DAMLAS Part 2
High Entropy Friday, 7/15

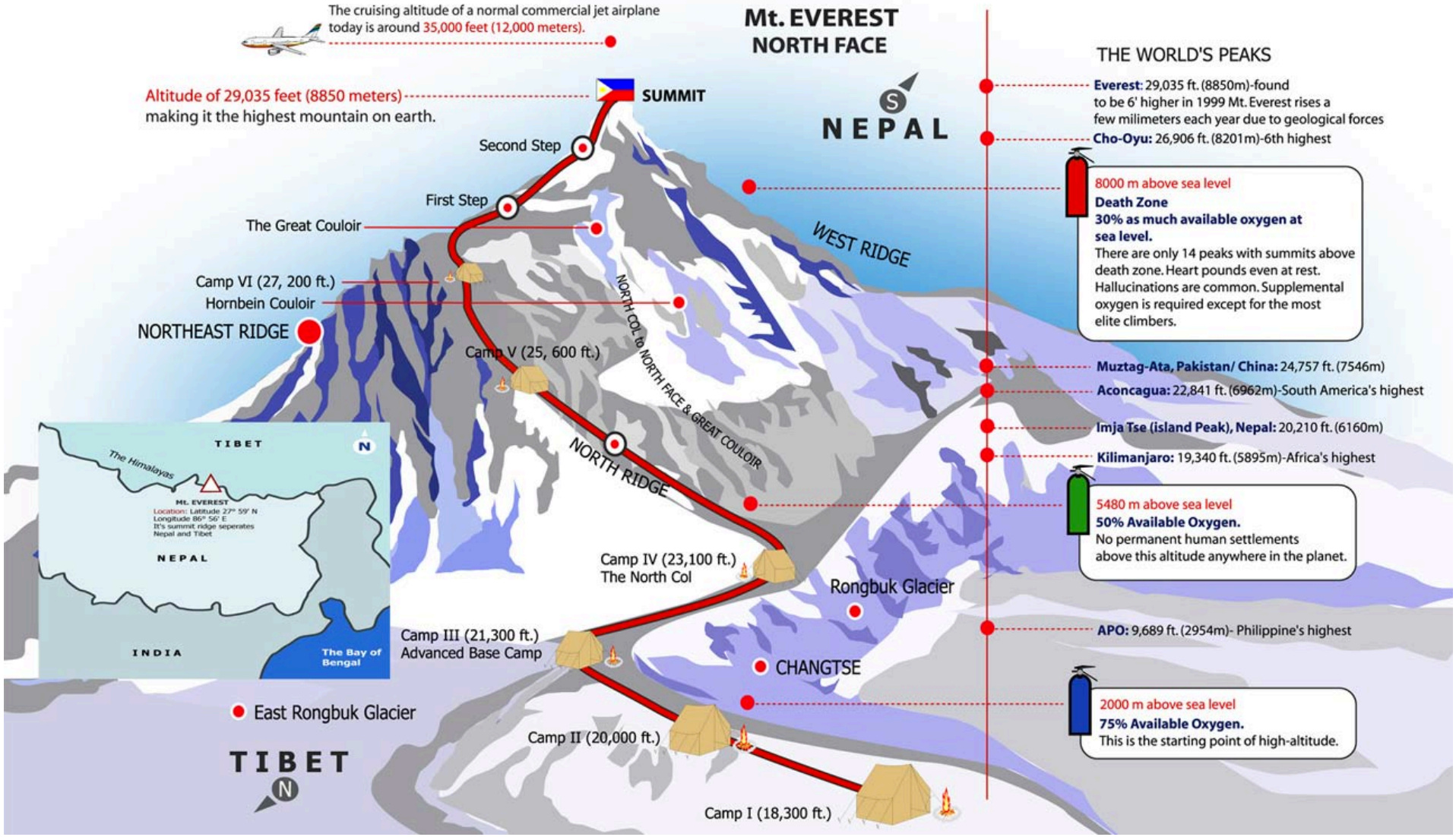
James G. Shanahan [1,2]

[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

(Hilary Step!): Summiting Mt Everest



Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering (HHH)**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

Machine Learning Background

Machine Learning (ML):"*a computer program that improves its performance at some task through experience*" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)
GOAL: Approximate $f(x_1,\ldots,x_n)$->y

Y is categorical

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

Machine Learning: Regression

Machine Learning (ML):"*a computer program that improves its performance at some task through experience*" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)
GOAL: Approximate $f(x_1,\ldots,x_n)$->y

Y is real valued

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					76
…	…	…	…	…	…
L (aka m)	0	4	...	8	97

Machine Learning semi-supervised

Machine Learning (ML):"*a computer program that improves its performance at some task through experience*" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)
GOAL: Approximate $f(x_1,\ldots,x_n)$->y

Y is only partially available

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					~~76~~
…	…	…	…	…	…
L (aka m)	0	4	...	8	97

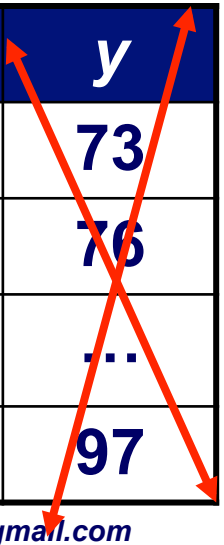
Machine Learning Unsupervised

Machine Learning (ML):"*a computer program that improves its performance at some task through experience*" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)
GOAL: Approximate $f(x_1,\ldots,x_n)$->y

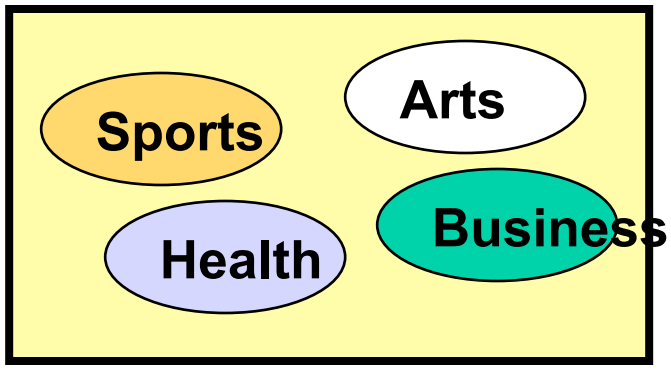
Y is not available

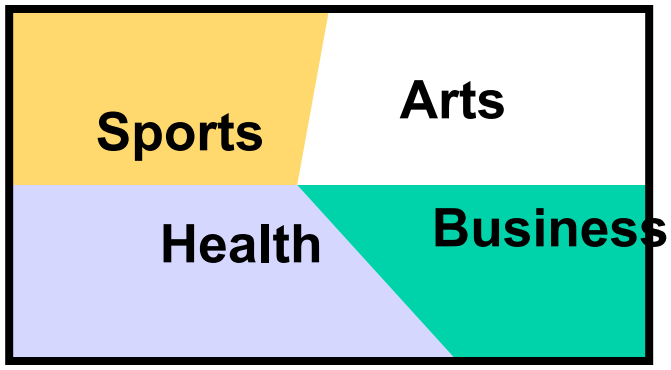
Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					76
…	…	…	…	…	…
L (aka m)	0	4	...	8	97



Families of Supervised Learning

- **Generative Classifier (Bottom-up learning)**
  - Build model of each class
  - Assume the underlying form of the classes and estimate their parameters (e.g., a Gaussian)
- **Discriminative Classifier (Top down)**
  - Build model of boundary between classes
  - Assume the underlying form of the discriminant and estimate its parameters (e.g., a hyperplane)



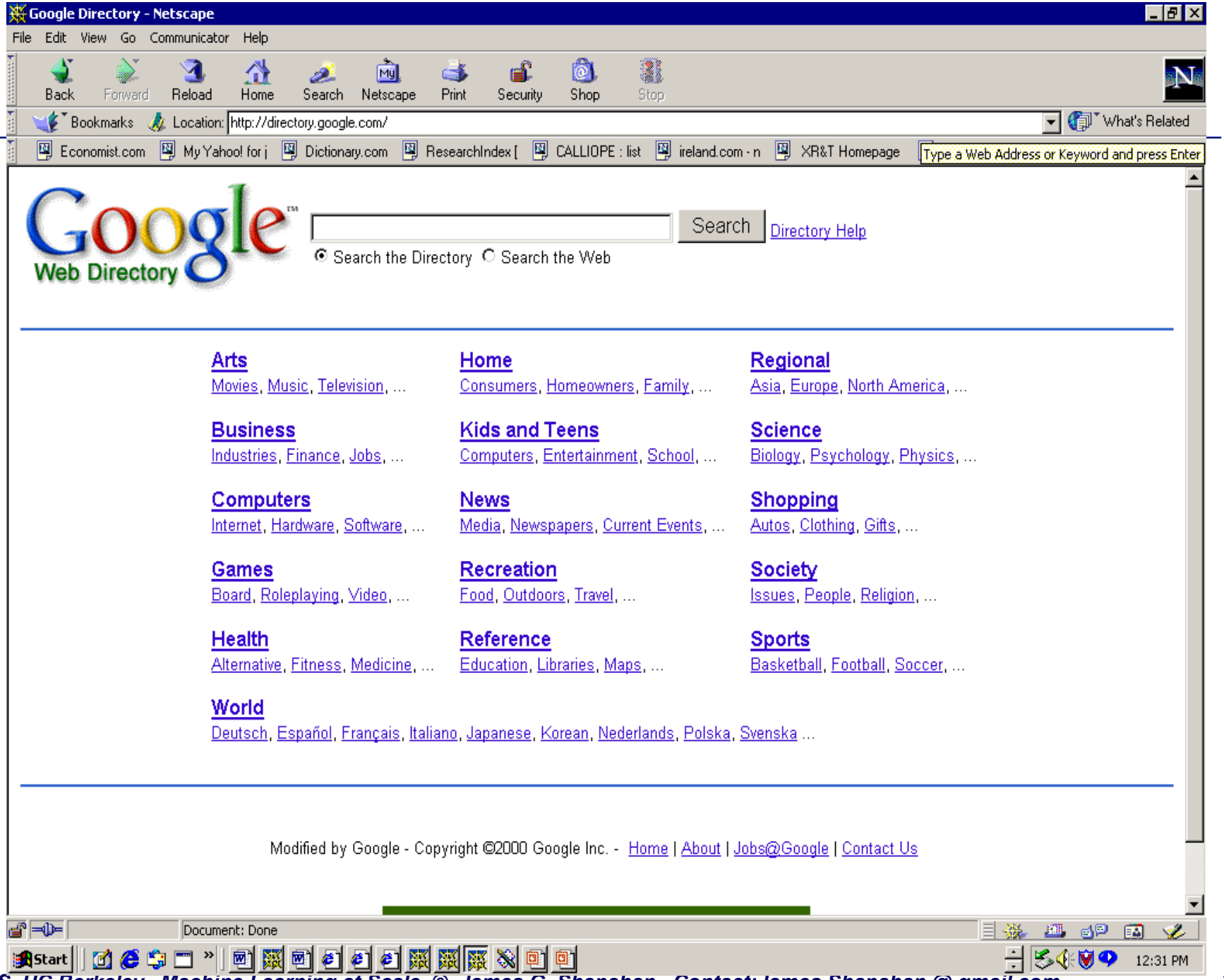


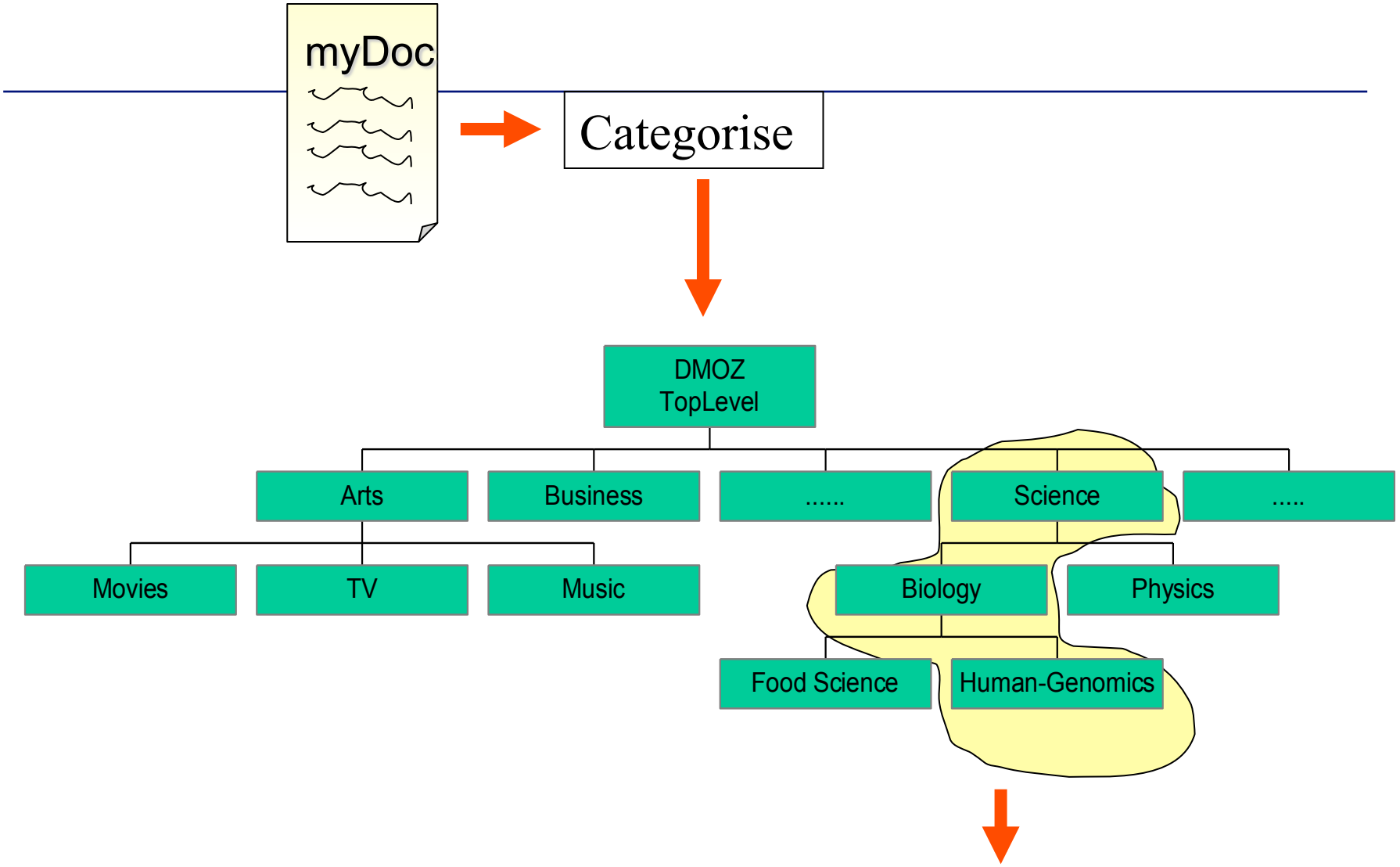
Generative vs. Discriminative

- **Generative learning (e.g., Bayesian Networks, HMM, Naïve Bayes, EM GMM) typically more flexible**
  - More complex problems
  - More flexible predictions
- **Discriminative learning (e.g., ANN, SVM) typically more accurate**
  - Better with small datasets
  - Faster to train

