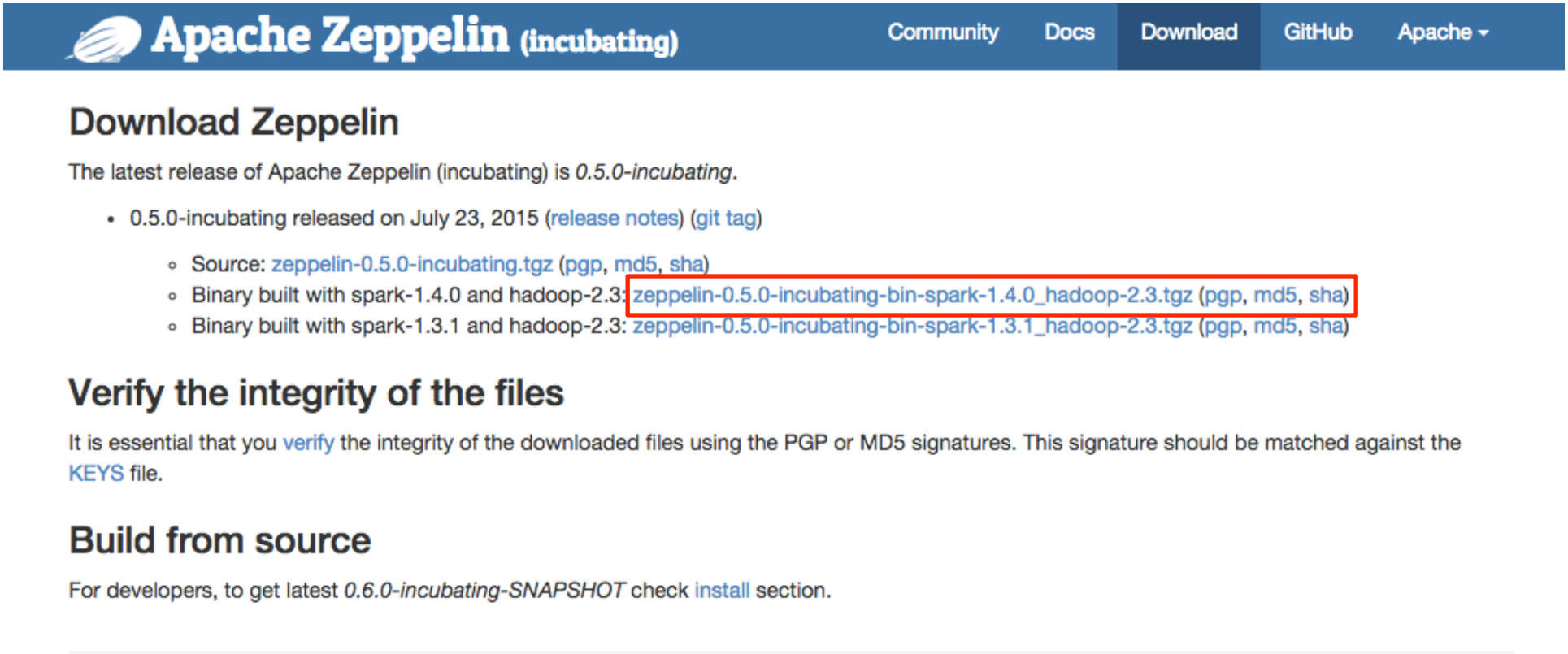


Build from Source

- **Install maven if you don't have it**
  - brew install maven
- **Unzip the source code and follow instructions**
  - http://www.makedatauseful.com/apache-zeppelin-on-osx-ultra-quick-start/

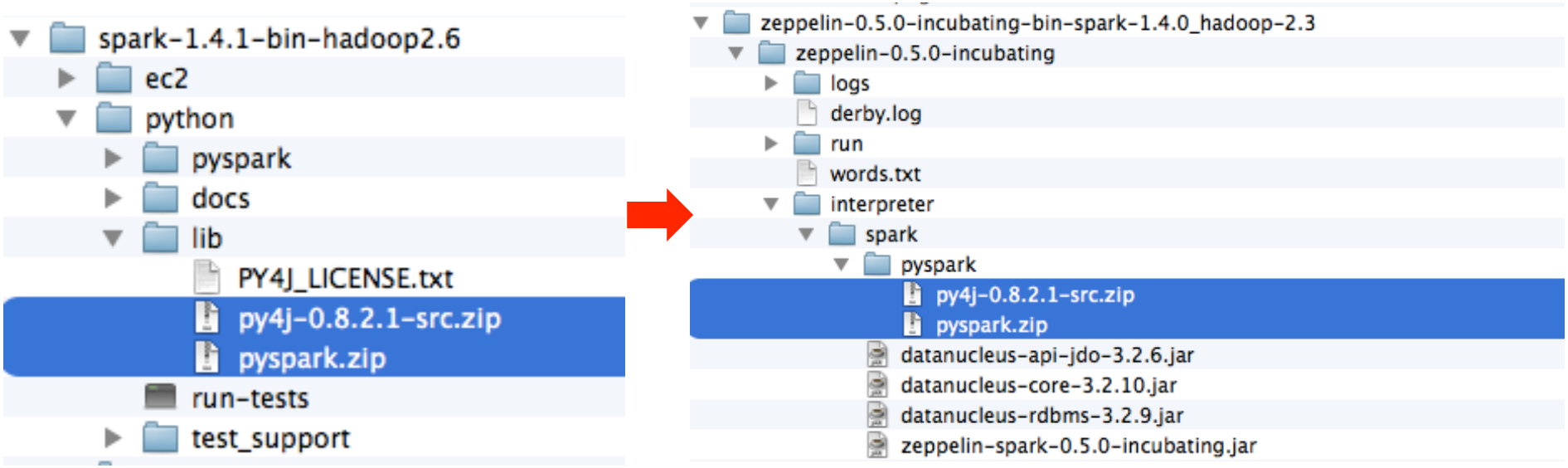
Binary built

- **Download**



For Binary Version

- **Unzip the file**
- **Copy two files from spark installation folder to the 'interpreter' folder**



Start it up Zeppelin on your local machine

- **To Start**
  - bin/zeppelin-daemon.sh start
- **In your browser:**
  - http://localhost:8080
- **To STOP**
  - bin/zeppelin-daemon.sh stop&

- **Zeppelin on the AWS cloud**

Zeppelin as an alternative to Jupyter Notebooks

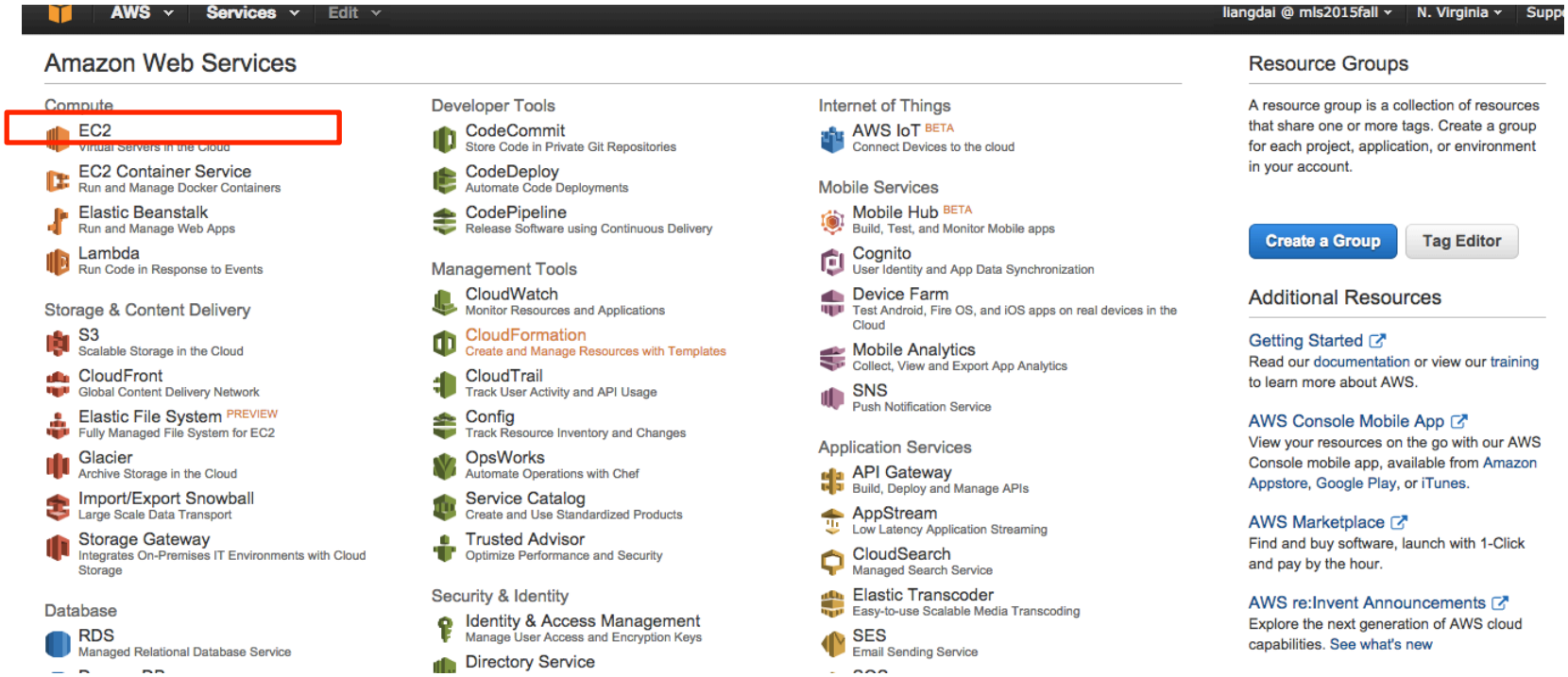
- Currently Zeppelin notebooks on AWS have some system bugs
- But it works for pySpark

Outline

- Create EC2 keypair
- Create EMR cluster (include Spark and Zeppelin)
- Connect to EMR and Run Zeppelin