Parametric vs. Non-Parametric ML Algorithms

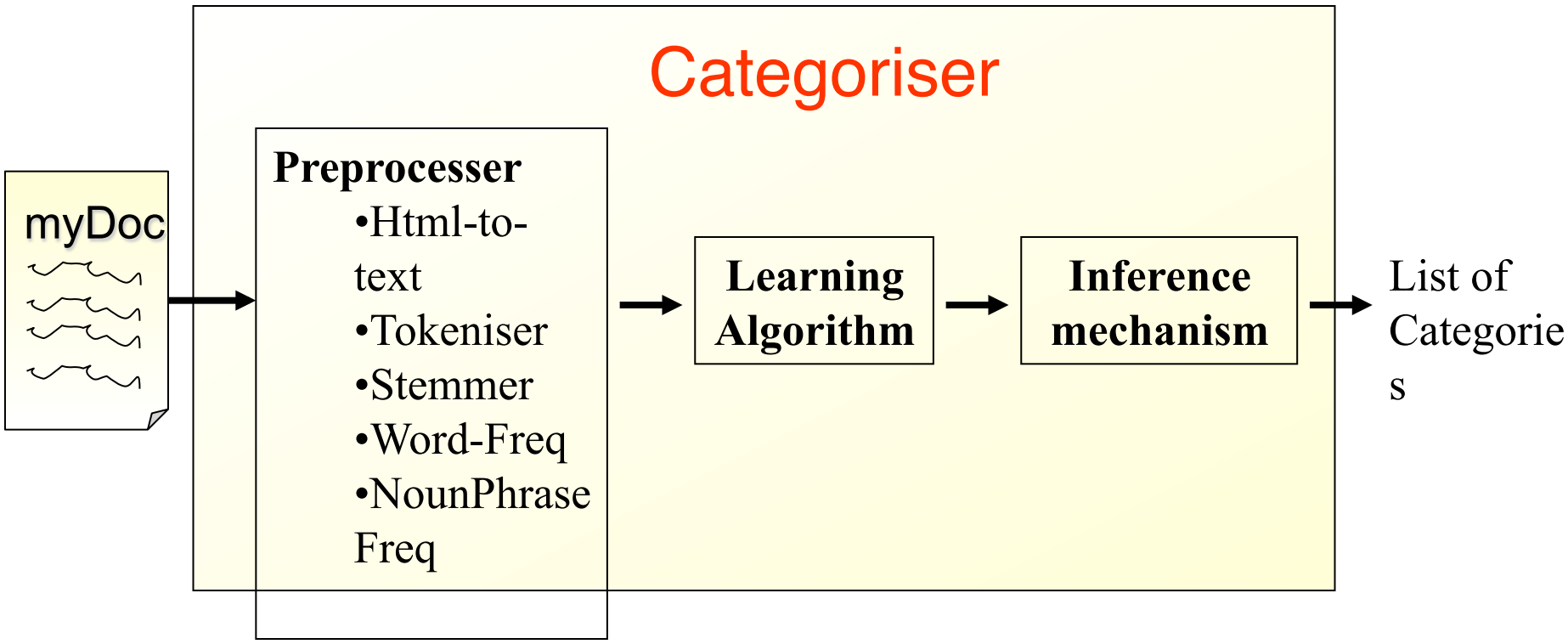
- **Parametric ML Algorithms (e.g., OLS, Decision Trees; SVMs, NNs)**
  - Model-based methods, such as neural networks and the mixture of Gaussians, use the data to build a parameterized model. After training, the model is used for predictions and the data are generally discarded.
- **Non-Parametric (lowess(); knn; some flavours of SVMs)**
  - In contrast, ``memory-based'' methods are non-parametric approaches that explicitly retain the training data, and use it each time a prediction needs to be made.
  - The term “non-parametric” (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis/model grows linearly with the size of the training set.


Google Directory - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://directory.google.com/
What's Related
Economist.com My Yahoo! for j Dictionary.com ResearchIndex [CALLIOPE : list ireland.com - n XR&T Homepage
Type a Web Address or Keyword and press Enter
Google
Web Directory
Search
Directory Help
Search the Directory
Search the Web
Arts
Movies, Music, Television, ...
Home
Consumers, Homeowners, Family, ...
Regional
Asia, Europe, North America, ...
Business
Industries, Finance, Jobs, ...
Kids and Teens
Computers, Entertainment, School, ...
Science
Biology, Psychology, Physics, ...
Computers
Internet, Hardware, Software, ...
News
Media, Newspapers, Current Events, ...
Shopping
Autos, Clothing, Gifts, ...
Games
Board, Roleplaying, Video, ...
Recreation
Food, Outdoors, Travel, ...
Society
Issues, People, Religion, ...
Health
Alternative, Fitness, Medicine, ...
Reference
Education, Libraries, Maps, ...
Sports
Basketball, Football, Soccer, ...
World
Deutsch, Español, Français, Italiano, Japanese, Korean, Nederlands, Polska, Svenska ...
Modified by Google - Copyright ©2000 Google Inc. - Home | About | Jobs@Google | Contact Us
Document: Done
Start
12:31 PM

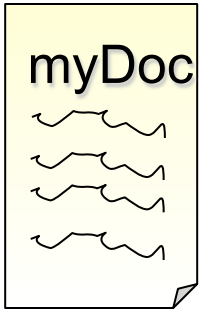


RESULT: Do Local search in Human-Genomics folder

Categoriser Architecture



Step 1: Preprocessing
Transform Document into Bag-of-words



Transformation

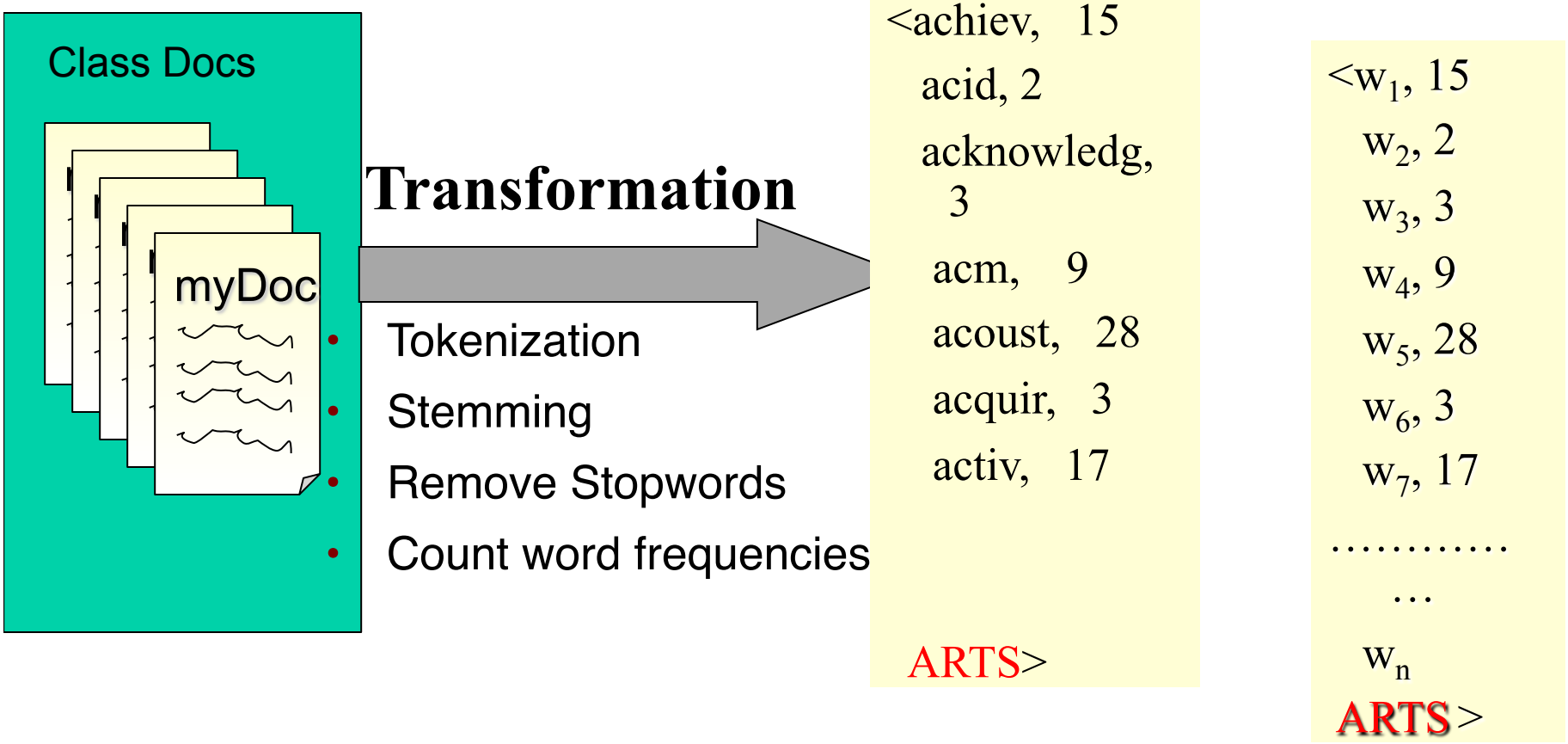
- Tokenization
- Stemming
- Remove Stopwords
- Count word frequencies

<achiev, 15
acid, 2
acknowledg, 3
acm, 9
acoust, 28
acquir, 3
activ, 17

ARTS>

<w_1, 15
w_2, 2
w_3, 3
w_4, 9
w_5, 28
w_6, 3
w_7, 17
.....
...
w_n
ARTS>

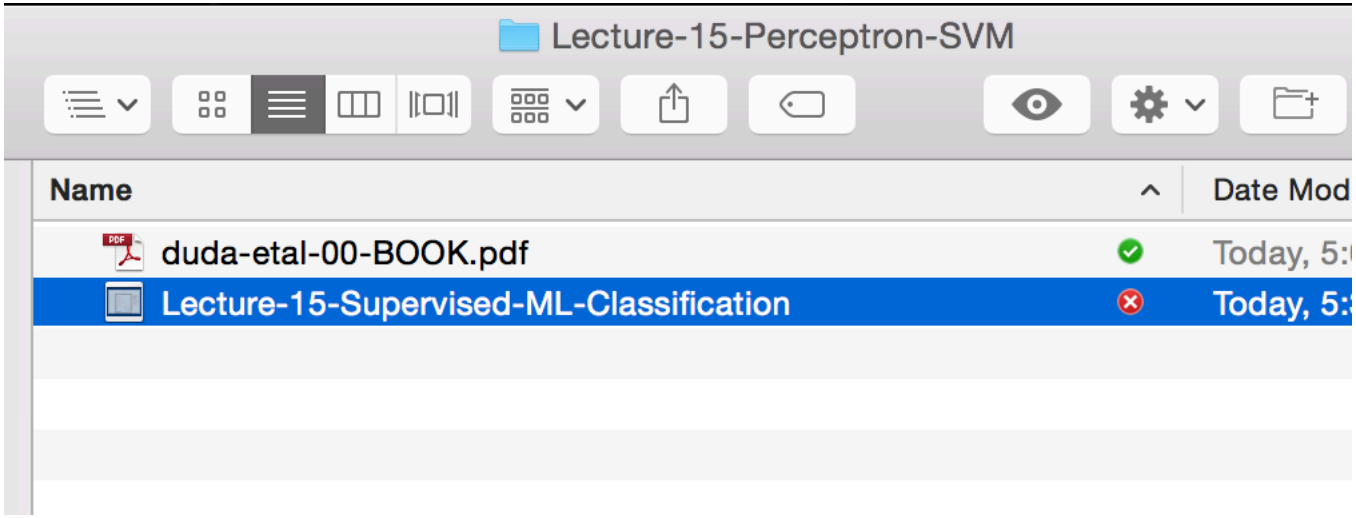
Step 1: Glob all docs for a class



Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering (HHH)**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

Switch slides to



Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering (HHH)**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

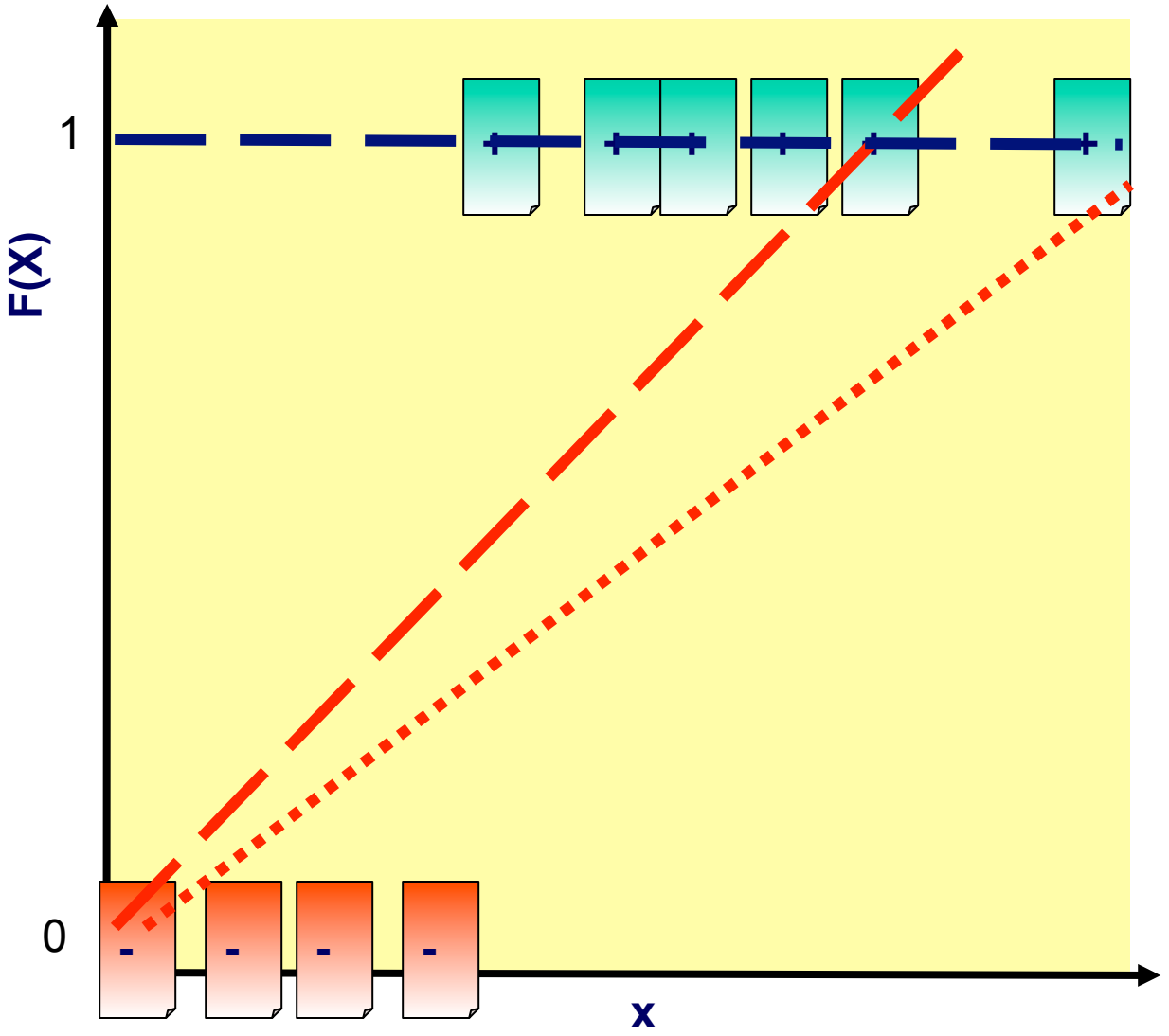
Logistic regression

Model classification as a regression?

A linear regression function is linear in the components of X

E.g., $y=ax_1 + bx_2 + c$

Very volatile; out of range f(x)

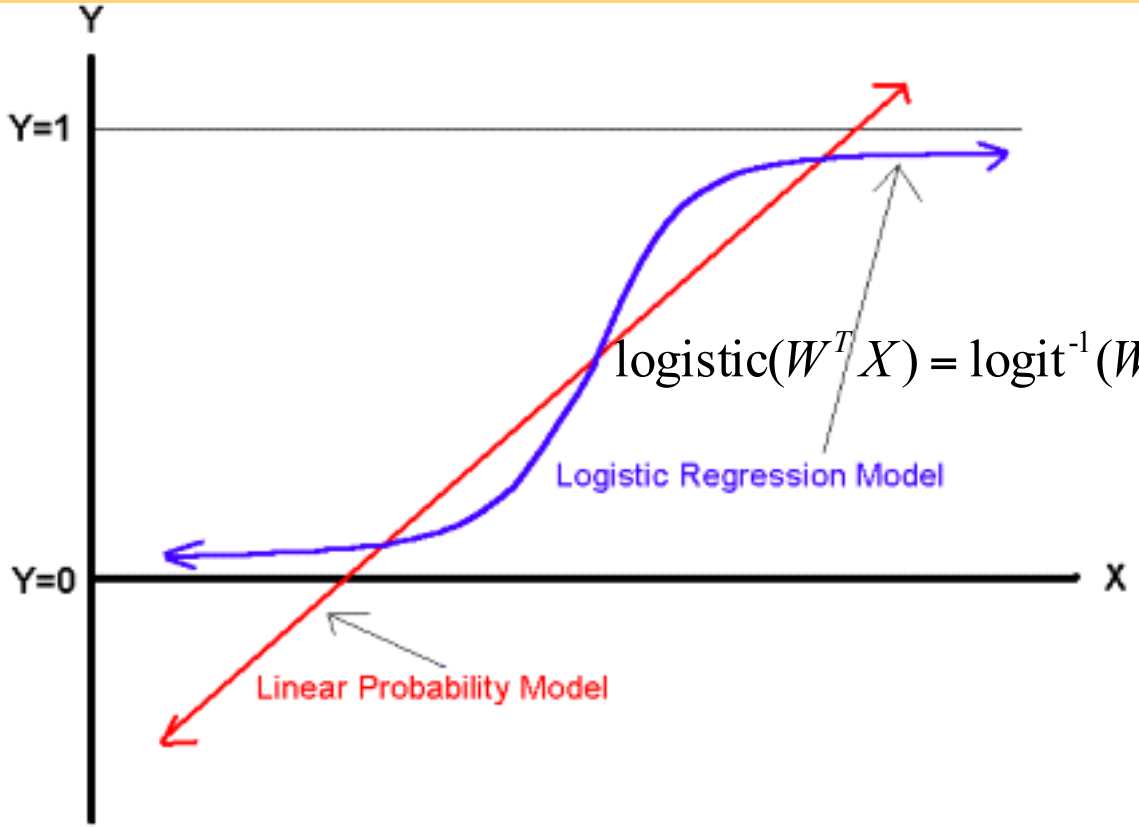


Training Data

E.g.	x_1	x_2	y
1	3	0	-1
2			+1
…	…	…	…
L	0	4	-1

Limit range of f(x) using a logit function

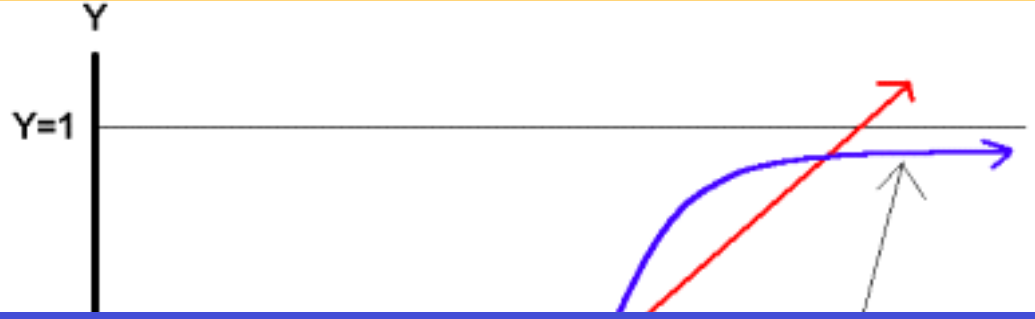
Intuitively it does not make sence to have f(x) >>1 or f(x)<<0
So limit using a sigmoid squashing function….



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

Limit range of f(x) using a logit function

Intuitively it does not make sence to have f(x) >>1 or f(x)<<0
So limit using a sigmoid squashing function….



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

$$\frac{p}{1-p} = \exp(W^T X)$$

$$p = (1-p)\exp(W^T X)$$

$$p = \frac{\exp(W^T X)}{1+\exp(W^T X)} = \frac{1}{1+\exp(-W^T X^i)}$$

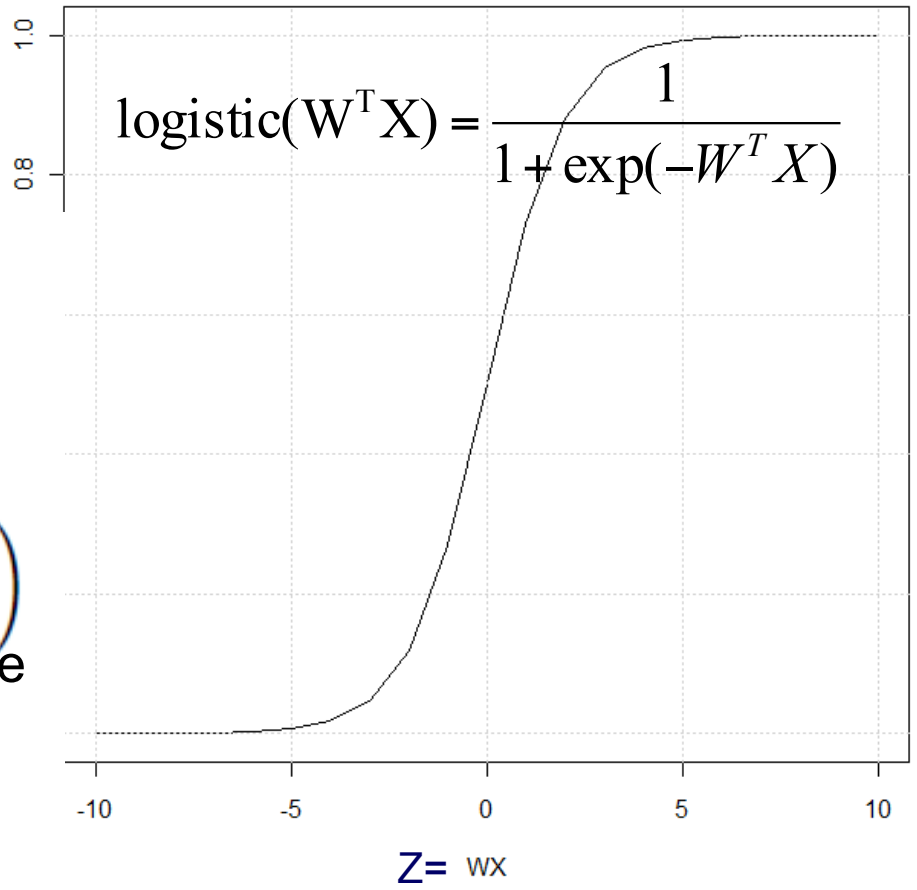
$$\frac{1}{1+\exp(-W^T X)}$$

Derivative of Logistic Function

- **Useful property of the derivative of the sigmoid function (logit inverse function) is its straight forward derivative (**

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\
&= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}
$$

Always positive



Notice g(z) is always bounded between [0,1] (a nice property) and as z increases g(z) approaches 1, as z decreases g(z) approaches to 0

Do a linear regression on Log odds = WX

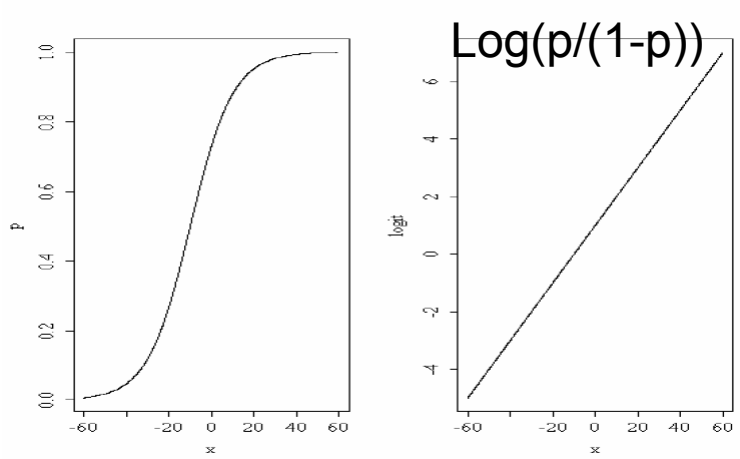
$$\log\left(\frac{p}{1-p}\right) = W^T X$$

$$\frac{p}{1-p} = \exp(W^T X)$$

$$p = (1-p)\exp(W^T X)$$

As shown earlier

$$p = \frac{\exp(W^T X)}{1+\exp(W^T X)} = \frac{1}{1+\exp(-W^T X^i)}$$



OLS (over the logit) can be problematic as logits for p's near 0 or 1 are infinite;

2 Approaches to Estimate Parameters of LR

- **Estimate weights via using Gaussian Naïve Bayes classifier**
  - Show equivalence between LR and Naïve Bayes (for continuous variables) and derive a means of setting the parameters (weights) using Gaussian Naïve Bayes classifier [See Mitchell 2005]
- **Via Maximum Likelihood Estimation**
  - is a statistical method for estimating the coefficients of a model.

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

LR via Naïve Bayes

- **Not used in practice**
- **Limited; need a more general method**
- **Using a Gaussian Naïve Bayes Classier the Naïve assumption may not be satisfied**
  - In this case we may wish to estimate the w_i *parameters directly from the data, rather* than going through the intermediate step of estimating the GNB parameters which forces us to adopt its more stringent modeling assumptions.

2 Approaches to Estimate Parameters of LR

- **Estimate weights via using Gaussian Naïve Bayes classifier**
  - Show equivalence between LR and Naïve Bayes (for continuous variables) and derive a means of setting the parameters (weights) using Gaussian Naïve Bayes classifier [See Mitchell 2005]
- **Via Maximum Likelihood Estimation**
  - is a statistical method for estimating the coefficients of a model.

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

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

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

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

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

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

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:

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

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.

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

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

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

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

Mistake in Charles's note

Writing the regularized optimization problem as a minimization gives

- Missing label in this formulation

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

The expression $-\log p(y_i|x_i;\beta)$ is called the "loss" for training example i. If the predicted probability, using β, of the true label y_i is close to 1, then the loss is small. But if the predicted probability of y_i is close to 0, then the loss is large. Losses are always non-negative; we want to minimize them. We also want to minimize the numerical magnitude of the trained parameters.

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

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

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

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

log loss

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

p p

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

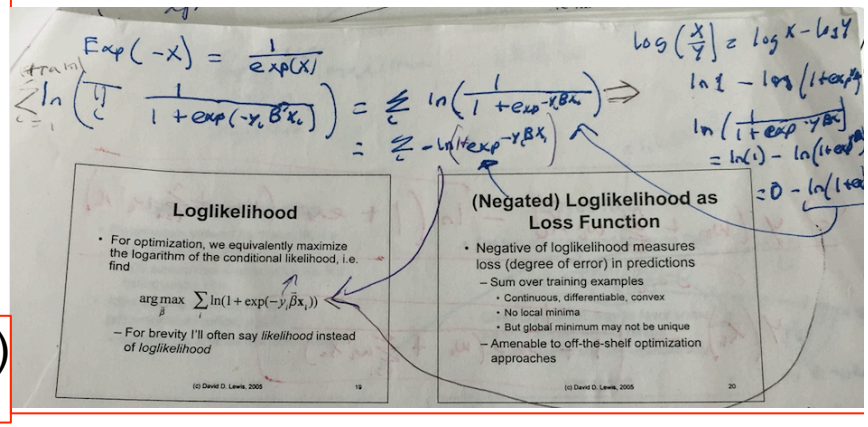
$$1 - g(x^{(i)})) = 1 - \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

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

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

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

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



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

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

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

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

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

This is a maximize version

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

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

Min log loss

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

Minimize 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).

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

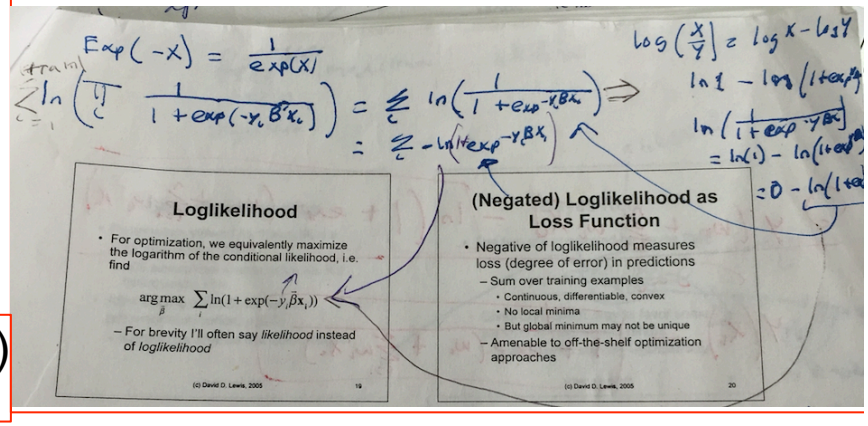
$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

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

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

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

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



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

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

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

Where m is defined

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

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

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

Squared Error loss term; 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.