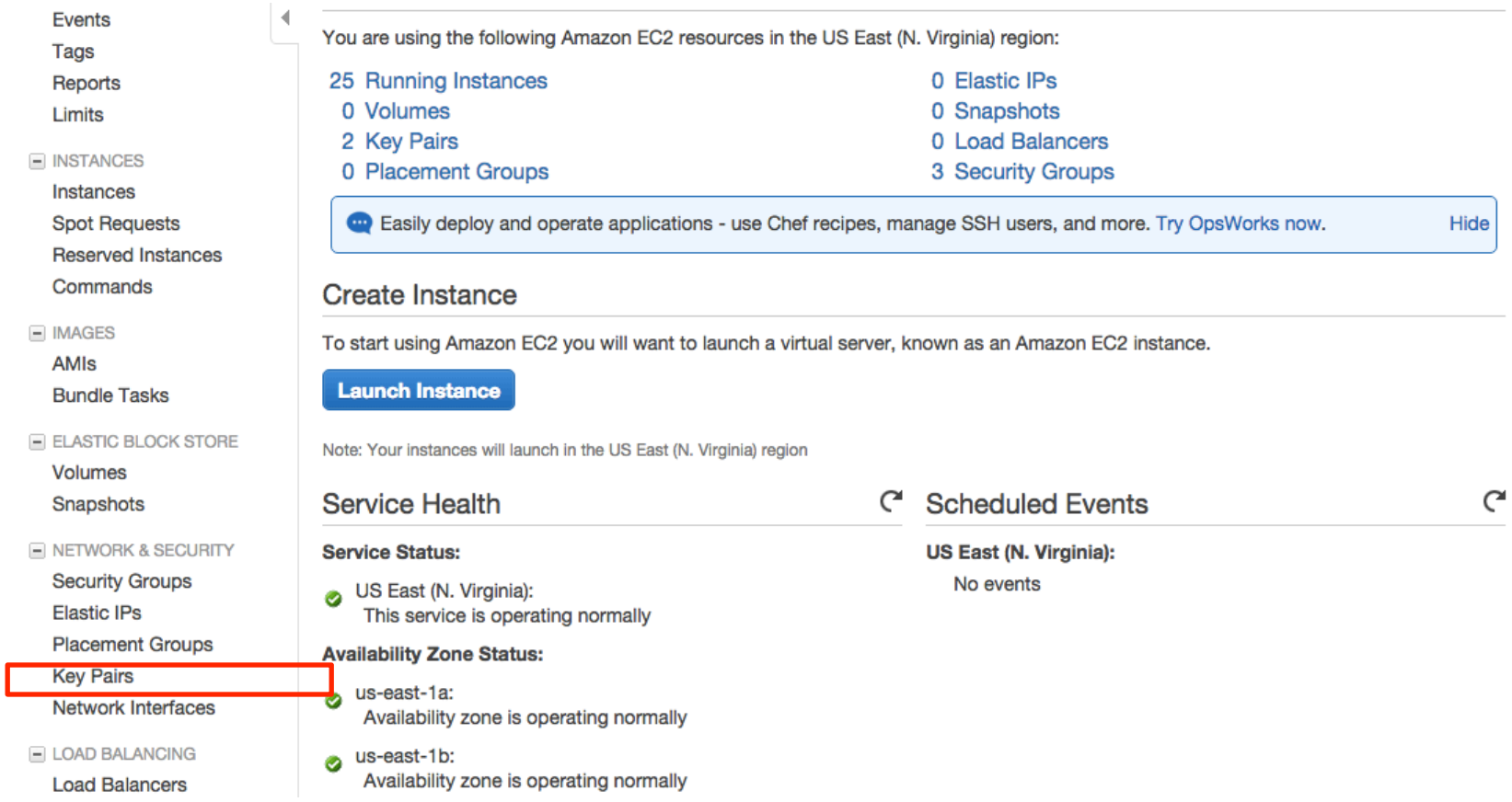
Create EC2 key pair

Click EC2 after logging in



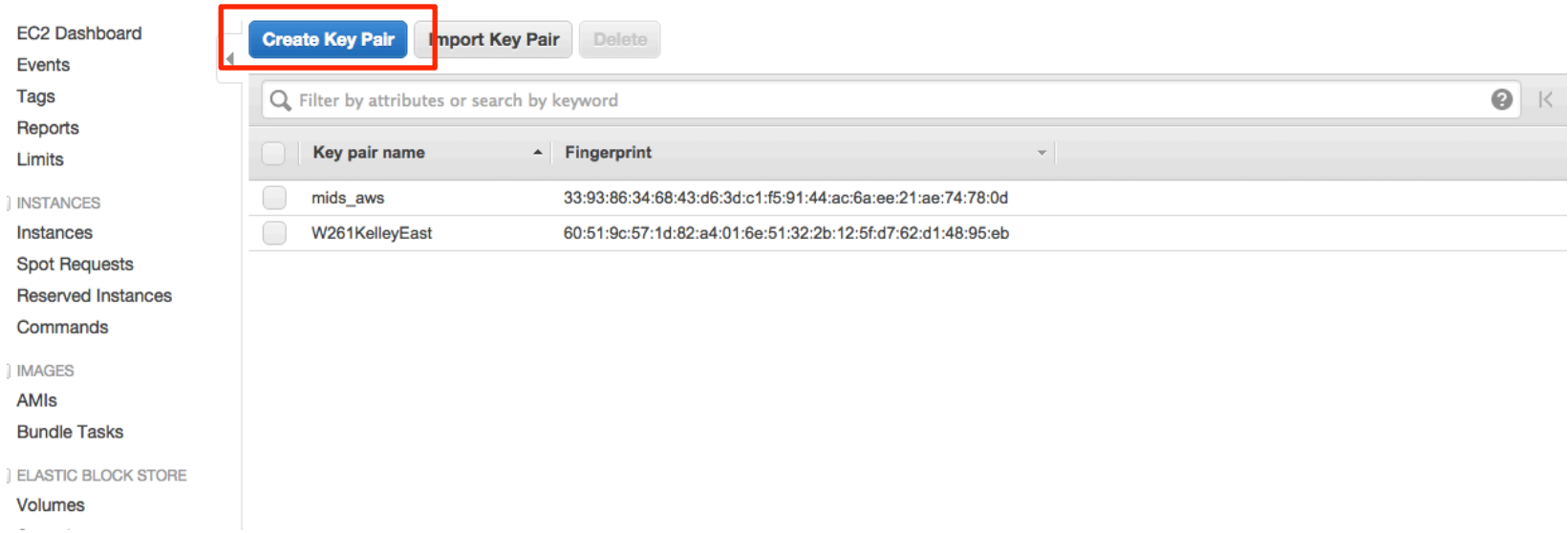
Create EC2 key pair

Click Key Pairs



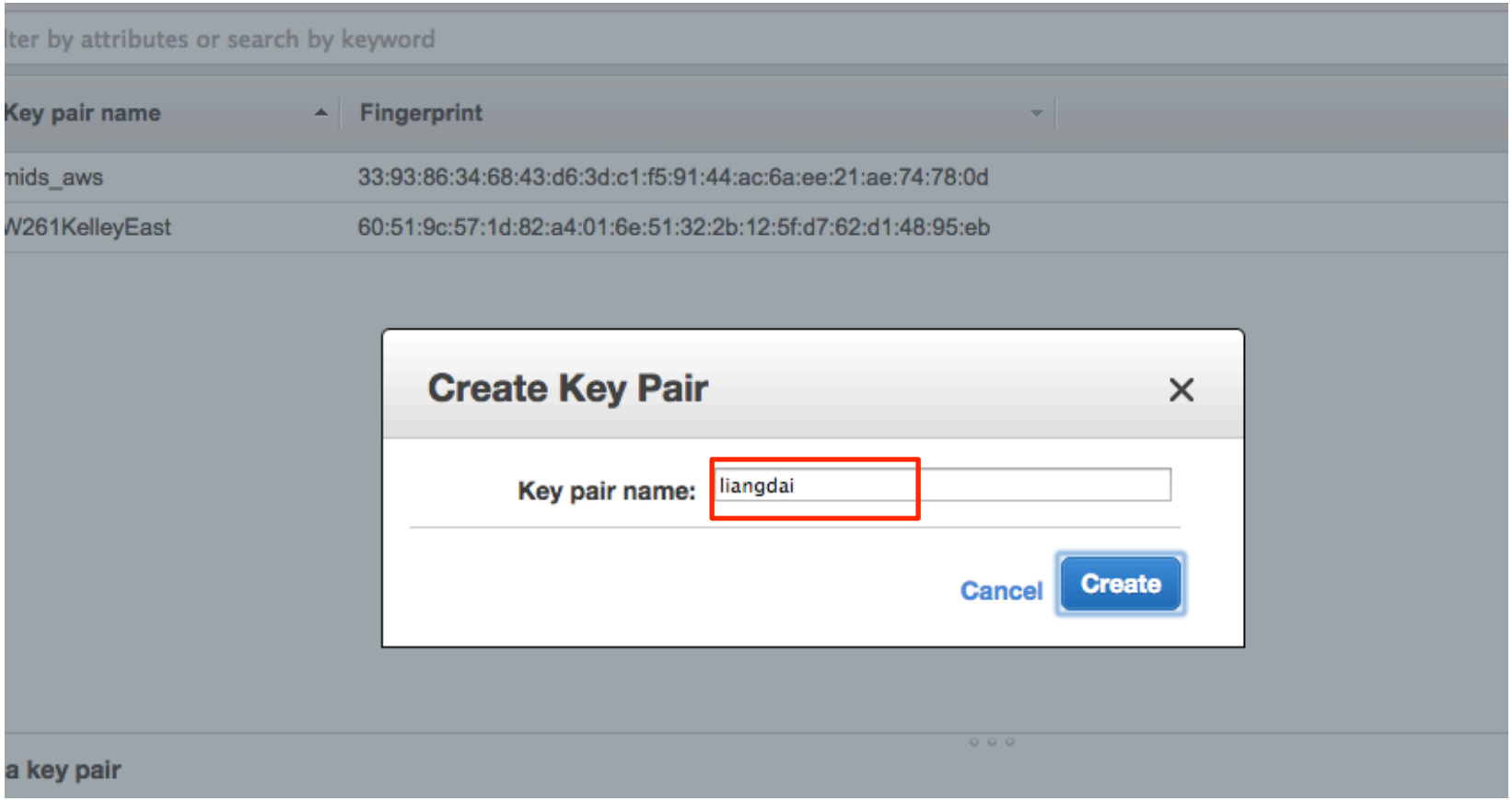
Create EC2 key pair

Create Key Pair



Create EC2 key pair

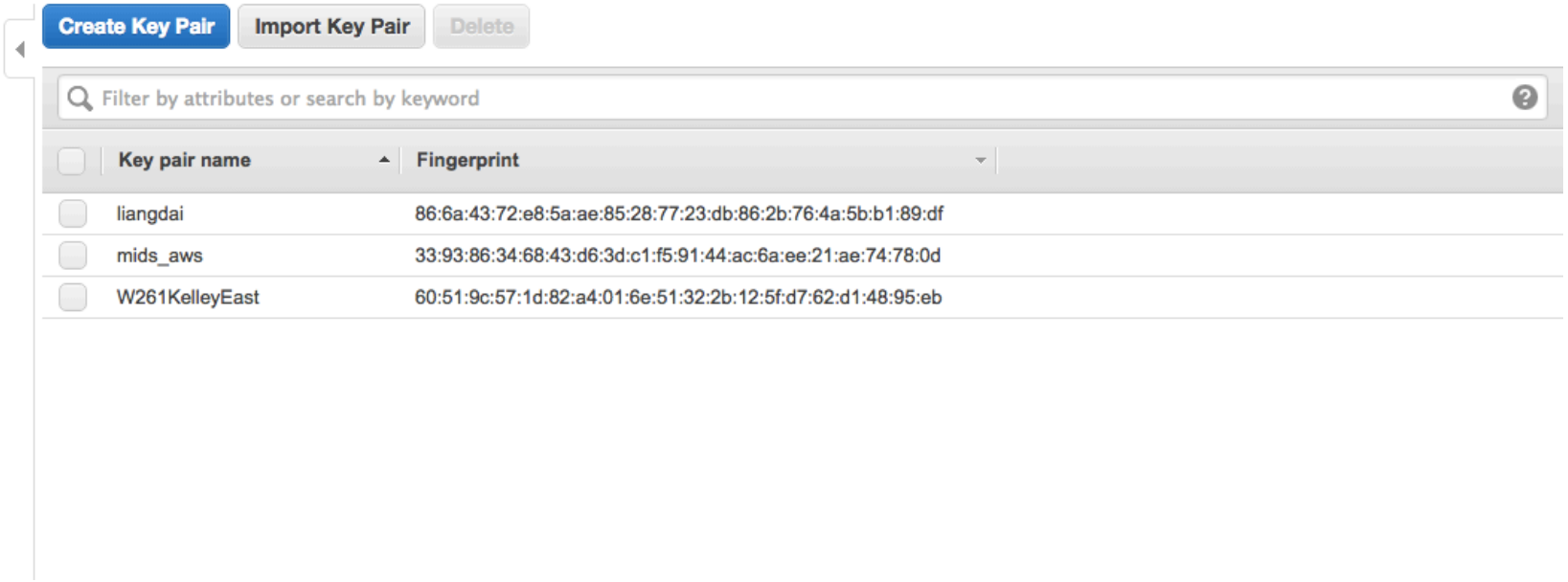
Name the key pair



Create EC2 key pair

Save .pem file

In windows, you need puttygen to generate .ppk file from .pem file



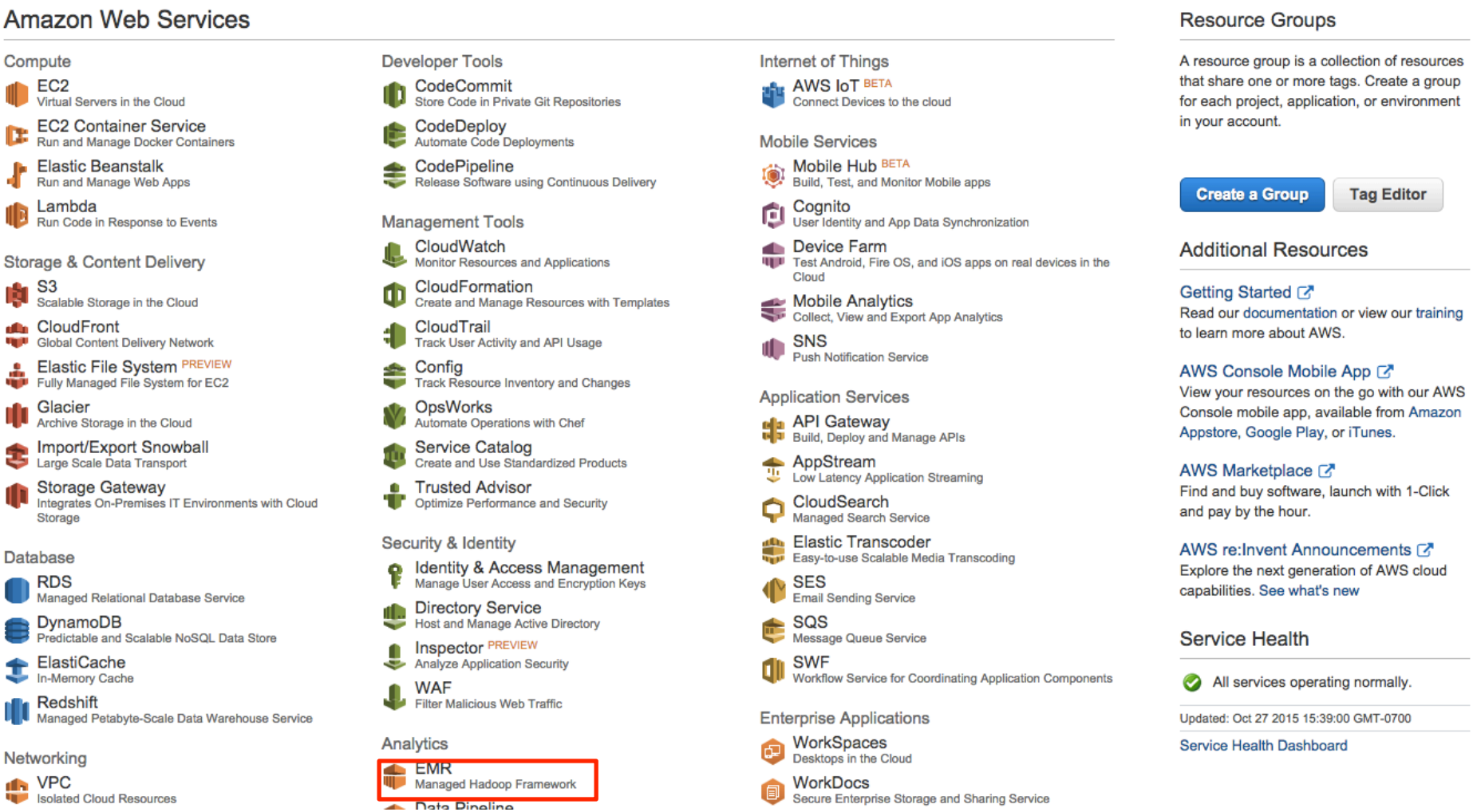
Create EMR cluster

- **EMR cluster can be created on webpage or through AWS CLI.**

- **Webpage is recommended**
  - The instruction are listed on webpage

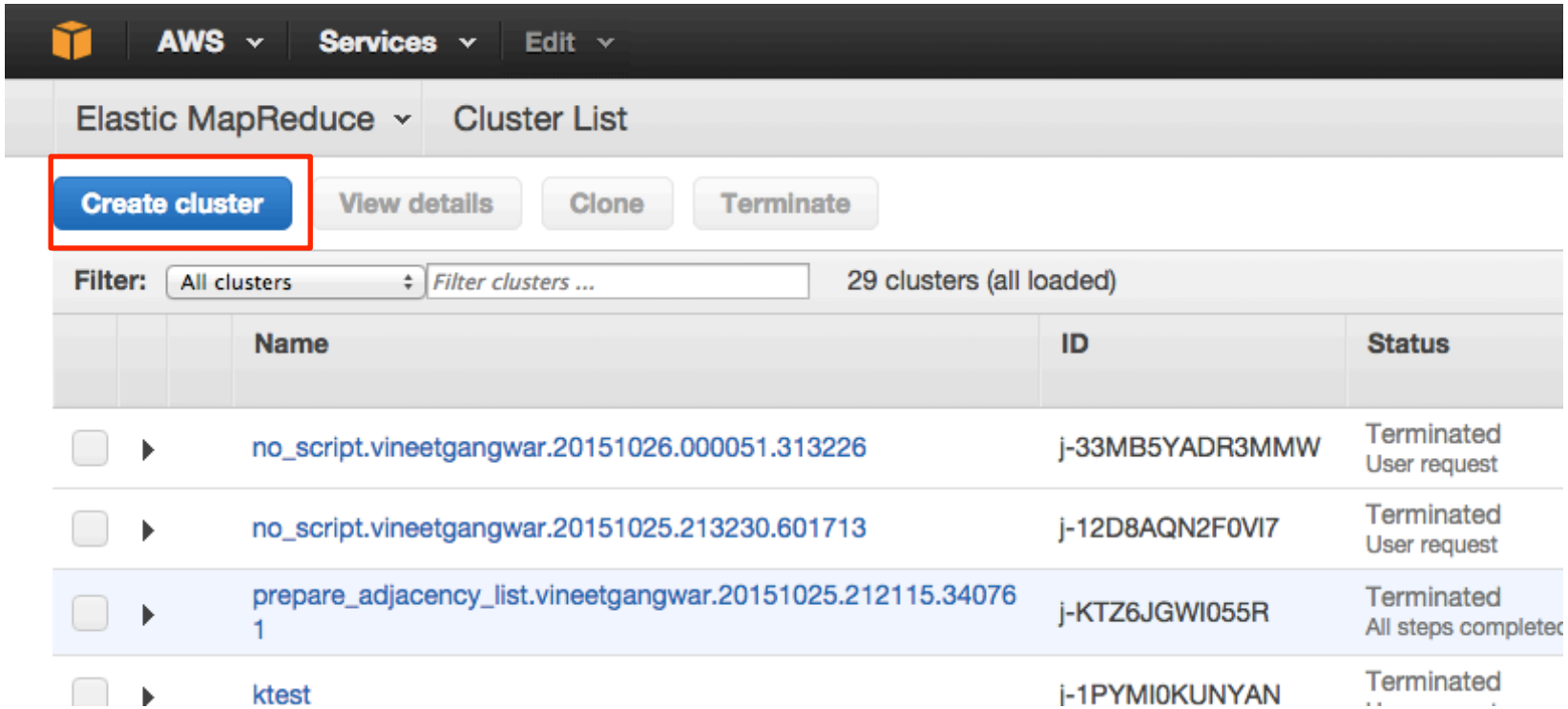
Create EMR cluster

Click EMR after login



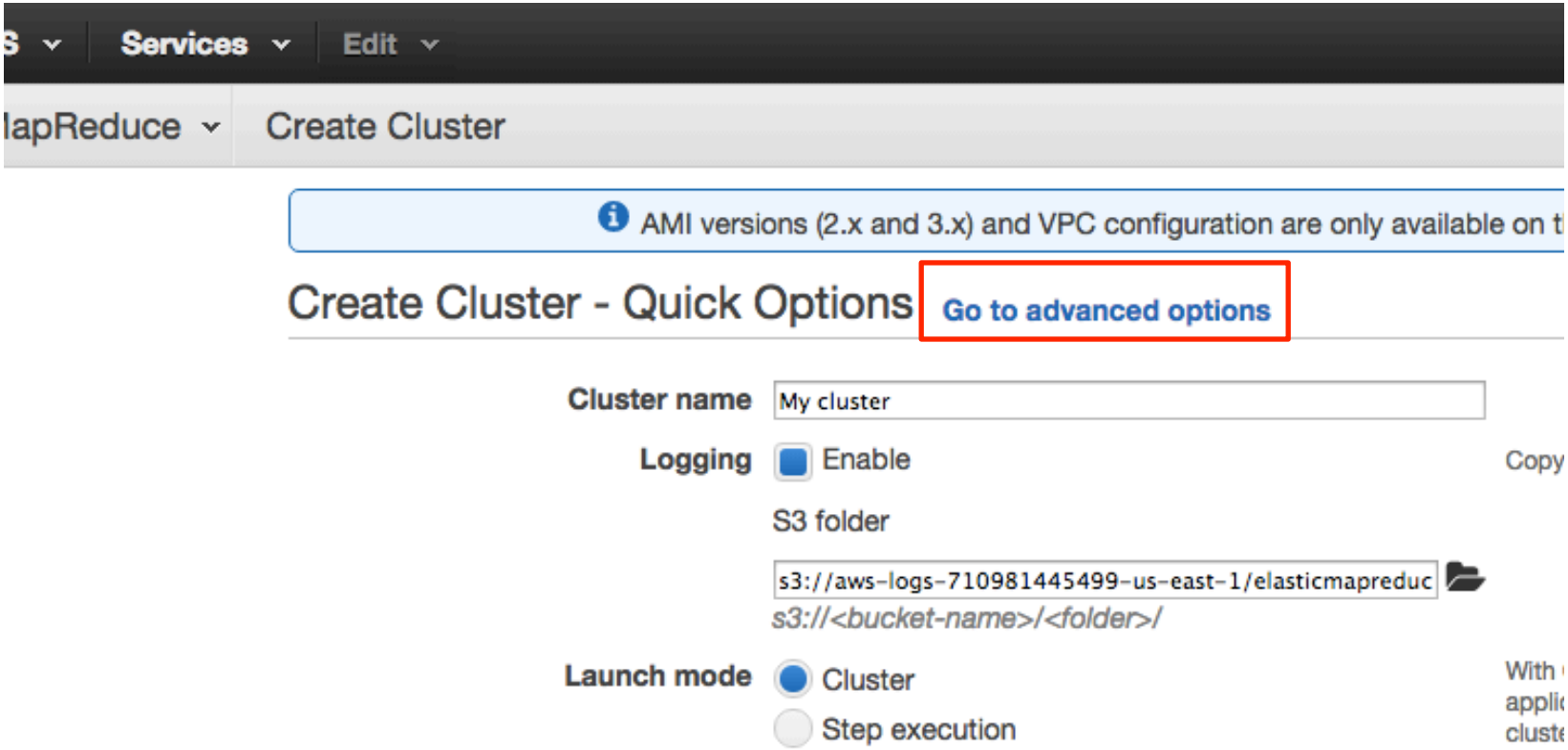
Create EMR cluster