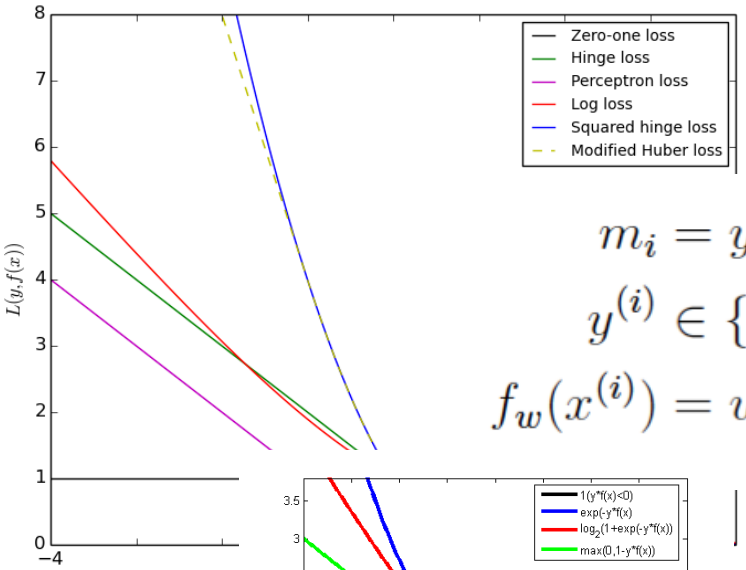
$$L_2(m) = (f_w(x) - y)^2 = (m-1)^2 \tag{14.11}$$

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

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

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

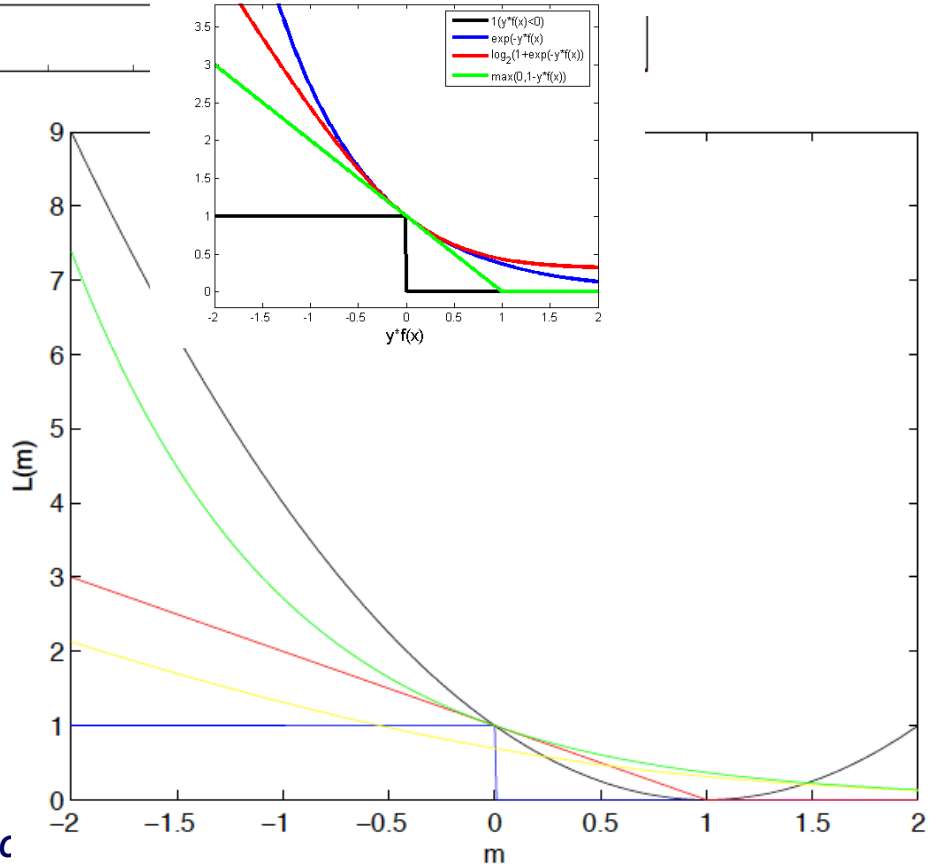
Loss Functions



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

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

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



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

Repeat

For j in 0..n #each variable

For i in 1..m #each example

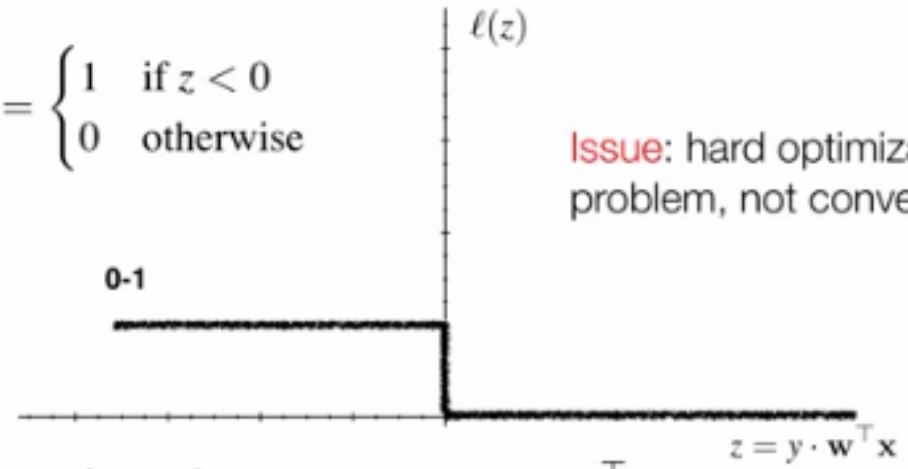
$$W_{j,t+1} = W_{j,t} + \alpha * (y - p) X_j$$

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

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!

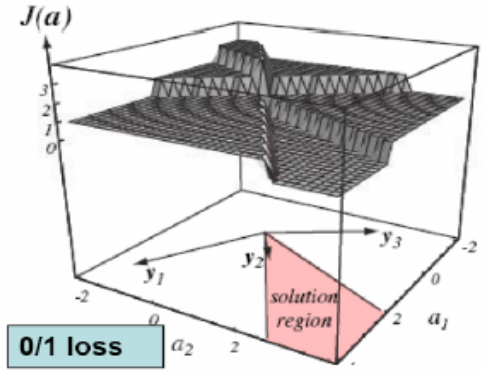
Let $y \in \{-1, 1\}$ and define $z = y \cdot \mathbf{w}^\top \mathbf{x}$

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

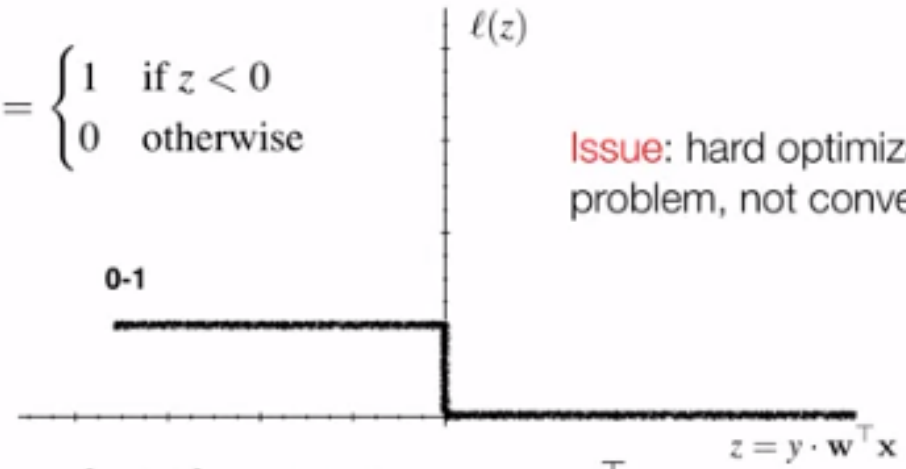
  L(y',y) = 0 when y'=y, otherwise L(y',y)=1
- Does not produce useful gradient since the surface of J is flat



What can we do?
- Approximate 0/1 loss with a convex loss surrogate
- See next slide

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!

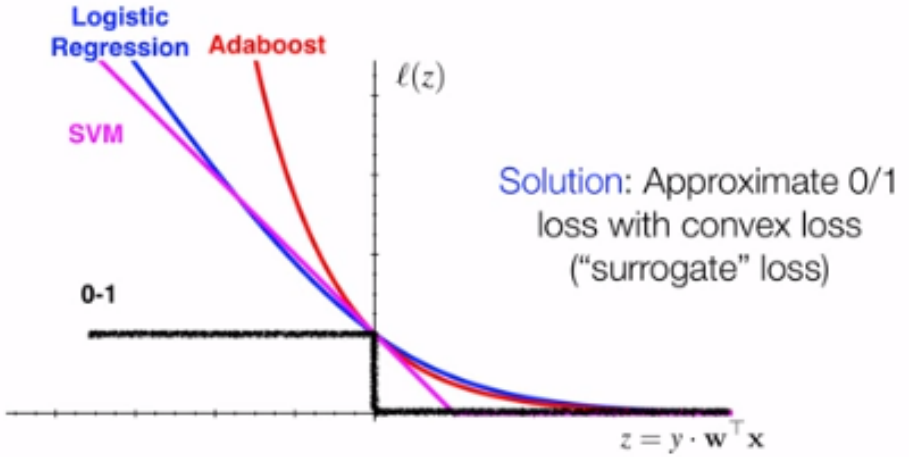
0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

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

Approximate 0/1 Loss

Approximate 0/1 loss with a convex loss surrogate



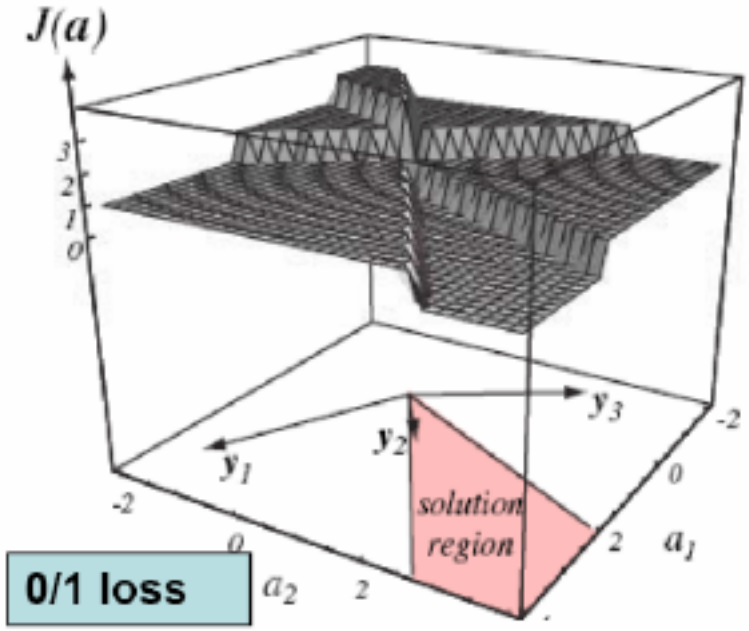
Solution: Approximate 0/1 loss with convex loss ("surrogate" loss)

SVM (hinge), Logistic regression (logistic), Adaboost (exponential)

Loss Functions

- 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

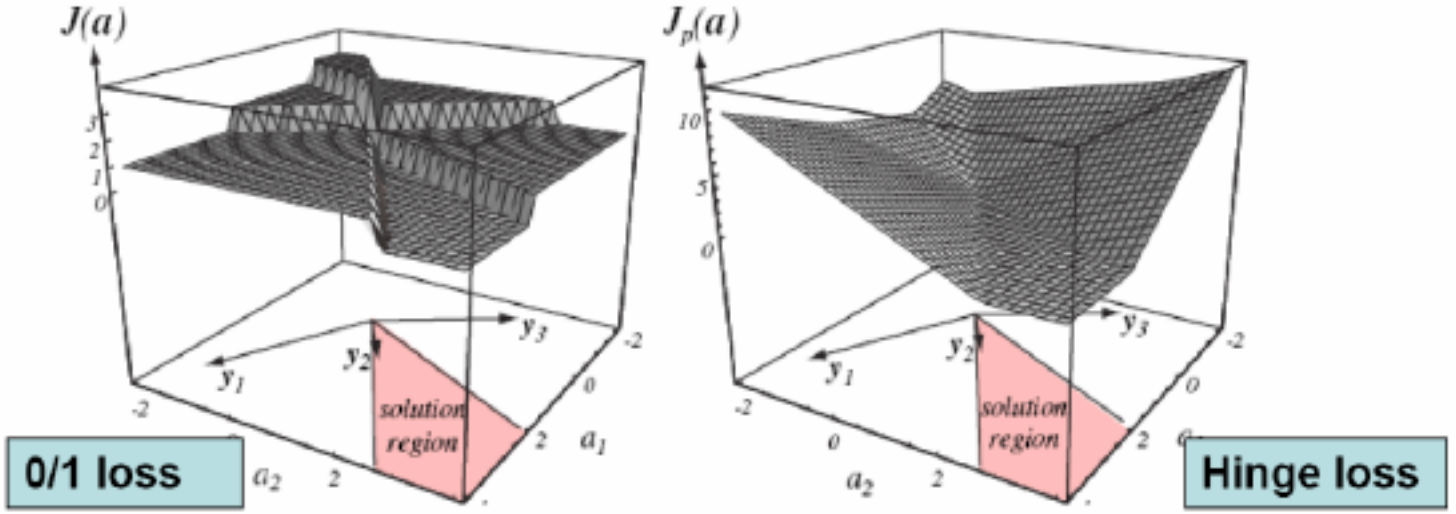


Loss Functions

- 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



es from Xiaoli Fern (OSU)

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

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

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

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

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

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

Distributed Gradient Descent

- **OLS**
- **Logistic Regression**
- **Bayesian logistic regression**
- **Probit regression**
- **Perceptron??**
- **SVMs and its many learning algorithms**
  - Linear SVMs
- **Neural Networks**

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

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

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

Setting α = 1/L #Lipschitz constant

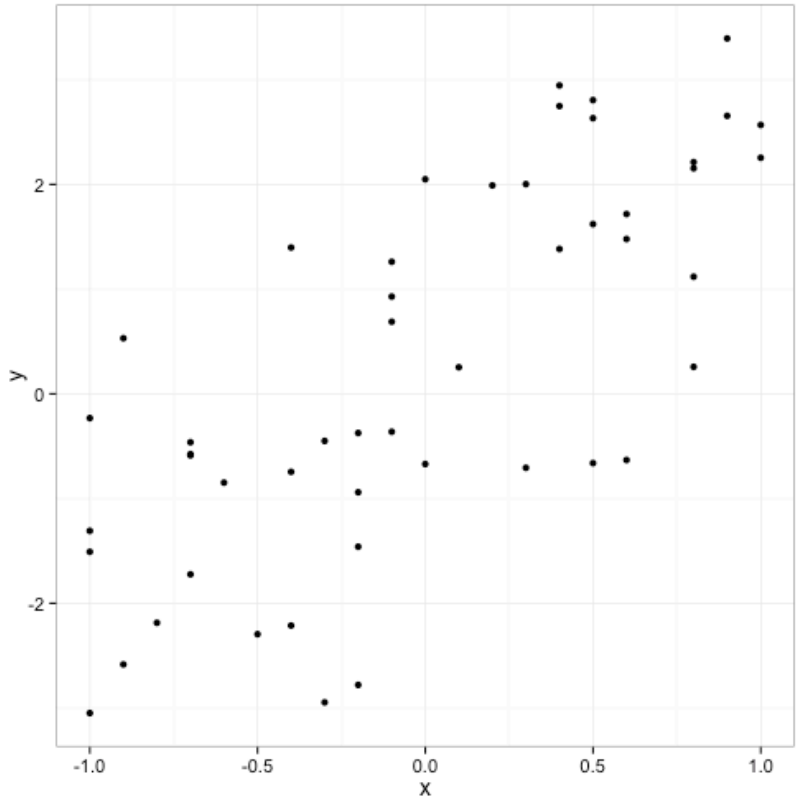
- **In OLS, f(b) is twice differentiable and its Hessian is XtX, which does not depend on b. Therefore, the smallest Lipschitz constant of ∇f is the largest eigenvalue of XtX. Naturally, we want to take the biggest steps possible, so if we can compute the Lipschitz constant L we se α=1/L.**
- http://www.statisticsviews.com/details/feature/5722691/Getting-to-the-Bottom-of-Regression-with-Gradient-Descent.html

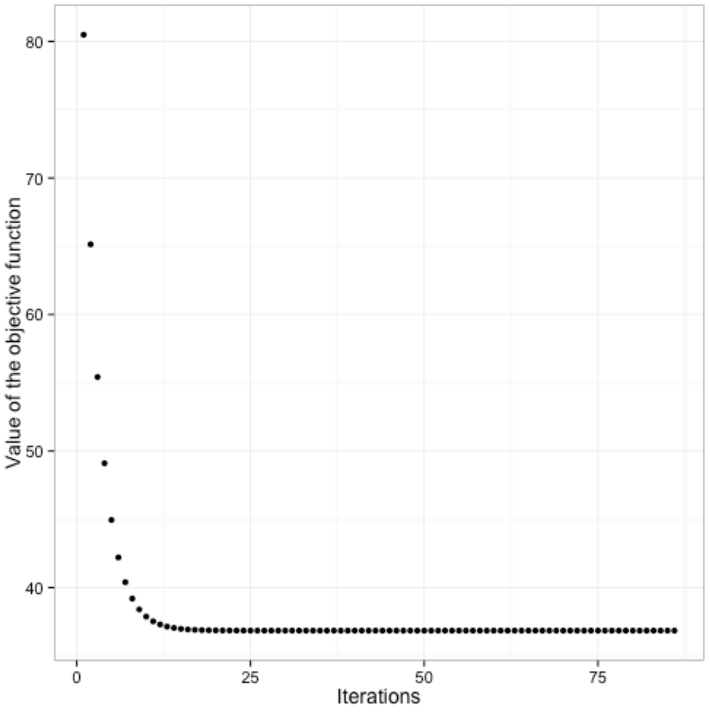
In OLS, $f(\mathbf{b})$ is twice differentiable and its Hessian is $\mathbf{X}^t\mathbf{X}$, which does not depend on $\mathbf{b}$. Therefore, the smallest Lipschitz constant of ∇f is the largest eigenvalue of $\mathbf{X}^t\mathbf{X}$. Naturally, we want to take the biggest steps possible, so if we can compute the Lipschitz constant L we set $\alpha = 1/L$.