Click Create cluster



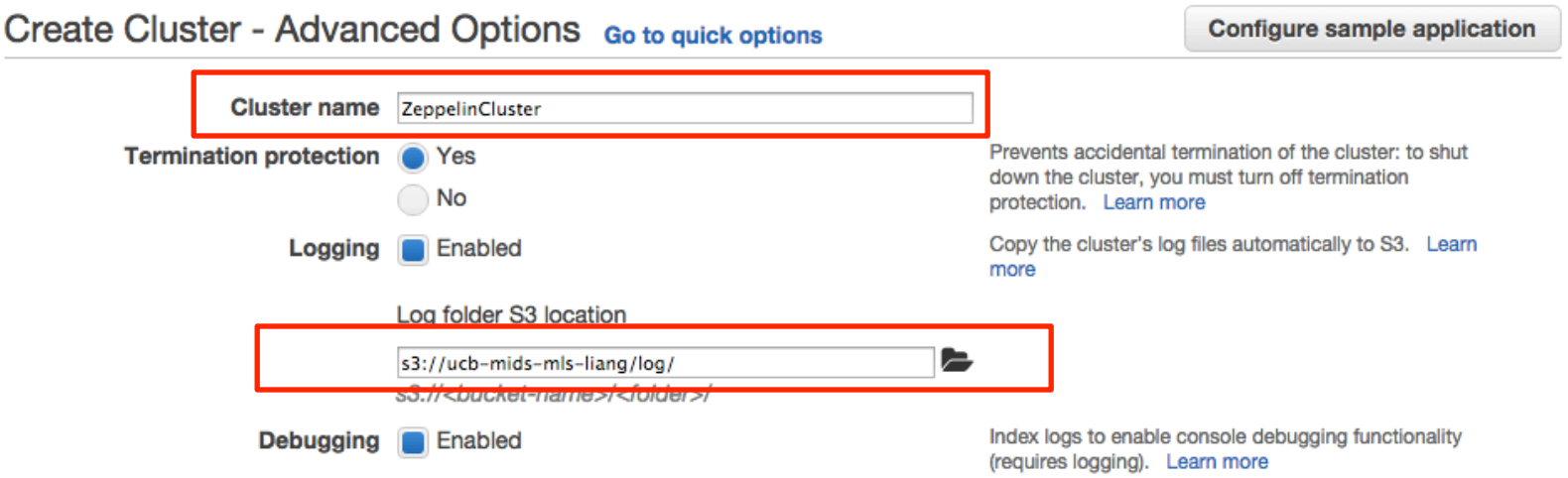
Create EMR cluster

Click Go to advanced options



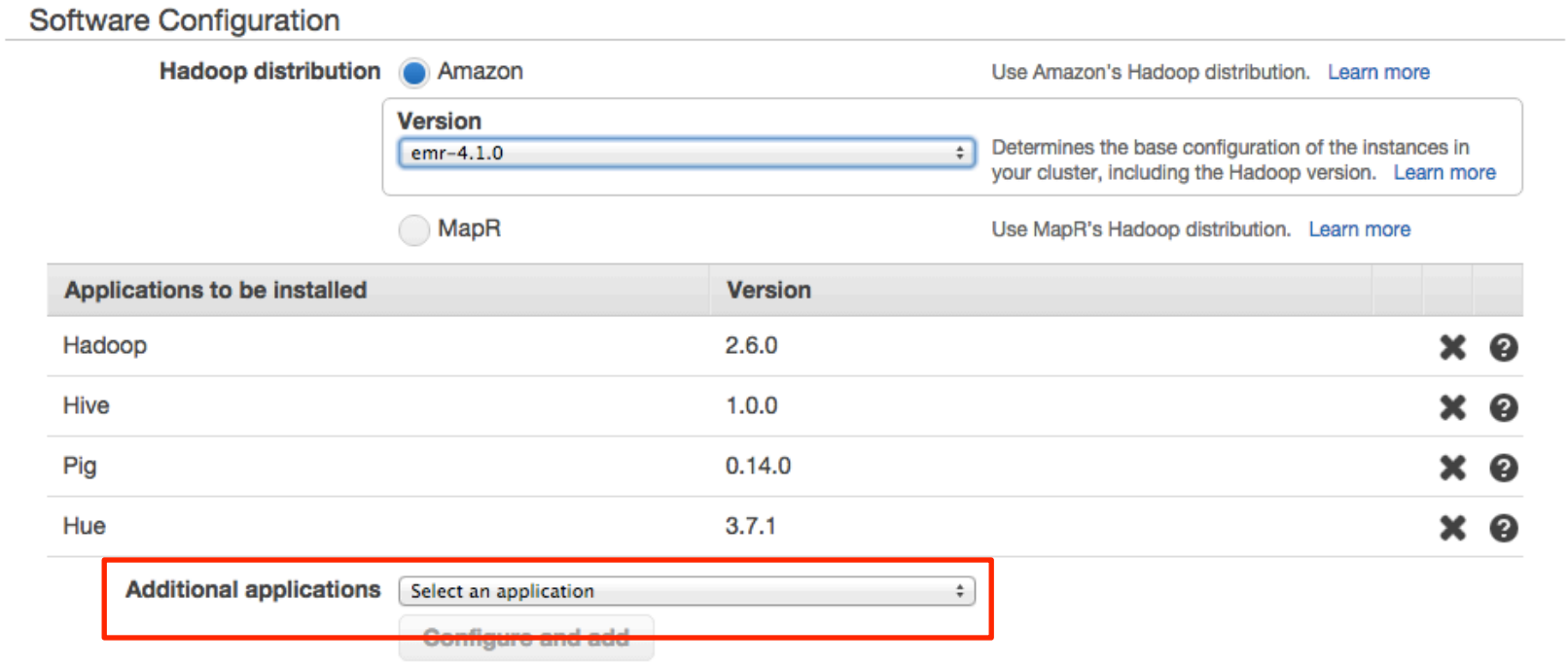
Create EMR cluster

Fill in cluster name and your log folder



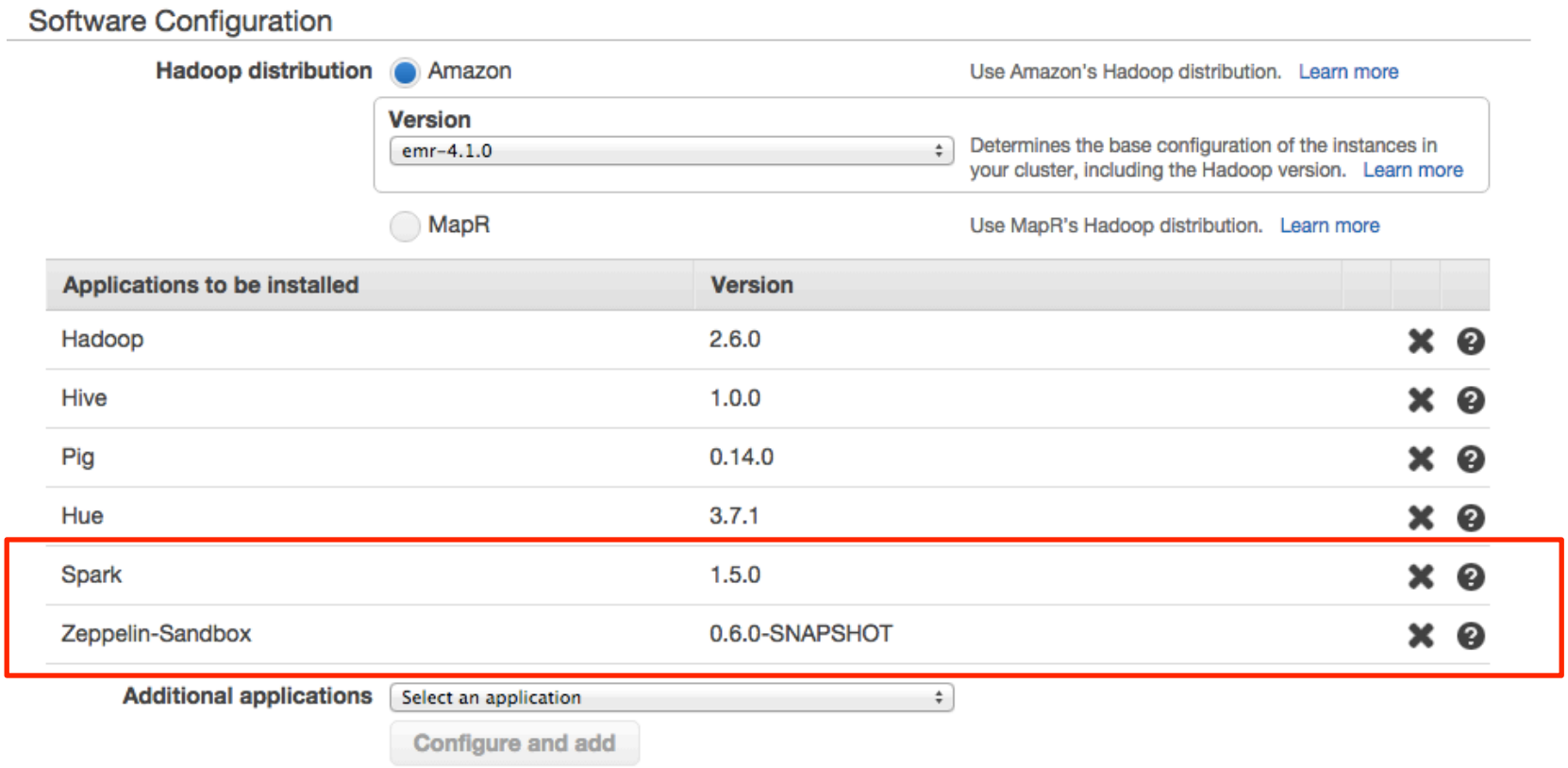
Create EMR cluster

- Choose the new emr version
- By default, hadoop, hive, pig and hue are installed
- Add additional applications