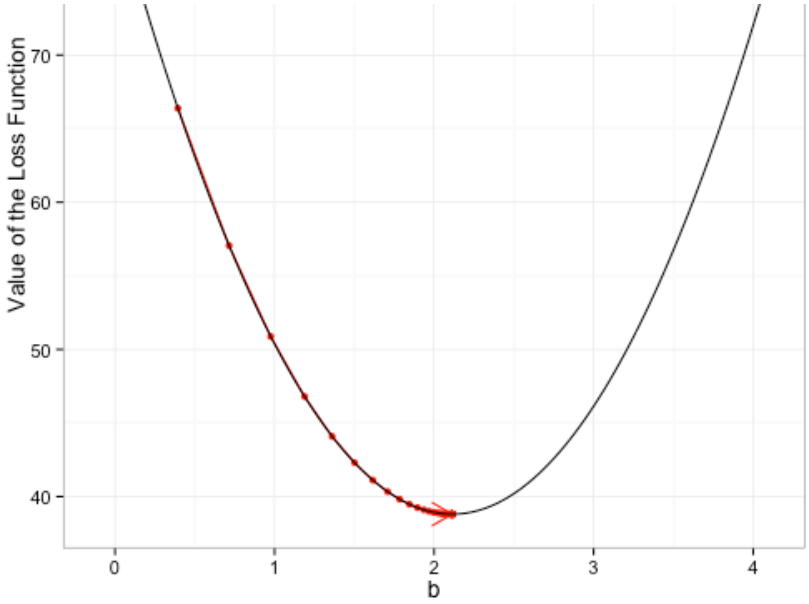
Using the simple example we employed previously, we observe that when our choice of α is not small enough, the norm of the gradient will diverge towards infinity and the algorithm will not converge. (Note that the code for this example is provided below but in the interest of space, we do not include the output in this article.)

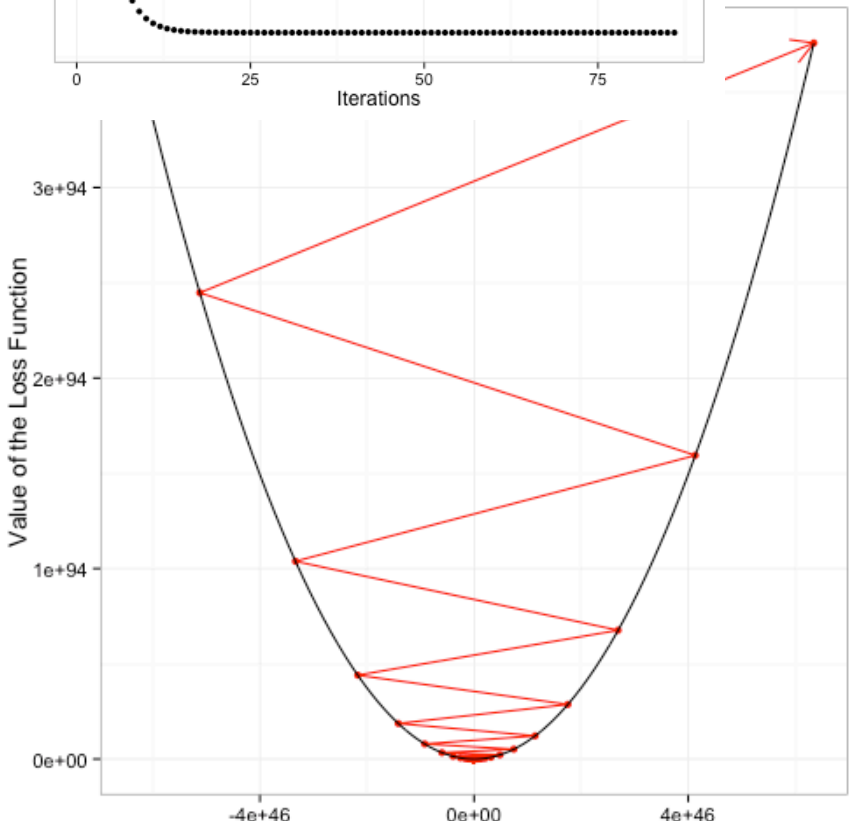
```
simple_ex2 <- gdescent(f,grad_f,X,y,alpha=0.05,liveupdates=TRUE)
```

The live updates in this example show the norm of the gradient in each iteration and we can see that the norm of the gradient diverges when α is not sufficiently small. The following two figures illustrate why this might occur. In the first figure, α is sufficiently small so each iteration in the algorithm results in a step towards the minimum, resulting in convergence of the algorithm.


y
x
2
0
-2
-1.0
-0.5
0.0
0.5
1.0


Value of the objective function
Iterations
80
70
60
50
40
0
25
50
75


Value of the Loss Function
b
70
60
50
40
0
1
2
3
4


Value of the Loss Function
3e+94
2e+94
1e+94
0e+00
-4e+46
0e+00
4e+46

Quiz 11.3.1 Loss Functions

- Loss function consists of:
  - A: loss term ($L(m_i(w)$, expressed in terms of the margin of each testing example) and
  - B: loss term ($L(m_i(w)$, expressed in terms of the margin of each support vector) and
  - C: regularization term ($R(w)$ expressed as a function of the model complexity)
- D: 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))

Which of the above statements are true?

- A, B, C, D
- B, C, D
- C, D CORRECT
- None of the above

ML Objectives

Almost all machine learning objectives are optimized using this update

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

w is a vector of dimension d
we're trying to find the best w via optimization

Distributed Gradient Descent

- **OLS**
- **Logistic Regression**
- **Bayesian logistic regression**
- **Probit regression**
- **Perceptron??**
- **SVMs and its many learning algorithms**
  - Linear SVMs
- **Neural Networks**

OLS 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 $W_{i+1} = W_i + \alpha * (y^i - W_i X^i) X^i$
    - 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

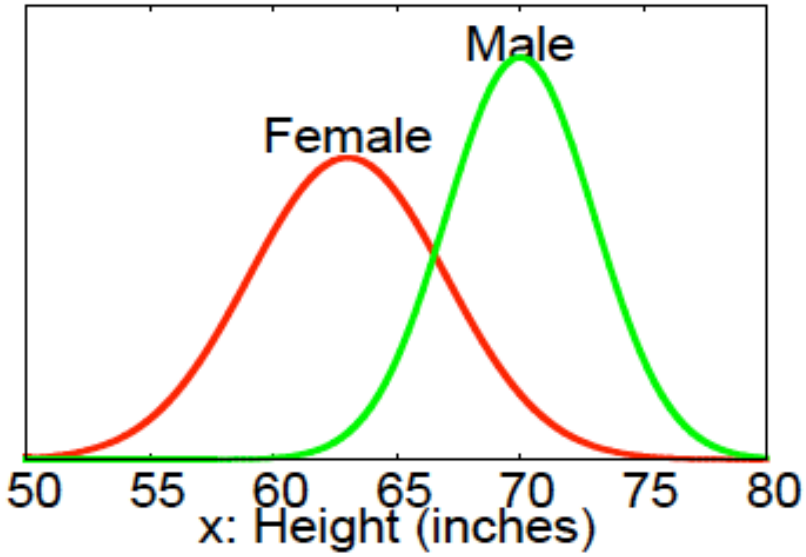
General framework for gradient descent Notebook

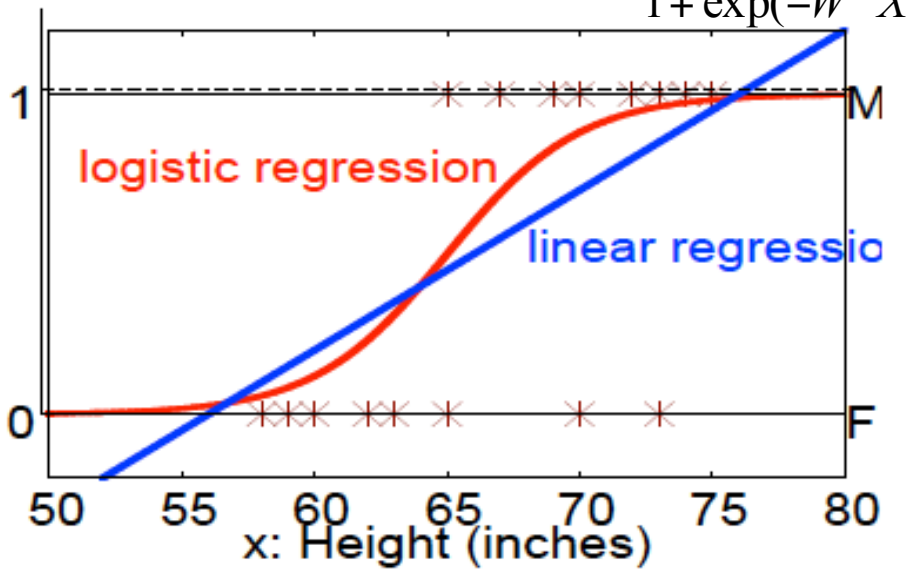
TODO : Liang Notebook in SPARK

Logistic Regression in One Slide

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

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





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

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

Logistic Regression

James G. Shanahan

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Large Scale Machine Learning

MIDS, UC Berkeley

Lecture 4, April, 2015

Assumption

- ***y*** **ϵ {-1,+1},** ***x*** **is feature vector.** ***p*** **is defined as**

$$p = \Pr(y = +1 \mid X)$$

- **Linearly related to features after logit transformation**

$$\log(\frac{p}{1-p}) = \mathbf{w}^T \cdot \mathbf{x} - b$$

- **Then probability is a logistic function of w·x,**

$$p = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x} + b)}$$

where ***w*** **is the coefficient vector and b is the intercept.**

Logistic regression
(a close relation of SVM when we use priors otherwise also)

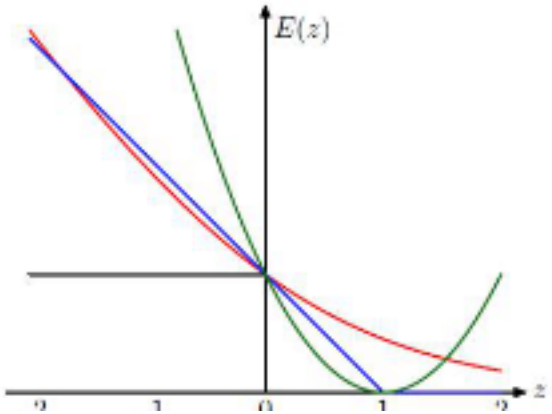
Logistic regression

Maximize the likelihood of w:

$$P(\mathbf{y} \mid X, \mathbf{w}) = \prod_i \frac{1}{1 + e^{-y_i(\mathbf{w}^\top \mathbf{x}_i)}}$$

$$\log P(\mathbf{y} \mid X, \mathbf{w}) = -\sum_i \log(1 + e^{-y_i(\mathbf{w}^\top \mathbf{x}_i)})$$

Pay loss:



The objective function is very similar to the SVM .. except for the loss function part Logistic regression uses the log-loss, SVM uses the hinge-loss

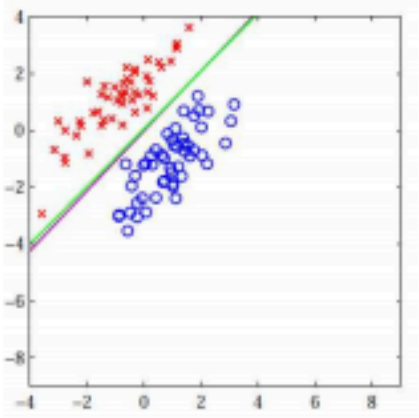
Log loss (in RED) and hinge loss (BLUE) are similar (log loss is differentiable at all points; not sparse …each point affect the gradient learning and the final result…

Logistic Regression

- What does the decision boundary look like for Logistic Regression?
- At the decision boundary labels +1/-1 becomes equiprobable

$$
\begin{aligned}
P(y = +1 \mid \mathbf{x}, \mathbf{w}) &= P(y = -1 \mid \mathbf{x}, \mathbf{w}) \\
\frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} &= \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{x})} \\
\exp(-\mathbf{w}^\top \mathbf{x}) &= \exp(\mathbf{w}^\top \mathbf{x}) \\
\mathbf{w}^\top \mathbf{x} &= 0
\end{aligned}
$$

- The decision boundary is therefore linear ⇒ Logistic Regression is a linear classifier (note: it's possible to kernelize and make it nonlinear)



Logistic Regression: Maximum-a-Posteriori Solution

- Let's assume a Gaussian prior distribution over the weight vector $\mathbf{w}$

$$P(\mathbf{w}) = \mathcal{N}or(\mathbf{w} \mid 0, \lambda^{-1}\mathbf{I}) = \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}\right)$$

- Maximum-a-Posteriori Solution: $\hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log P(\mathbf{w} \mid \mathcal{D})$

$$= \arg\max_{\mathbf{w}} \{\log P(\mathbf{w}) + \log P(\mathcal{D} \mid \mathbf{w}) - \log P(\mathcal{D})\}$$

$$= \arg\max_{\mathbf{w}} \{\log P(\mathbf{w}) + \log P(\mathcal{D} \mid \mathbf{w})\}$$

Approx

$$= \arg\max_{\mathbf{w}} \left\{-\frac{D}{2}\log(2\pi) - \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w} + \sum_{n=1}^{N} -\log[1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)]\right\}$$

$$= \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \log[1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)] + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w} \quad \text{(ignoring constants and changing max to min)}$$

- No closed-form solution exists but we can do gradient descent on $\mathbf{w}$
- See "A comparison of numerical optimizers for logistic regression" by Tom Minka on optimization techniques (gradient descent and others) for logistic regression (both MLE and MAP) http://www.cs.utah.edu/~piyush/teaching/20-9-print.pdf

Logistic Regression: MLE vs MAP (summary)

- MLE solution:

$$\hat{\mathbf{w}}_{MLE} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \log[1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)]$$

- MAP solution:

$$\hat{\mathbf{w}}_{MAP} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \log[1 + \exp(-y_n \mathbf{w}^\top \mathbf{x}_n)] + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}$$

- **Take-home messages** (we already saw these before :-)):
  - MLE estimation of a parameter leads to unregularized solutions
  - MAP estimation of a parameter leads to regularized solutions
  - The prior distribution acts as a regularizer in MAP estimation
- Note: For MAP, different prior distributions lead to different regularizers
  - Gaussian prior on $\mathbf{w}$ regularizes the ℓ_2 norm of $\mathbf{w}$
  - Laplace prior $\exp(-C||\mathbf{w}||_1)$ on $\mathbf{w}$ regularizes the ℓ_1 norm of $\mathbf{w}$

Logistic Regression: some notes

- The objective function is very similar to the SVM
  - .. except for the loss function part
  - Logistic regression uses the log-loss, SVM uses the hinge-loss
- Generalization to more than 2 classes is straightforward
  - .. using the *soft-max* function instead of the logistic function

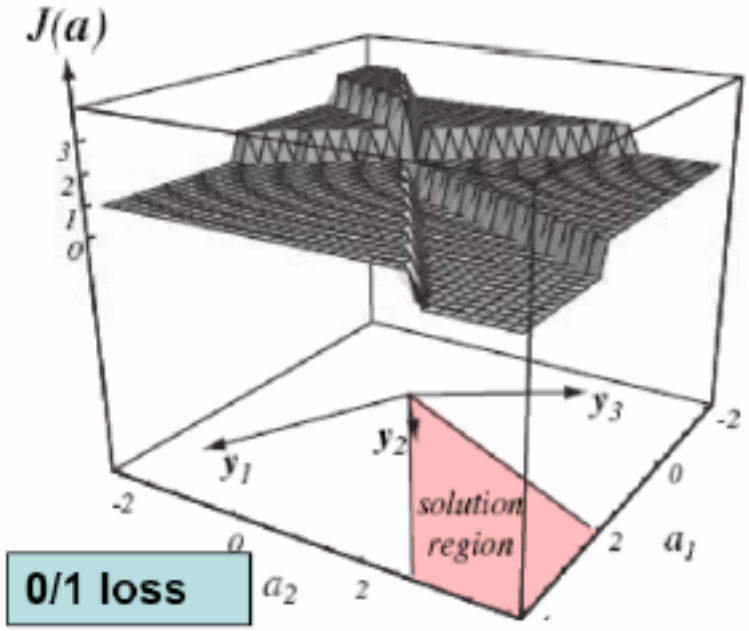
$$P(y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_k \exp(\mathbf{w}_k^\top \mathbf{x})}$$

  - We maintain a separator weight vector $\mathbf{w}_k$ for each class k
- Possible to kernelize it to learn nonlinear boundaries

Loss Functions

- 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

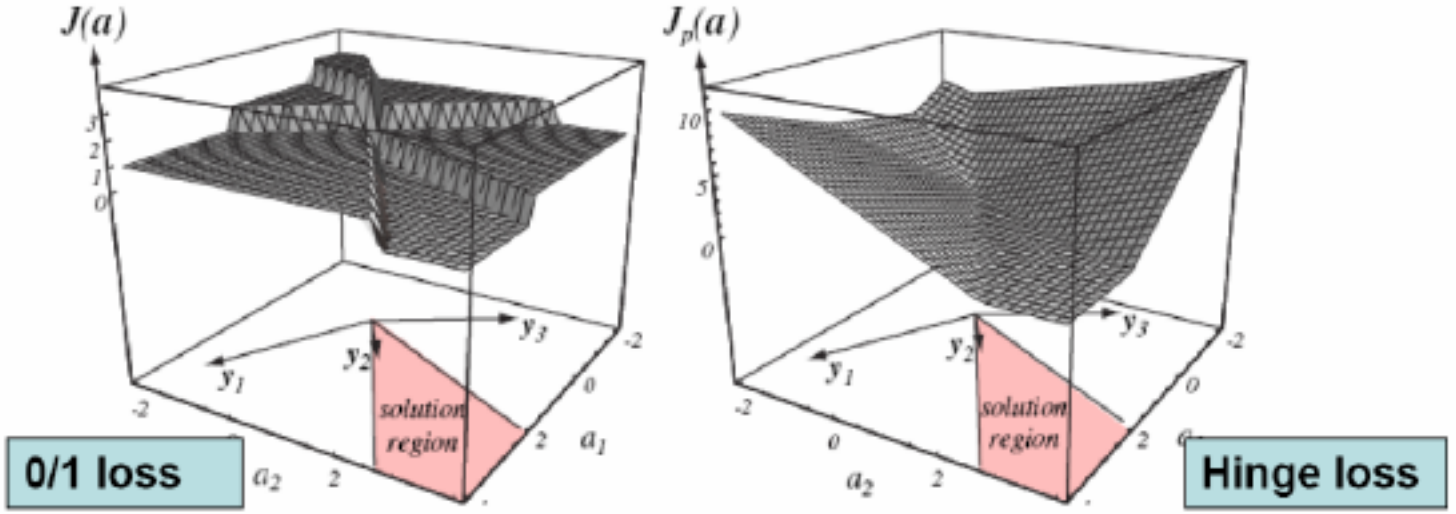


Loss Functions

- 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



es from Xiaoli Fern (OSU)

Gradient for hinge loss function

Gradient For Hinge Loss Function

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \max(0, -y_i \mathbf{w}\cdot\mathbf{x}_i)$$

$$J_i(\mathbf{w}) = \max(0, -y_i \mathbf{w}\cdot\mathbf{x}_i)$$

$$\frac{\partial J_i}{\partial w_j} = \begin{cases} 0 & \text{if } y_i \mathbf{w}\cdot\mathbf{x}_i > 0 \\ -y_i x_{ij} & \text{otherwise} \end{cases}$$

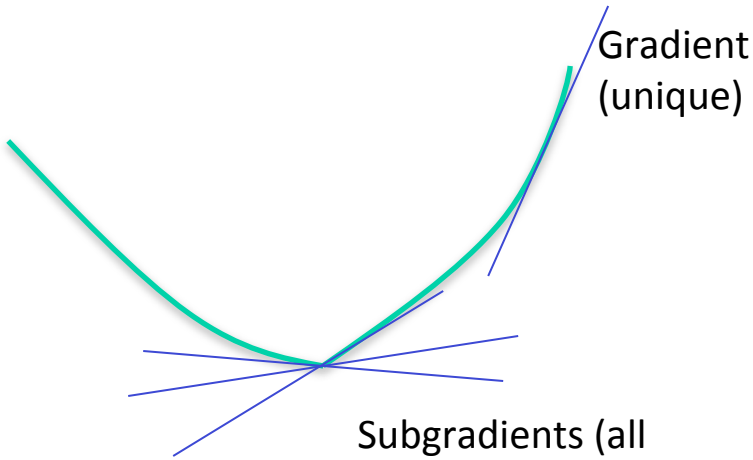
$$\nabla J_i = \begin{cases} 0 & \text{if } y_i \mathbf{w}\cdot\mathbf{x}_i > 0 \\ -y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

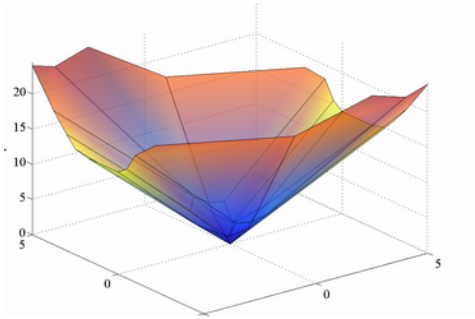
$$\nabla J = \frac{1}{N} \sum_{y_i \mathbf{w}\cdot\mathbf{x}_i < 0} -y_i \mathbf{x}_i$$

NOTE:
Show Hinge loss
We do NOT show the regularization loss (in the case of a perceptron)
In SVM/Bayesian Logistic Regression (aka MAP logistic regression)

Stochastic Subgradient Descent

- **Gradient descent optimization in perceptron (smooth objective)**
- **Subgradient descent in pegasos (non differentiable objective)**





MLE: Maximum Likelihood Expectation

- **Maximize the Log likelihood:**

$$l(W) = \ln \prod_i P_i = \ln \prod_i \left(\frac{1}{1+\exp(-\mathbf{w}^T \cdot \mathbf{x}_i + b)} \right)^{\frac{1+y_i}{2}} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^T \cdot \mathbf{x}_i + b)} \right)^{\frac{1-y_i}{2}}$$

Maximize this log likelihood is equal to minimizing the following

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

$$l(W) = \sum_i \log\left(1 + \exp\left(- y\mathbf{w}^T \mathbf{x}_i\right)\right)$$

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

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

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

log loss

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

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).

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

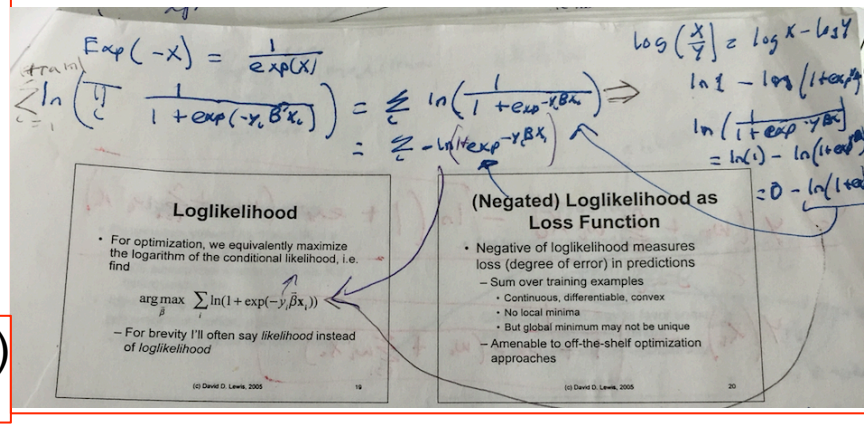
$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

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

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

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

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



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

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

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

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

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

This is a maximize version

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

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

Gradient Descent

- **Objective Function (Logloss):**

$$\sum_i \log\left(1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)\right)$$

- **Gradient**

$$\nabla\mathbf{w} = \sum_i -y\left(1-\frac{1}{1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)}\right)\cdot\mathbf{x}_i$$

- **Update w till it converges**

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

Regularization

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

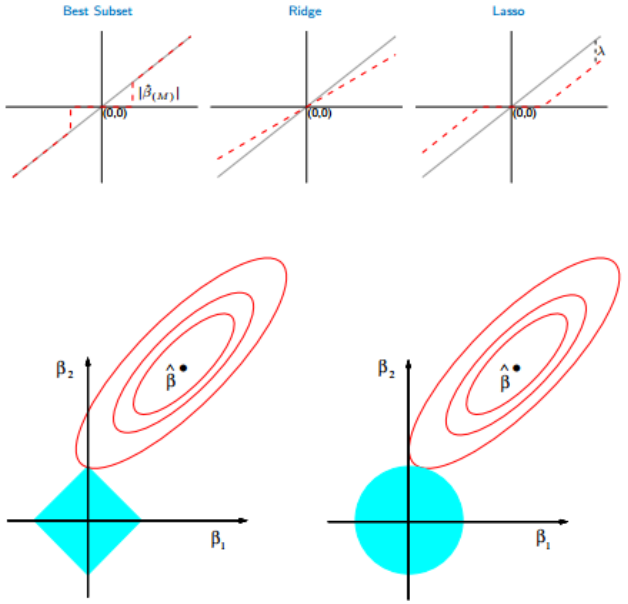


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

From The Elements of Statistical Learning

Gradient Descent-Lasso

- **Objective Function (Logloss + LassReg):**

$$\sum_i \log\left(1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)\right)+\lambda|\mathbf{w}|$$

- **Gradient**

$$\nabla\mathbf{w}=\sum_i -y\left(1-\frac{1}{1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)}\right)\cdot\mathbf{x}_i+\lambda\left(\mathbf{1}_{>0}(\mathbf{w})\cdot 2-1\right)$$

- **Update w till it converges**

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

Gradient Descent-Ridge

- **Objective Function (Logloss+RidgeReg):**

$$\sum_i \log\left(1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)\right)+\lambda\mathbf{w}^2$$

- **Gradient**

$$\nabla\mathbf{w}=\sum_i -y\left(1-\frac{1}{1+\exp\left(-y(\mathbf{w}^T\mathbf{x}_i+b)\right)}\right)\cdot\mathbf{x}_i+\lambda\mathbf{w}$$

- **Update w till it converges**

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

Python Notebooks for Logistic Regression

- **NoteBook for Logistic Regression on a single node**
  - http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kurbw695jdvxib0/LogisticRegression-Single-Core-Notebook-.ipynb

- **NoteBook for distributed Logistic Regression on Spark**
  - http://nbviewer.ipython.org/urls/dl.dropbox.com/s/r20ff7q0yni5kiu/LogisticRegression-Spark-Notebook.ipynb

Section 11.4 Display at time 09:10

Logistic Regression: Single node Code

Python-Gradient Descent

Input:

- data: Feature Information
- y: label information
- eta: learning rate
- iter_num : maximum iteration number
- regPara: regularization parameter
- stopCriteria: stop criteria

Output:

- w : coeffcients of boudary

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kurbw695jdvxib0/LogisticRegression-Single-Core-Notebook-.ipynb

Section 11.4 Display at time 09:15

```
#gradient descent (and with NO stochasticity!)
# Objective Function
# minw   λ/2   w'w   +   1/m Σi log(1+exp(yi(w'xi − b)))
# gradient
#   -y*(1-1/(1+exp(yi(w'xi − b))))*x

def logisticReg_GD(data,y,w=None,eta=0.05,iter_num=500,regPara=0.01,stopCriteria=0.0001,reg="Ridge"):
    data = np.append(data,np.ones((data.shape[0],1)),axis=1)
    if w is None:
        w = np.random.normal(size=data.shape[1])
    for i in range(iter_num):
        wxy = np.dot(data,w)*y
        g = np.dot(data.T,-y*(1-1/(1+np.exp(-wxy))))/data.shape[0] # Gradient of log loss'
        wreg = w*1
        wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
        if reg == "Ridge":
            wreg = w*1
            wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
        elif reg == "Lasso":
            wreg = w*1
            wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
            wreg = (wreg>0).astype(int)*2-1
        else:
            wreg = np.zeros(w.shape[0])
        wdelta = eta*(g+regPara*wreg) #gradient: log loss + regularized term
        if sum(abs(wdelta))<=stopCriteria*sum(abs(w)): #Convergence condition
            break
        w = w - wdelta
    return w
```

NOTE This is the single node implementation of logistic regression.

The distributed version will follow momentarily.

Please click on the link and take some time out to review this notebook.