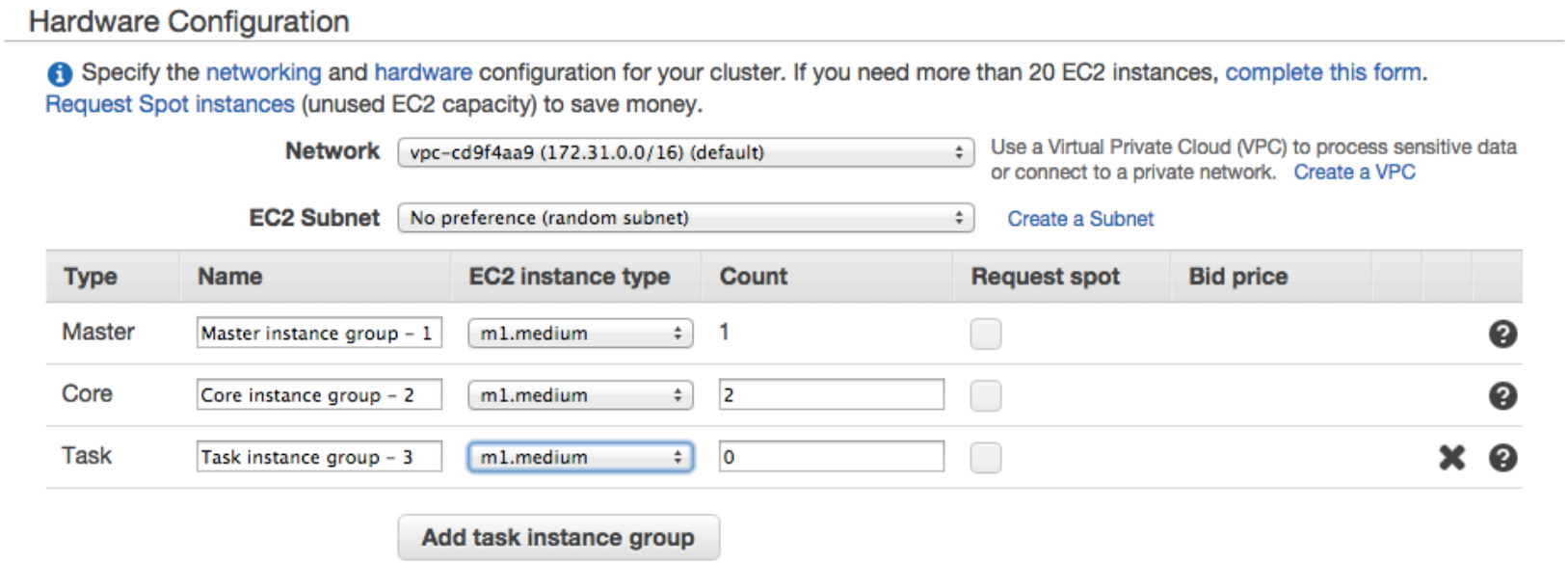
Create EMR cluster

Add Spark and Zeppelin



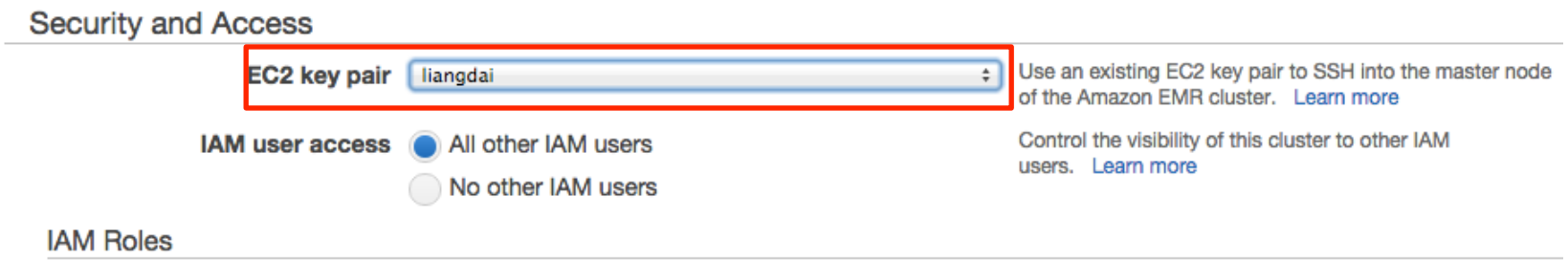
Create EMR cluster

Configure hardware

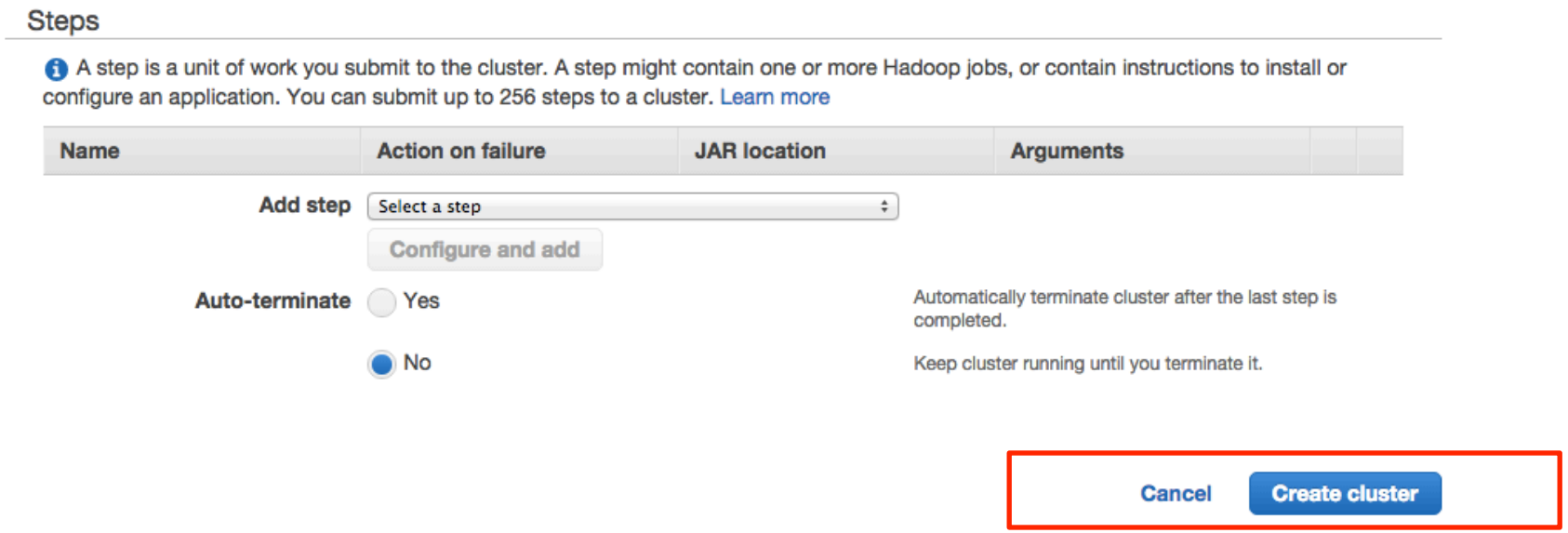


Create EMR cluster

Choose your own key pair



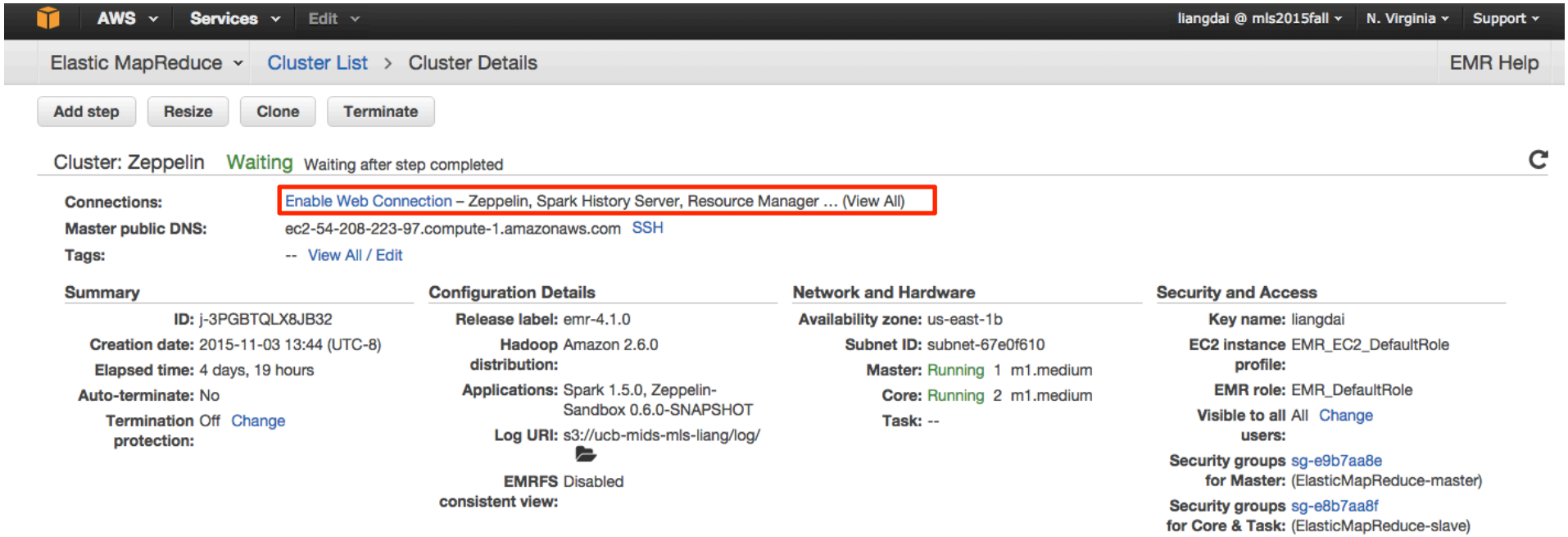
Create EMR cluster



Change the permission of pem file

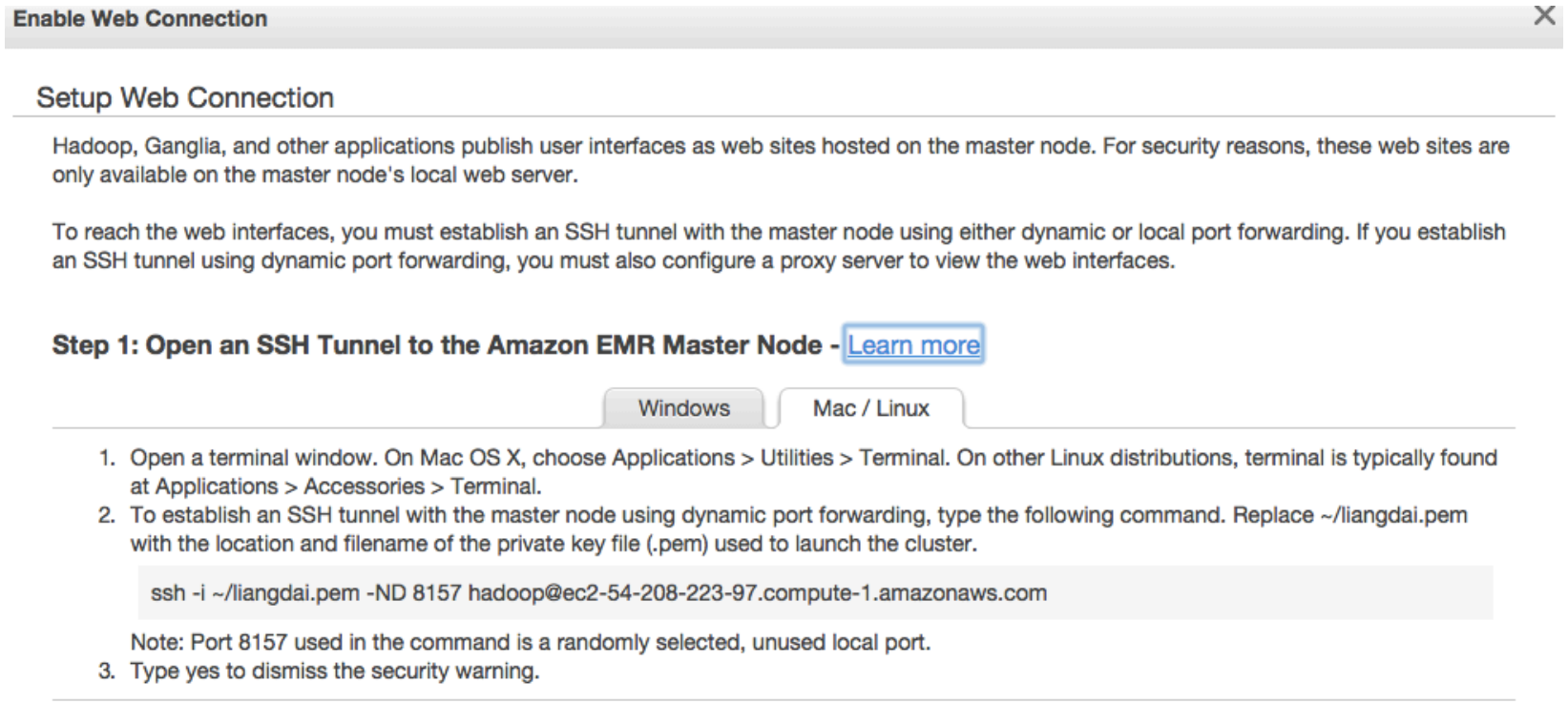
- **Chmod 400 path/to/.pemfile**

Connect to master node



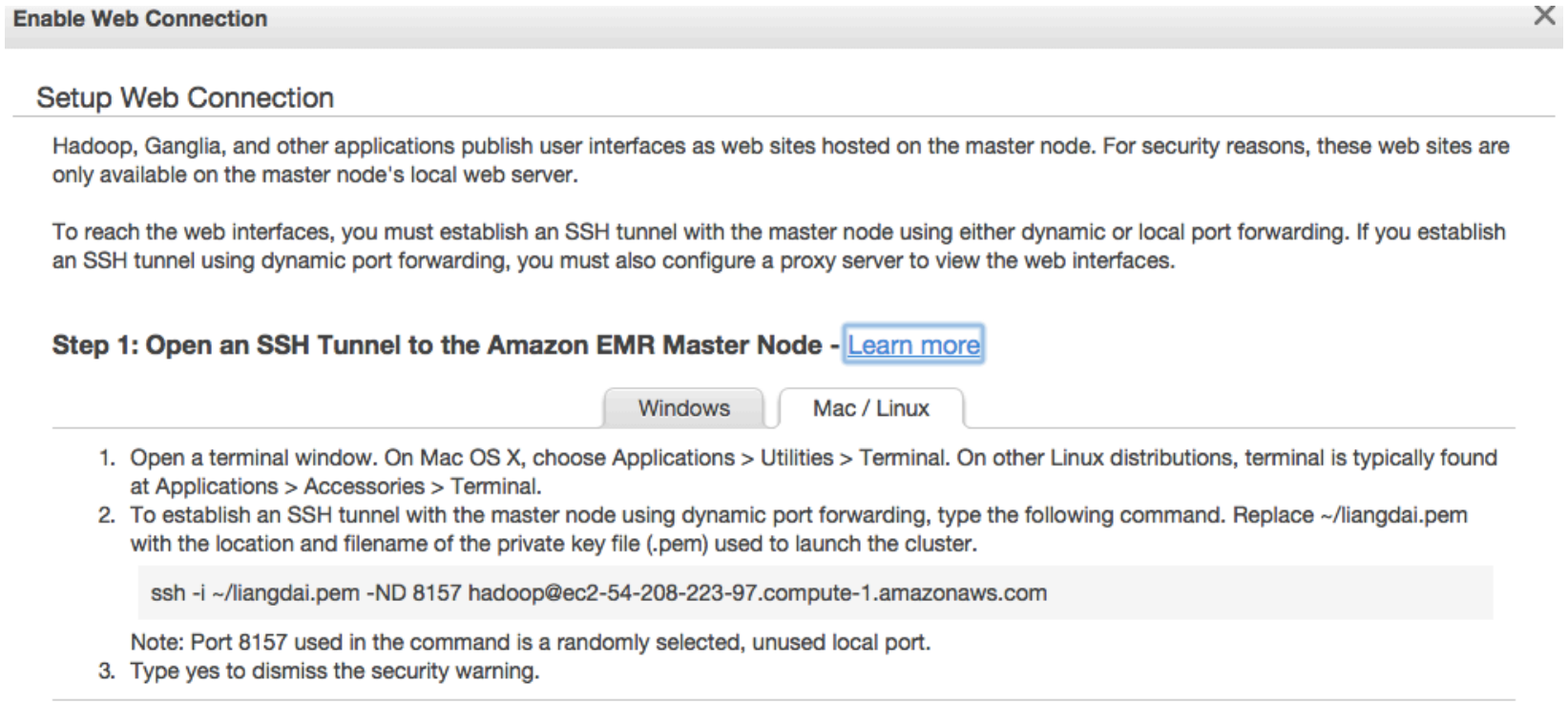
Follow the instructions (Step 1)

Mac/Linux



Follow the instructions (Step1)

Windows



Follow the instructions (Step 2)

Step 2: Configure a proxy management tool - Learn more

Chrome | Firefox

1. Download and install the standard version of FoxyProxy from:
   http://foxyproxy.mozdev.org/downloads.html
2. Restart Chrome after installing FoxyProxy.
3. Using a text editor create a file named foxyproxy-settings.xml containing the following:

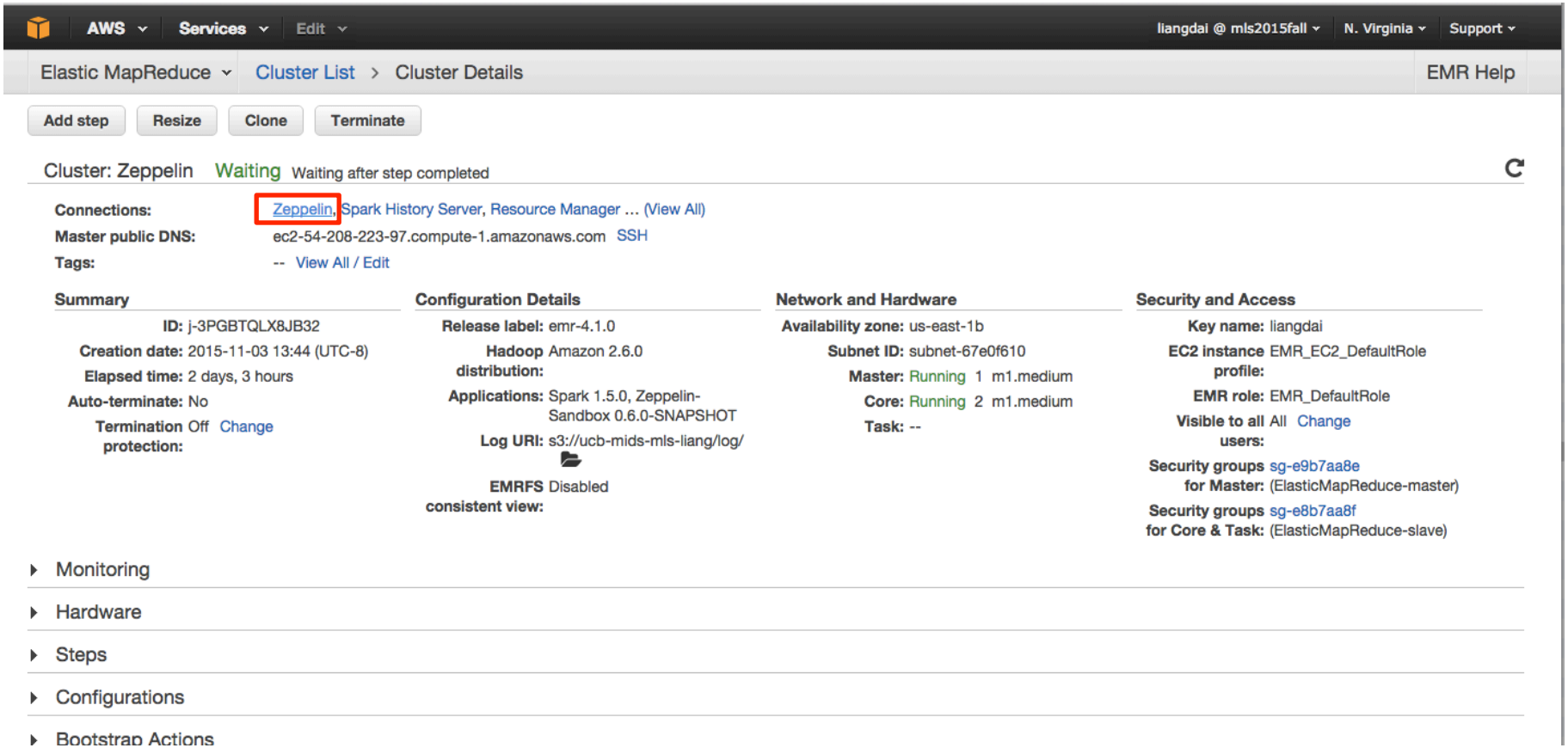
```
<?xml version="1.0" encoding="UTF-8"?>
<foxyproxy>
    <proxies>
        <proxy name="emr-socks-proxy" id="2322596116" notes="" fromSubscription="false" enabled="true" mode="manual"
selectedTabIndex="2" lastresort="false" animatedIcons="true" includeInCycle="true" color="#0055E5" proxyDNS="true"
noInternalIPs="false" autoconfMode="pac" clearCacheBeforeUse="false" disableCache="false"
clearCookiesBeforeUse="false" rejectCookies="false">
            <matches>
                <match enabled="true" name="*ec2*.amazonaws.com*" pattern="*ec2*.amazonaws.com*" isRegEx="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
                <match enabled="true" name="*ec2*.compute*" pattern="*ec2*.compute*" isRegEx="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
                <match enabled="true" name="10.*" pattern="http://10.*" isRegEx="false" isBlackList="false"
isMultiLine="false" caseSensitive="false" fromSubscription="false" />
            </matches>
            <manualconf host="localhost" port="8157" socksversion="5" isSocks="true" username="" password=""
domain="" />
        </proxy>
```

Notes:

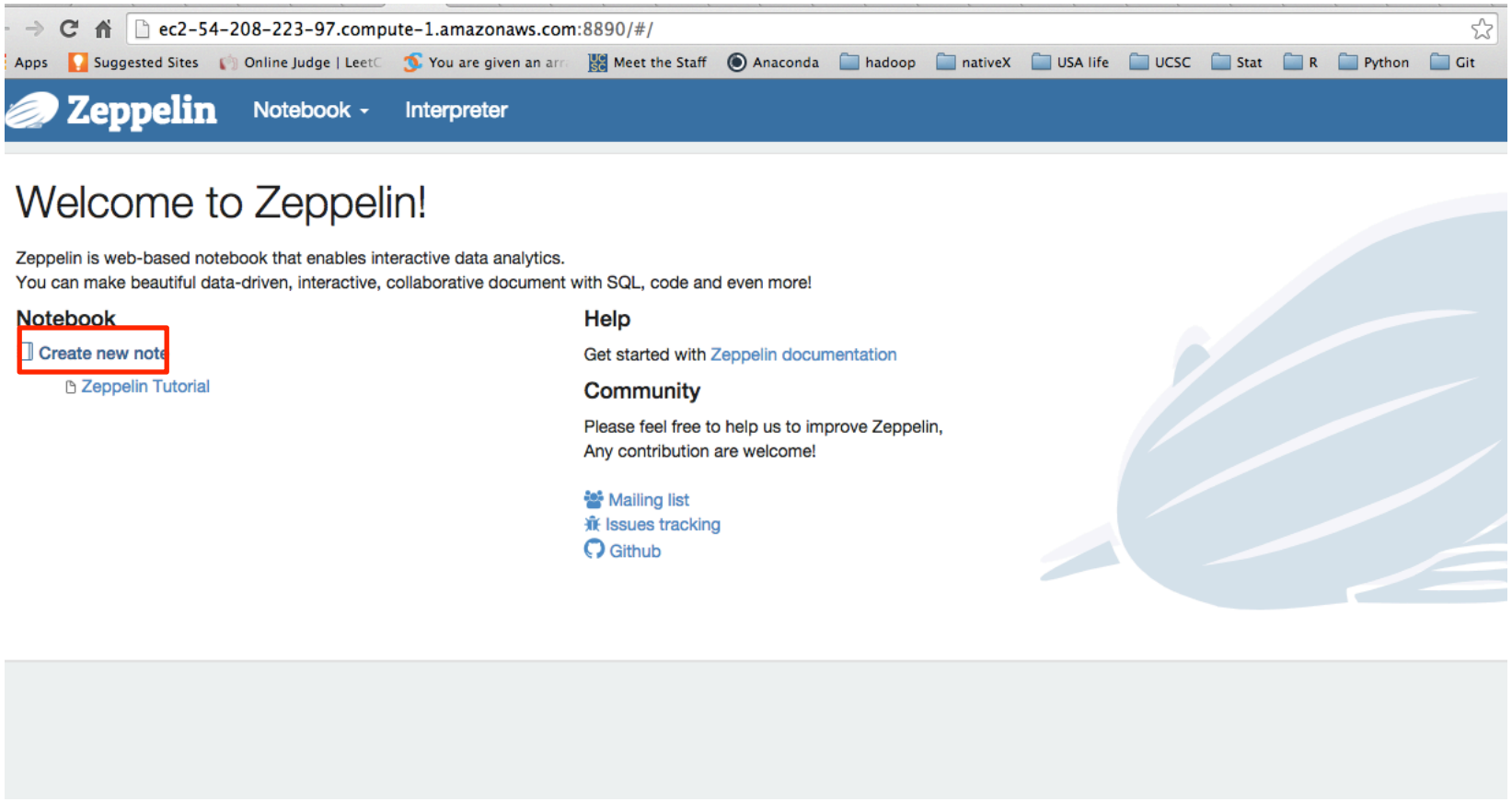
- Port 8157 is the local port number used to establish the SSH tunnel with the master node. This must match the port number you used in PuTTY or terminal.
- The *ec2*.amazonaws.com* pattern matches the public DNS name of clusters in the us-east-1 region.
- The *ec2*.compute* pattern matches the public DNS name of clusters in all other regions.
- The 10.*pattern provides access to the JobTracker log files in Hadoop 1.x. Alter this filter if it conflicts with your network access plan.