Distributed Logistic Regression: PySpark Code

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/r20ff7q0yni5kiu/LogisticRegression-Spark-Notebook.ipynb

```
#gradient descent (and with NO stochasticity!)
# Objective Function
# minw   λ/2   w'w   +   1/m Σi log(1+exp(yi(w'xi − b)))
# gradient
#   -y*(1-1/(1+exp(yi(w'xi − b))))*x

def logisticReg_GD_Spark(data,y,w=Non
    #eta learning rate
    #regPara
    dataRDD = sc.parallelize(np.append(y[:,None],data,axis=1)).cache()
    if w is None:
        w = np.random.normal(size=data.shape[1]+1)
    for i in range(iter_num):
        w_broadcast = sc.broadcast(w)
        g = dataRDD.map(lambda x: -x[0]*(1-1/(1+np.exp(-x[0]*np.dot(w_broadcast.value,np.append(x[1:],1))))) \
                        *np.append(x[1:],1)).reduce(lambda x,y:x+y)/data.shape[0] # Gradient of logloss

        if reg == "Ridge":
            wreg = w*1
            wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
        elif reg == "Lasso":
            wreg = w*1
            wreg[-1] = 0 #last value of weight vector is bias term; ignore in regularization
            wreg = (wreg>0).astype(int)*2-1
        else:
            wreg = np.zeros(w.shape[0])
        wdelta = eta*(g+regPara*wreg)  #gradient: hinge loss + regularized term
        if sum(abs(wdelta))<=stopCriteria*sum(abs(w)): # converged as updates to weight vector are small
            break
        w = w - wdelta
    return w
```

Section 11.4 Display at time 09:38

How to run the Spark code

Ridge Logistic Rigression

```
np.random.seed(400)
w_gd_spark_sep_ridge = logisticReg_GD_Spark(data_sep,y_sep,reg="Ridge")
print w_gd_spark_sep_ridge
```

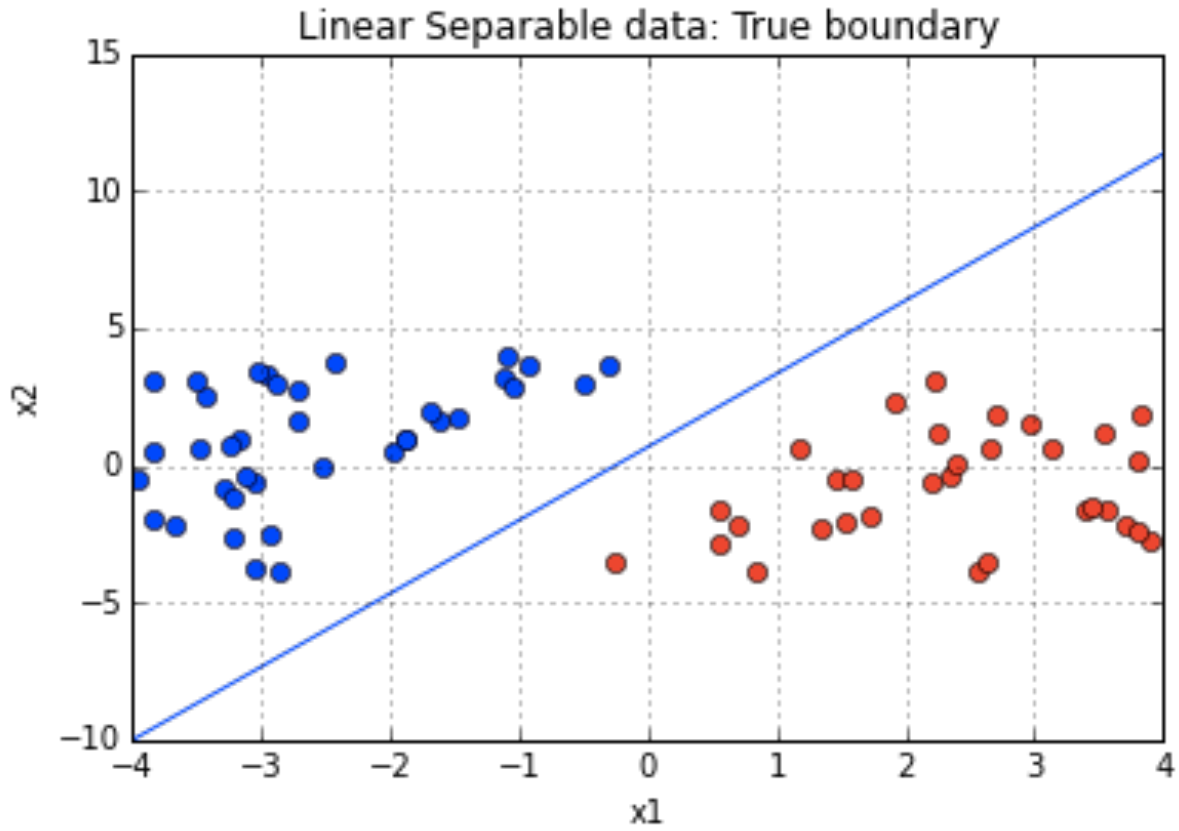
```
[ 1.84106359 -0.63166074  0.00472968]
```

Lasso Logistic Rigression

```
np.random.seed(400)
w_gd_spark_sep_lasso = logisticReg_GD_Spark(data_sep,y_sep,reg="Lasso")
print w_gd_spark_sep_lasso
```

```
[ 1.90212754 -0.60763246  0.18656826]
```

Linear separable data



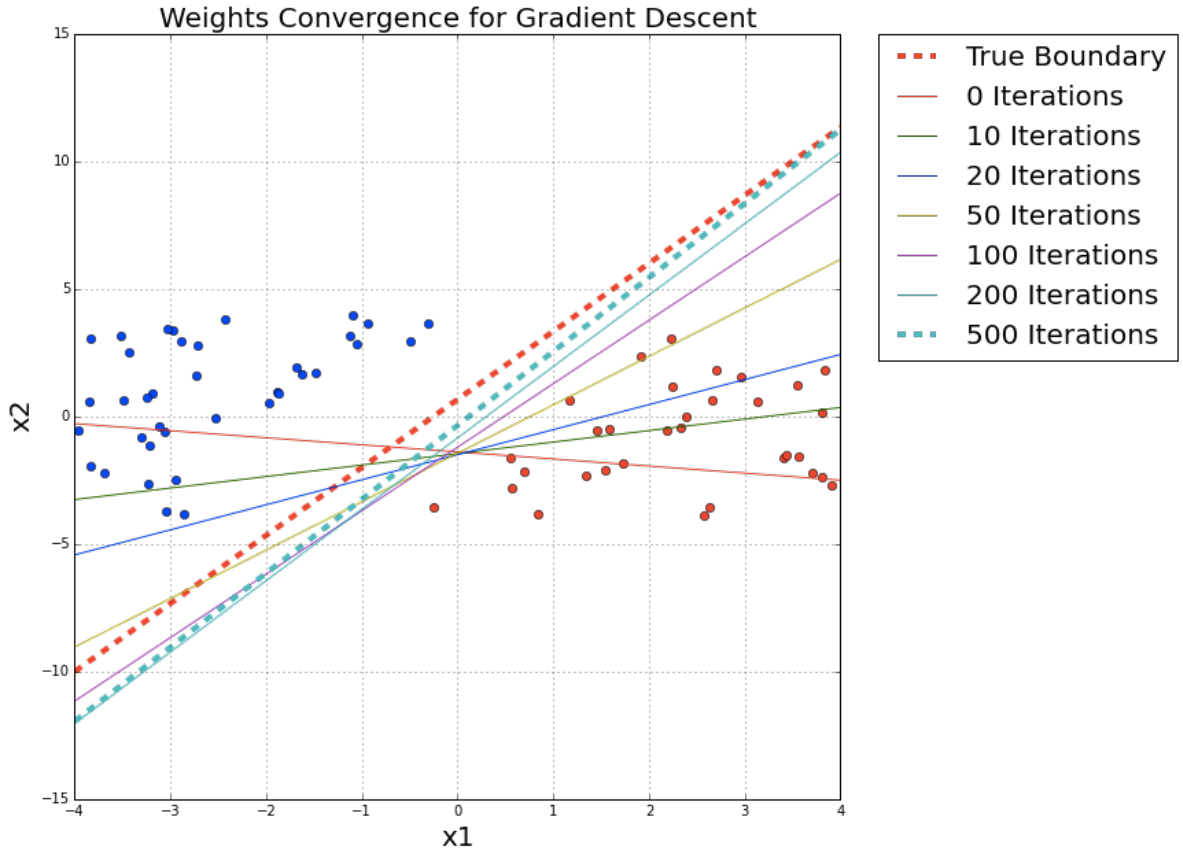
Ridge logistic regression

Ridge Logistic Regression

```
np.random.seed(400)
w_gd_sep_ridge = logisticReg_GD(data_sep,y_sep,reg="Ridge")
print w_gd_sep_ridge
```

```
[ 1.84106359 -0.63166074  0.00472968]
```

How w gets converged (ridge)



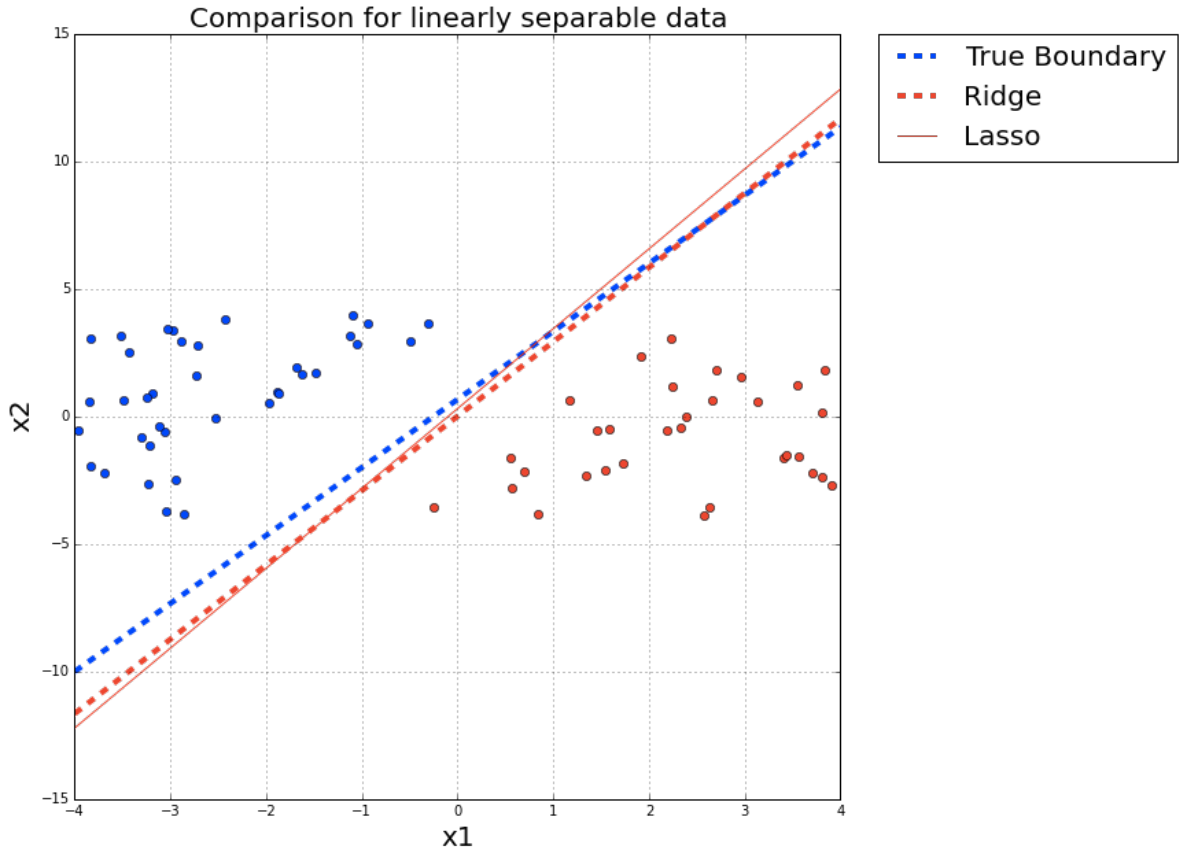
Lasso logistic regression

Lasso Logistic Regression

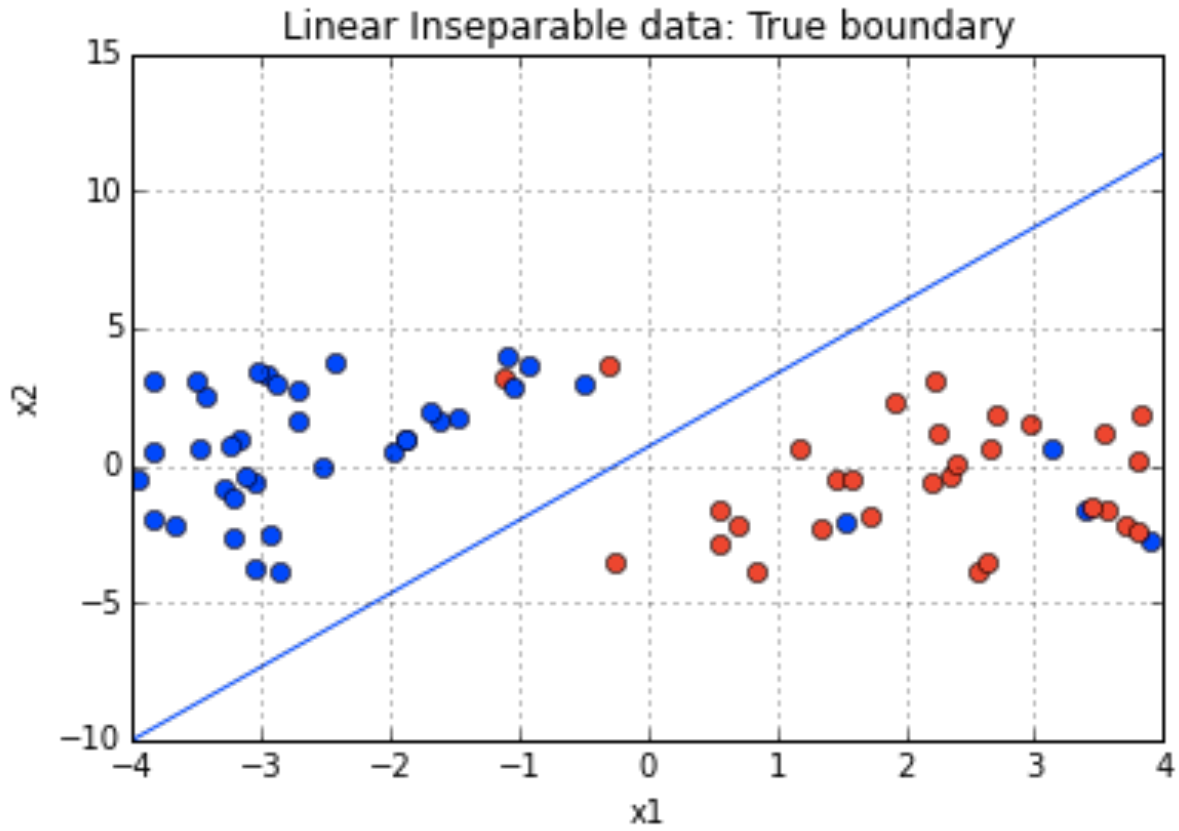
```
np.random.seed(400)
w_gd_sep_lasso = logisticReg_GD(data_sep,y_sep,reg="Lasso")
print w_gd_sep_lasso
```

```
[ 1.90212754 -0.60763246  0.18656826]
```

Comparison


Comparison for linearly separable data
True Boundary
Ridge
Lasso
x1
x2
15
10
5
0
−5
−10
−15
−4
−3
−2
−1
0
1
2
3
4

Linear inseparable data



Ridge logistic regression

Ridge Logistic Regression

```
np.random.seed(400)
w_gd_insep_ridge = logisticReg_GD(data_insep,y_insep,reg="Ridge")
print w_gd_insep_ridge
```

```
[ 0.88960208 -0.06413379 -0.24526051]
```