4. Click on the FoxyProxy icon in the toolbar and select Options.
5. Click Import/Export.
6. Click Choose File, select foxyproxy-settings.xml, and click Open.
7. In the Import FoxyProxy Settings dialog, click Add.
8. At the top of the page, for Proxy mode, choose Use proxies based on their pre-defined patterns and priorities

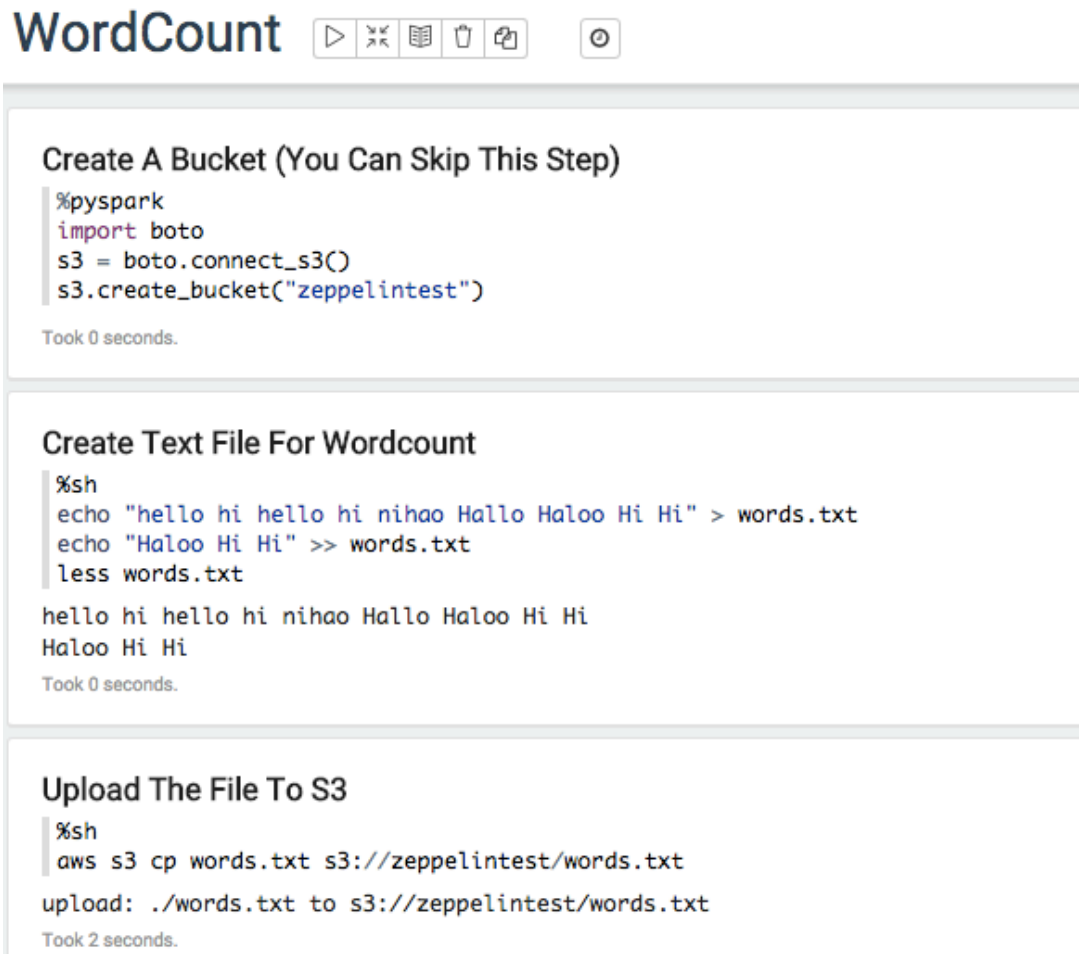
Click Zeppelin



Create notebooks



Prepare a file



Word Count: Python

```
%pyspark
textFile = sc.textFile("s3n://zeppelintest/words.txt")
wordCounts=textFile.flatMap(lambda line: line.split(" ")).map(lambda word: (word,1)).reduceByKey(lambda x,y:x+y )
print(wordCounts.collect())
```

```
[(u'nihao', 1), (u'Haloo', 2), (u'Hallo', 1), (u'hi', 2), (u'Hi', 4), (u'hello', 2)]
```

Took 4 seconds

Word Count: Scala

Scala WordCount

```
val textFile = sc.textFile("s3n://zeppelintest/words.txt")
val wordCounts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)
wordCounts.collect()
```

```
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[199] at textFile at <console>:23
wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[202] at reduceByKey at <console>:25
res12: Array[(String, Int)] = Array((Hallo,1), (hello,2), (hi,2), (Haloo,2), (Hi,4), (nihao,1))
```

Took 3 seconds.

Advantages

- simplicity : for beginner or the marketer in the company, it's easier for him to manipulate the data. In particular thanks to queries in SparkSQL and a nice display widget
- language-agnostic, with a real plugin architecture, named "interpreters". The "cluster" function of iPython or SparkNotebook is quite difficult to understand and customize. Scala and Python are the first 2 main languages available.
- Compared with Jupyter, Zeppelin is focusing on providing analytical environment on top of distributed eco-system.

Other Notebooks

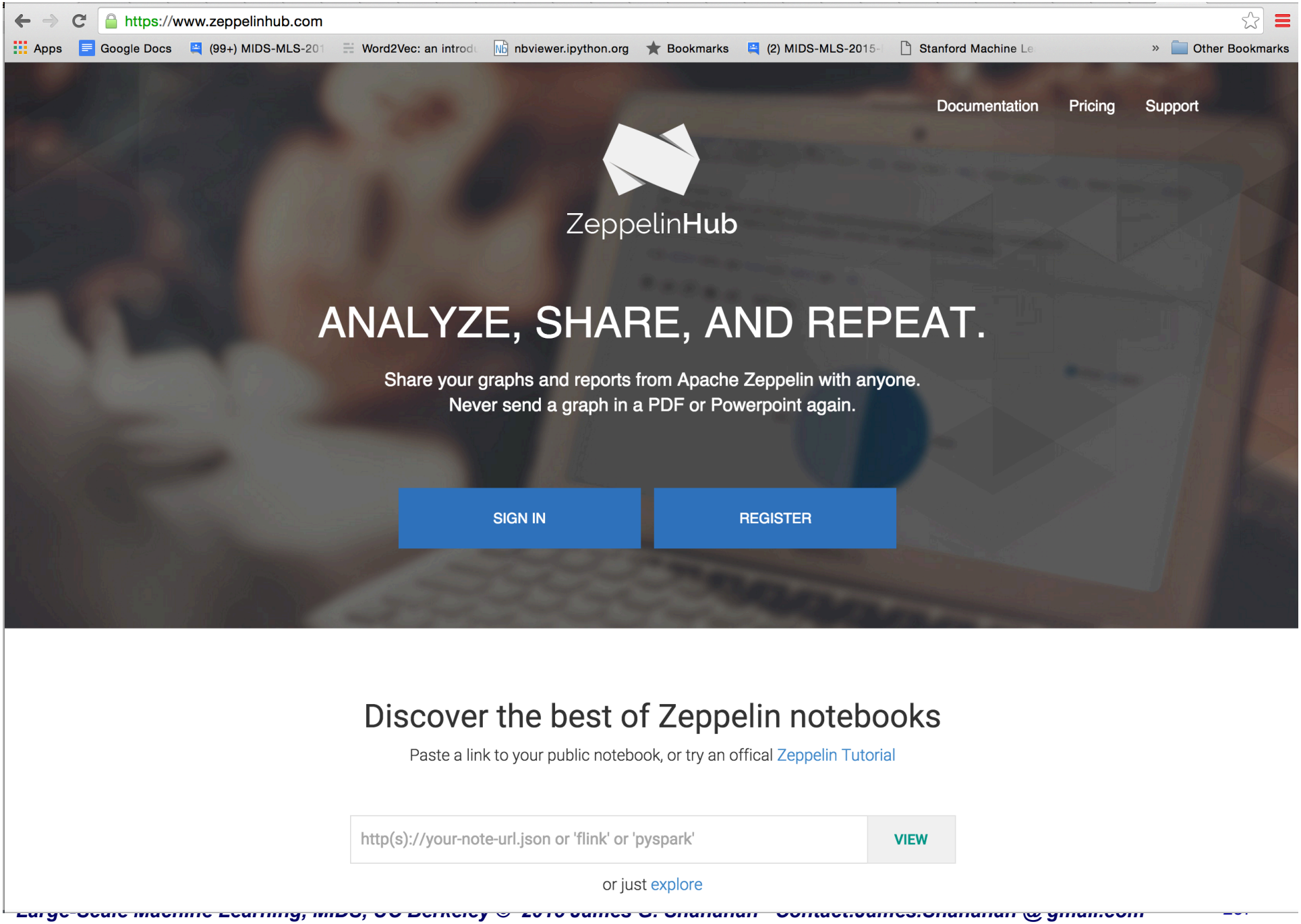
- **Spark Notebook**
  - https://github.com/andypetrella/spark-notebook
- **Databricks Cloud (not free)**
  - https://databricks.com/product/databricks
- **Beaker (not free)**
  - http://beakernotebook.com/

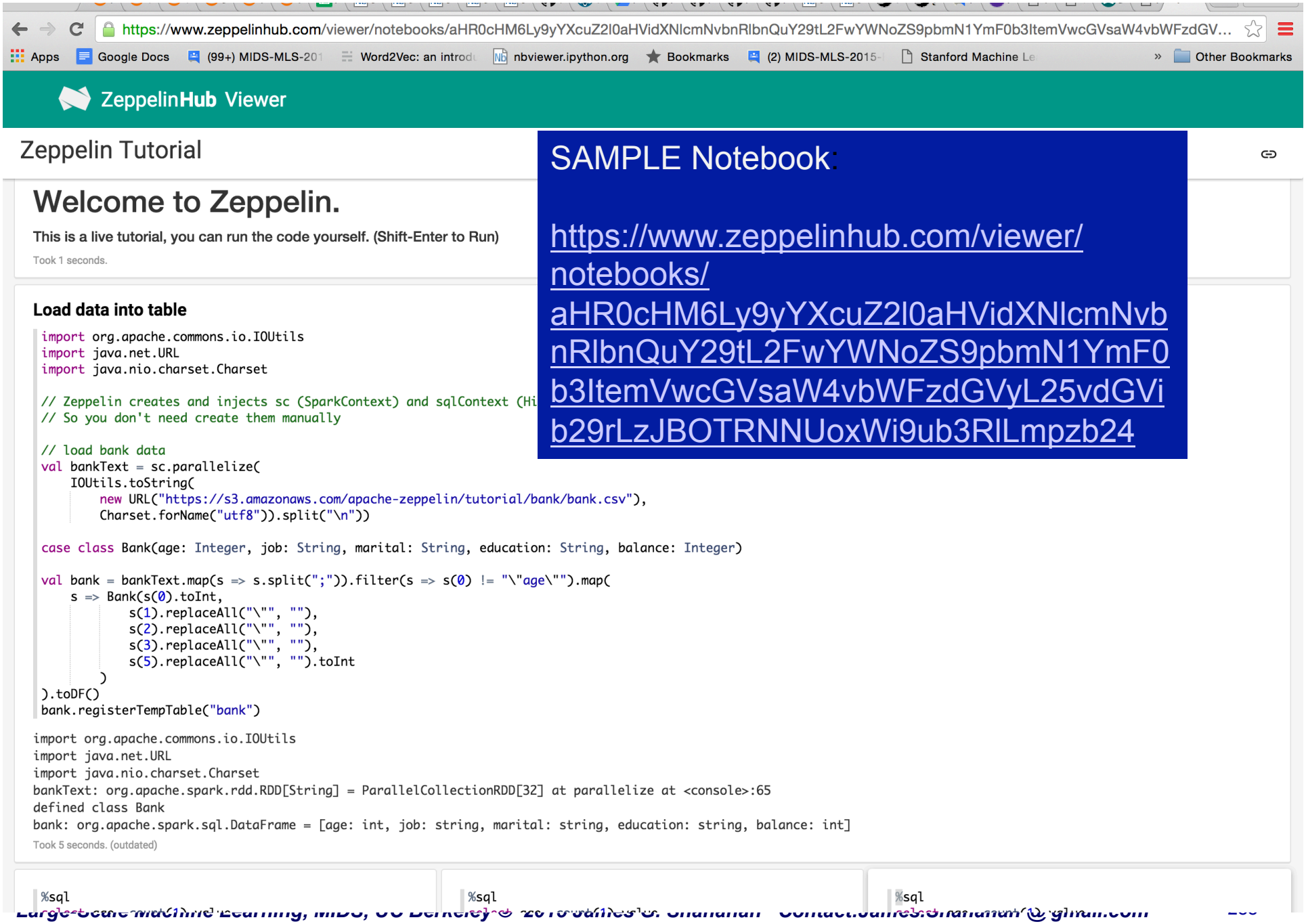
HW11

- **Please complete HW11 in Zeppelin Notebooks**

Zeppelin viewer

- **Use Zeppelin viewer to share your notebooks**
  - https://www.zeppelinhub.com/viewer
- **Here is a sample working notebook viewer link :**
  - https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2l0aHVidXNlcmNvbnRlbnQuY29tL21lYWJoaXNoZWtrdW1hci96ZXBwZWxpbl9ub3RlYm9va3MvbWFzdGVyLzJCSFZDRzJBRS9ub3RlLmpzb24


https://www.zeppelinhub.com
Apps
Google Docs
(99+) MIDS-MLS-201
Word2Vec: an introdu
nbviewer.ipython.org
Bookmarks
(2) MIDS-MLS-2015-
Stanford Machine Le
Other Bookmarks
Documentation
Pricing
Support
ZeppelinHub
ANALYZE, SHARE, AND REPEAT.
Share your graphs and reports from Apache Zeppelin with anyone.
Never send a graph in a PDF or Powerpoint again.
SIGN IN
REGISTER
Discover the best of Zeppelin notebooks
Paste a link to your public notebook, or try an offical Zeppelin Tutorial
http(s)://your-note-url.json or 'flink' or 'pyspark'
VIEW
or just explore


https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2l0aHVidXNlcmNvbnRlbnQuY29tL2FwYWNoZS9pbmN1YmF0b3ItemVwcGVsaW4vbWFzdGV...
Apps
Google Docs
(99+) MIDS-MLS-201
Word2Vec: an introd
nbviewer.ipython.org
Bookmarks
(2) MIDS-MLS-2015-
Stanford Machine Le
Other Bookmarks
ZeppelinHub Viewer
Zeppelin Tutorial
SAMPLE Notebook:
https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2l0aHVidXNlcmNvbnRlbnQuY29tL2FwYWNoZS9pbmN1YmF0b3ItemVwcGVsaW4vbWFzdGVyL25vdGVib29rLzJBOTRNNUoxWi9ub3RlLmpzb24
Welcome to Zeppelin.
This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)
Took 1 seconds.
Load data into table
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
// Zeppelin creates and injects sc (SparkContext) and sqlContext (Hi
// So you don't need create them manually
// load bank data
val bankText = sc.parallelize(
IOUtils.toString(
new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv"),
Charset.forName("utf8")).split("\n"))
case class Bank(age: Integer, job: String, marital: String, education: String, balance: Integer)
val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
s => Bank(s(0).toInt,
s(1).replaceAll("\"", ""),
s(2).replaceAll("\"", ""),
s(3).replaceAll("\"", ""),
s(5).replaceAll("\"", "").toInt
)
).toDF()
bank.registerTempTable("bank")
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[32] at parallelize at <console>:65
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, balance: int]
Took 5 seconds. (outdated)
%sql
%sql
%sql

Zeppelin Tutorial

SAMPLE Notebook:

https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2l0aHVidXNlcmNvbnRlbnQuY29tL2FwYWNoZS9pbmN1YmF0b3ItemVwcGVsaW4vbWFzdGVyL25vdGVib29rLzJBOTRNNUoxWi9ub3RlLmpzb24

Load data into table

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or S
// So you don't need create them manually

// load bank data
val bankText = sc.parallelize(
    IOUtils.toString(
        new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv")
        Charset.forName("utf8")).split("\n"))

case class Bank(age: Integer, job: String, marital: String, education: String, bal

val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
    s => Bank(s(0).toInt,
            s(1).replaceAll("\"", ""),
            s(2).replaceAll("\"", ""),
            s(3).replaceAll("\"", ""),
            s(5).replaceAll("\"", "").toInt
        )
).toDF()
bank.registerTempTable("bank")
```

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[32] at parallelize at <console>:65
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, balance: int]
Took 5 seconds. (outdated)
```

```
%sql
select age, count(1) value
from bank
where age < 30
group by age
order by age
```

settings

```
%sql
select age, count(1) value
from bank
where age < ${maxAge=30}
group by age
order by age
```

maxAge

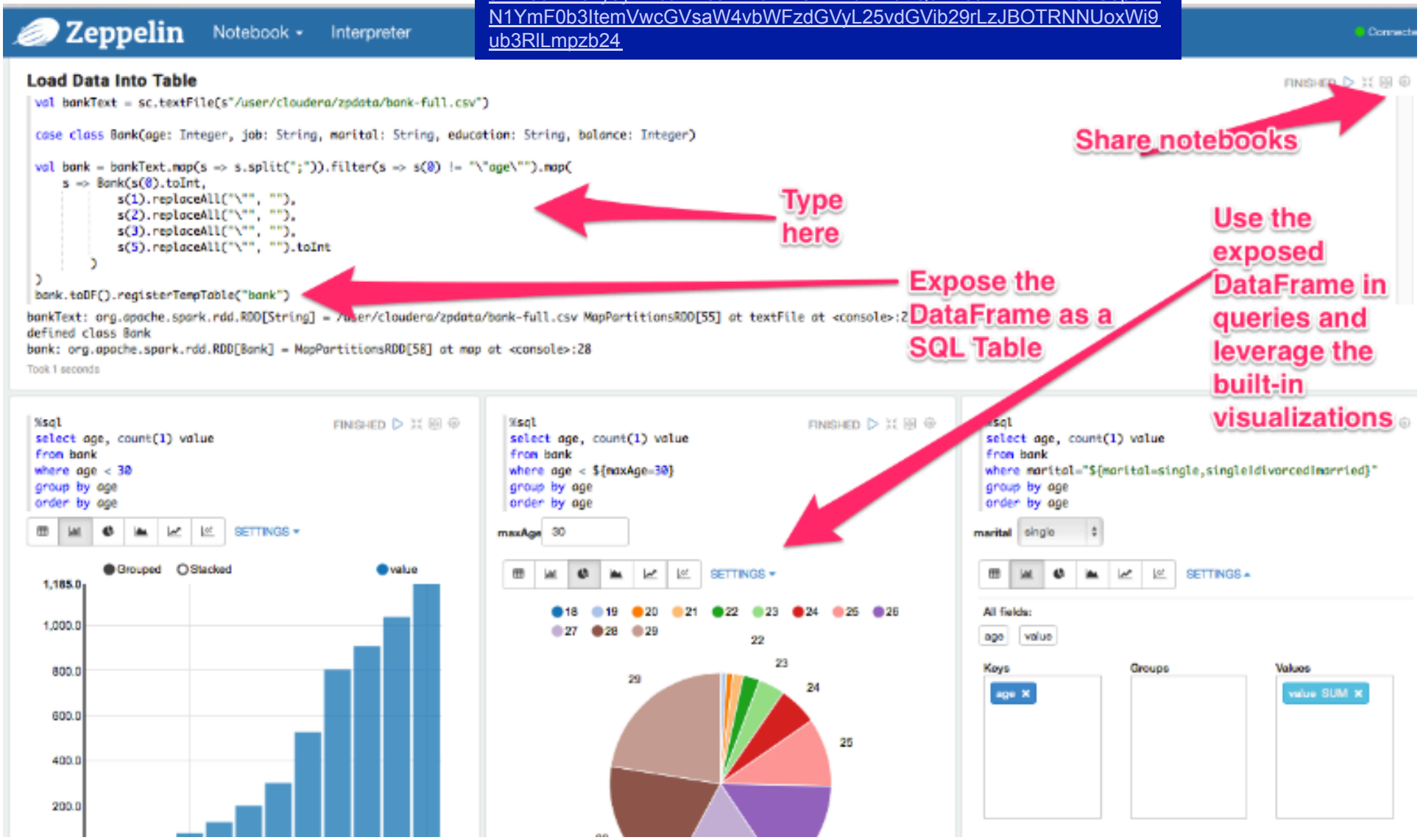
35

```
%sql
select age, count(1) value
from bank
where marital="${marital=single,single|divorced|married}"
group by age
order by age
```

marital

SAMPLE Notebook

https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2I0aHVidXNlcmNvbnRlbnQuY29tL2FwYWNoZS9pbmN1YmF0b3ItemVwcGVsaW4vbWFzdGVyL25vdGVib29rLzJBOTRNNUoxWi9ub3RlLmpzb24



https://www.zeppelinhub.com/viewer/notebooks/aHR0cHM6Ly9yYXcuZ2l0aHVidXNlcmNvbnRlbnQuY29tL21lYWJoaXNoZWtrdW1hci96ZXBwZWxpbl9ub3RlYm9va3Mvb

Apps Google Docs (99+) MIDS-MLS-201 Word2Vec: an introd nbviewer.ipython.org Bookmarks (2) MIDS-MLS-2015- Stanford Machine Le Other Bookmarks

(99+) MIDS-MLS-2016-Spring - Google Groups
https://groups.google.com/forum/#!forum/mids-mls-2016-spring

ZeppelinH

Equations In Zeppelin

When $a \neq 0$, there are two solutions to $ax^2 + bx + c = 0$ and they are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Took 0 seconds. (outdated)

Zeppelin Viewer is a community site for sharing Zeppelin notebooks.

Your use of and access to this site is subject to the terms of use.
Apache Zeppelin (incubating) is a trademark of the Apache Software Foundation.
This site is maintained as a community service by NFLabs.

Optimization in Spark

- http://spark.apache.org/docs/latest/mllib-optimization.html#Choosing-an-Optimization-Method
  - Stochastic Gradient Descent (Minibatch size = minibatchFraction)
  - L-BFGS

The SGD implementation in GradientDescent uses a simple (distributed) sampling of the data examples. We recall that the loss part of the optimization problem (1) is $\frac{1}{n}\sum_{i=1}^{n} L(\mathbf{w};\mathbf{x}_i,y_i)$, and therefore $\frac{1}{n}\sum_{i=1}^{n} L'_{\mathbf{w},i}$ would be the true (sub)gradient. Since this would require access to the full data set, the parameter `miniBatchFraction` specifies which fraction of the full data to use instead. The average of the gradients over this subset, i.e.

$$\frac{1}{|S|}\sum_{i\in S} L'_{\mathbf{w},i},$$

is a stochastic gradient. Here S is the sampled subset of size $|S| =$ `miniBatchFraction` $\cdot n$.

In each iteration, the sampling over the distributed dataset (RDD), as well as the computation of the sum of the partial results from each worker machine is performed by the standard spark routines.

If the fraction of points `miniBatchFraction` is set to 1 (default), then the resulting step in each iteration is exact (sub)gradient descent. In this case there is no randomness and no variance in the used step directions. On the other extreme, if `miniBatchFraction` is chosen very small, such that only a single point is sampled, i.e. $|S| =$ `miniBatchFraction` $\cdot n = 1$, then the algorithm is equivalent to standard SGD. In that case, the step direction depends from the uniformly random sampling of the point.

Outline

- **Spark re-Review**
- **Loss functions**
  - Loss term: Understanding loss functions (graphically)
  - Perceptron learning graphically speaking
  - Regularization term
- **SVMs**
- **Distributed Gradient descent**
- **Logistic regression, diagnostics, priors as a form of regularization**
- **Zeppelin notebooks backed by a Spark cluster**