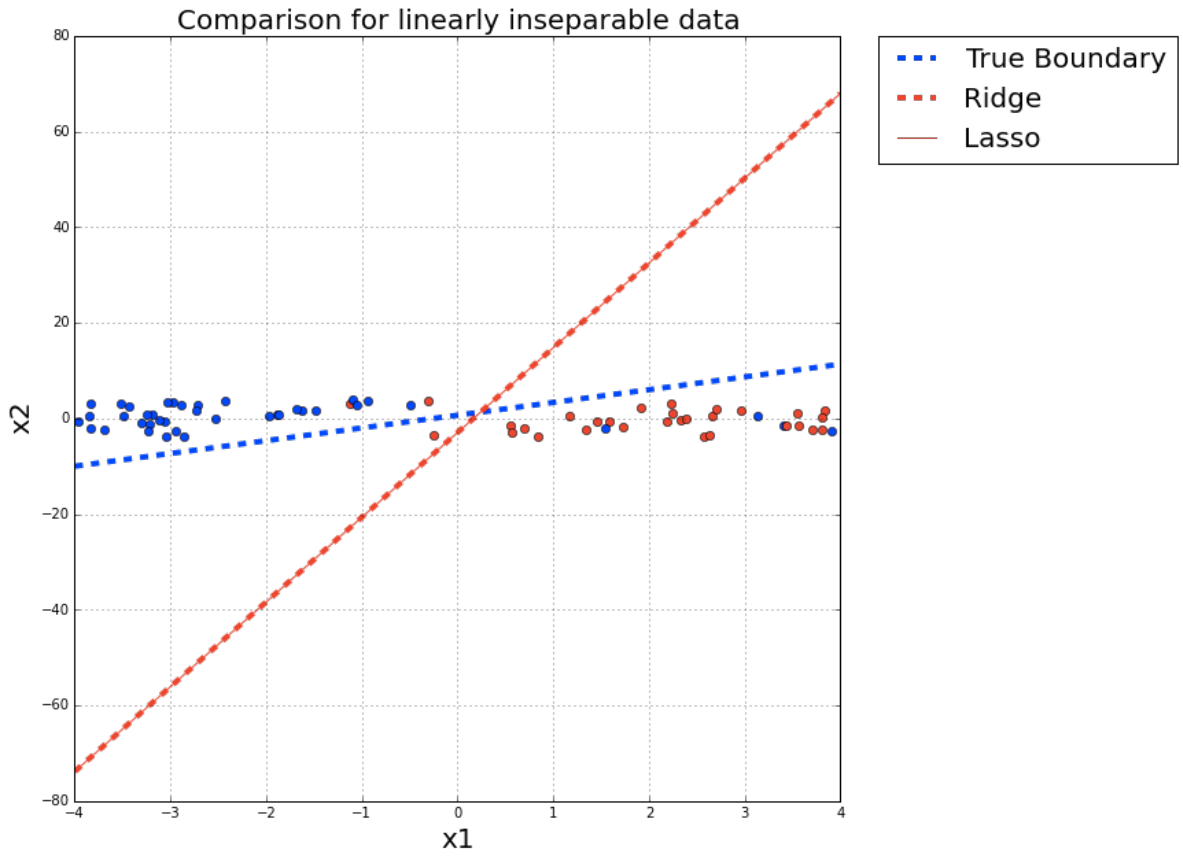
Lasso logistic regression

Lasso Logistic Regression

```
np.random.seed(400)
w_gd_insep_lasso = logisticReg_GD(data_insep,y_insep,reg="Lasso")
print w_gd_insep_lasso
```

```
[ 0.88745901 -0.0500822  -0.14752302]
```

Comparison


Comparison for linearly inseparable data
True Boundary
Ridge
Lasso
x1
x2
80
60
40
20
0
−20
−40
−60
−80
−4
−3
−2
−1
0
1
2
3
4

- **End of section**

Logit Space versus logistic function

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

Logit (Log odds) produces a value in logit space

$$\frac{p}{1-p} = \exp(W^T X)$$

$$p = (1-p)\exp(W^T X)$$

$$p = \frac{\exp(W^T X)}{1+\exp(W^T X)} = \frac{1}{1+\exp(-W^T X^i)}$$

Logistic Function produces a probability

Derivative of Logistic Function

$$y = \frac{1}{1+e^{-f(X)}} \qquad \frac{dy}{dX} = y(1-y)\frac{df}{dX}$$

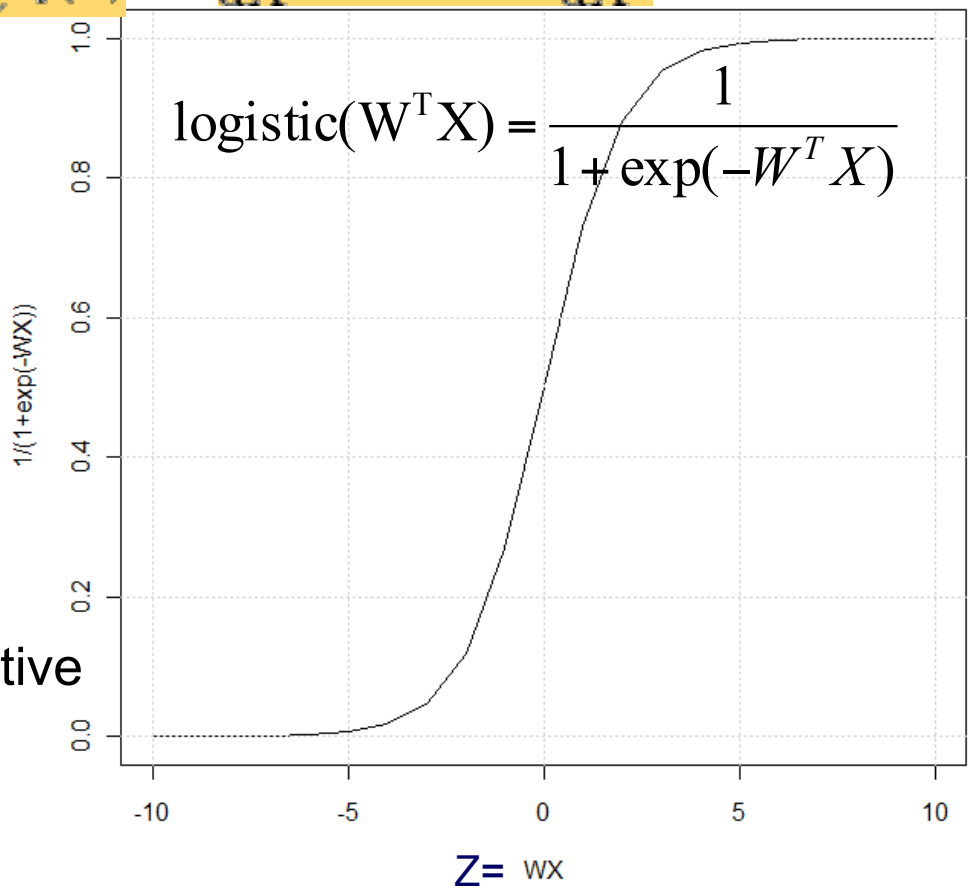
Derivative of Logistic Function

- **Useful property of the derivative of the sigmoid function (logit inverse function) is its straight forward derivative**

$$y = \frac{1}{1+e^{-f(X)}} \qquad \frac{dy}{dX} = y(1-y)\frac{df}{dX}.$$

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

Always positive



Notice g(z) is always bounded between [0,1] (a nice property)
and as z increases g(z) approaches 1,
as z decreases g(z) approaches to 0

Parametric Form of Logistic Regression

- 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

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

Notice that equation (17) follows directly from equation (16), because the sum of these two probabilities must equal 1.

Training LR: Maximize conditional data likelihood

- **One reasonable approach to training Logistic Regression is to choose parameter values that maximize the conditional data likelihood.**
- **The conditional data likelihood is the probability of the observed Y values in the training data, conditioned on their corresponding X values. We choose parameters W that satisfy**
- $$W \leftarrow \arg\max_{W} \prod_{l} P(Y^l | X^l, W)$$
  - where W = <w0,w1 ...wni> is the vector of parameters to be estimated, Y^l denotes the observed value of Y in the lth training example, and X^l denotes the value of X in the lth training example.
  - The expression to the right of the argmax is the conditional data likelihood.
  - Here we include W in the conditional, to emphasize that the expression is a function of the W we are attempting to maximize.

-

LR: Maximize conditional data likelihood

- **The expression to the right of the *argmax* is the conditional data likelihood.**

$$W \leftarrow \arg\max_{W} \prod_{l} P(Y^l | X^l, W)$$

Select W s:t likelihood of W generating the data is maximized

$$L(\vec{w}, b) = \prod_{i=1}^{n} p(\vec{x}_i)^{y_i} (1 - p(\vec{x}_i)^{1-y_i}$$

Y can take only values 0 or 1, so only one of the two terms in the expression will be non-zero for any given Y^l; *recall m^0 =1 .*

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

Working with logs is simpler and more effective computationally; amenable to off-the-shelf optimization approaches; concave function in W so gradient ascent will converge to global maximum (though many may exist). L(W) continuous, differentiable

Logistic Regression has a global optimum

- ***Since l(w) is concave function of w; so local optimum is a global optimum!!!***

- **Bad news:**
  - no closed-form solution to maximize *l(w)*
  - *gradient equations are non-linear*

Maximize conditional data likelihood

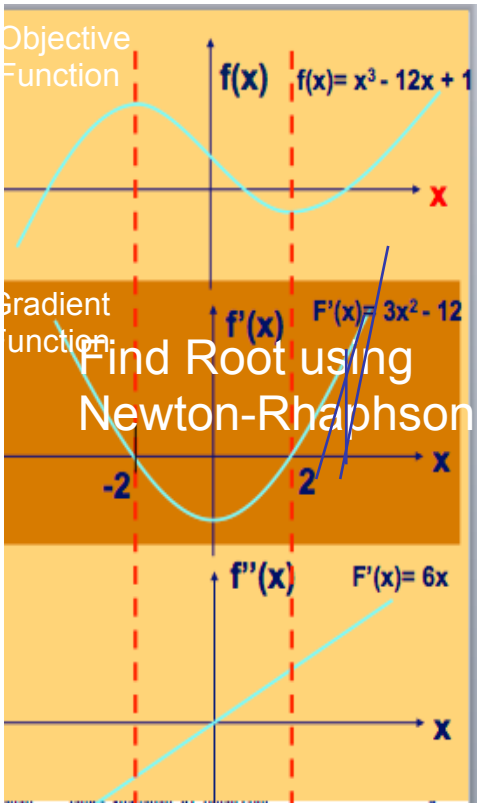
- **Whether or not regularization (discussed in a separate section) is used, it is usually not possible to find a closed-form solution; instead, an iterative numerical method must be used**
  - Gradient Descent
  - Newton-Rhapson
  - iteratively reweighted least squares (IRLS) or,
  - more commonly these days, a quasi-Newton method such as the L-BFGS method.
- **Bayesian Logistic Regression**
  - Use a general approximation method such as the Metropolis–Hastings algorithm.
  - E.g. in Spark see, http://blog.cloudera.com/blog/2014/08/bayesian-machine-learning-on-apache-spark/

Gradient Descent (a simpler root finder)

GIVEN: Minimize f(x)

STEP 1: Find the zeros of the gradient function f'(x)

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \left[H(x^i)\right]^{-1} J(x^i) \qquad \text{Multivariate Case}$$

Newton-Raphson

Calculating f''(x), the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)

- **Convergence criteria. E.g., decision vector X does not change that much or error does not change**

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

Find best W via Maximum Likelihood

$$P(y \mid X;W) = p^{y}(1-p)^{1-y} \quad \text{where p} = \Pr(y^{i} = 1 \mid X^{i};W)$$

$$L(W) = \Pr(Y \mid X;W) \quad \text{for all m training examples}$$

More compactly for each example

$$= \prod_{i=1}^{m} \Pr(y^{i} \mid X^{i};W)$$

$$= \prod_{i=1}^{m} p^{y^{i}}(1-p)^{1-y^{i}} \quad \text{where p} = \Pr(y^{i} = 1 \mid X^{i};W)$$

$$\text{l(W)} = \log(\text{L(W)})$$

$$= \sum^{m} y^{i} \log p + (1-y^{i}) \log(1-p)$$

Notice $\text{p} = 1/(1+\exp(-W^{T}X))$

$\frac{\partial p}{\partial W} = p(1-p)$ logistic function is super diff nice

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

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

See Tom Mitchell extra book chapter

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

Gradient for logistic regression

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

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

In matrix form, we write

$$\frac{\partial L(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i; \beta)) \; .$$

To solve the set of $p + 1$ nonlinear equations $\frac{\partial L(\beta)}{\partial \beta_{1j}} = 0$, $j = 0, 1, ..., p$, use the Newton-Raphson algorithm.

The Newton-Raphson algorithm requires the second-derivatives or Hessian matrix:

$$\frac{\partial^2 L(\beta)}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{N} x_i x_i^T p(x_i; \beta)(1 - p(x_i; \beta)) \; .$$

Second-order optimization

- First-order methods do not use the Hessian, and do not model the curvature of the space. Hence they can be slow to converge.
- **Second-order** optimization methods make use of the Hessian, in one form or another, and converge much faster.
- However, storing the full Hessian takes $O(D^2)$ space, and inverting it can take $O(D^3)$ time, so the overall computation time for second-order methods may be higher than for first-order methods, depending on the cost of evaluating the objective function and its gradient.

Second-order optimization

- Newtons Algorithm
- Iteratively reweighted least squares (IRLS)

- Quasi-Newton (variable metric) methods
- BFGS
- L-BFGS

Newton's algorithm

- The most basic second-order optimization algorithm is **Newton's algorithm**, which consists of updates of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_K^{-1}\mathbf{g}_k$$

- This algorithm can be derived as follows. Consider making a second-order Taylor series approximation of $f(\boldsymbol{\theta})$ around $\boldsymbol{\theta}_k$:

$$f_{quad}(\boldsymbol{\theta}) = f_k + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T\mathbf{H}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Let us rewrite this as

$$f_{quad}(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{A}\boldsymbol{\theta} + \mathbf{b}^T\boldsymbol{\theta} + c$$

where

$$\mathbf{A} = \frac{1}{2}\mathbf{H}_k, \;\; \mathbf{b} = \mathbf{g}_k - \mathbf{H}_k\boldsymbol{\theta}_k, \;\; c = f_k - \mathbf{g}_k^T\boldsymbol{\theta}_k + \frac{1}{2}\boldsymbol{\theta}_k^T\mathbf{H}_k\boldsymbol{\theta}_k$$

- The iteration can be expressed compactly in matrix form.
  - Let $\mathbf{y}$ be the column vector of y_i.
  - Let $\mathbf{X}$ be the $N \times (p+1)$ input matrix.
  - Let $\mathbf{p}$ be the N-vector of fitted probabilities with ith elemer $p(x_i; \beta^{old})$.
  - Let $\mathbf{W}$ be an $N \times N$ diagonal matrix of weights with ith element $p(x_i; \beta^{old})(1 - p(x_i; \beta^{old}))$.
  - Then

$$\frac{\partial L(\beta)}{\partial \beta} = \mathbf{X}^T(\mathbf{y} - \mathbf{p})$$
$$\frac{\partial^2 L(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T\mathbf{W}\mathbf{X} .$$

Pseudo Code

1. $0 \rightarrow \beta$
2. Compute $\mathbf{y}$ by setting its elements to
$$y_i = \begin{cases} 1 & \text{if } g_i = 1 \\ 0 & \text{if } g_i = 2 \end{cases} ,$$
$i = 1, 2, ..., N$.
3. Compute $\mathbf{p}$ by setting its elements to
$$p(x_i; \beta) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \quad i = 1, 2, ..., N.$$
4. Compute the diagonal matrix $\mathbf{W}$. The ith diagonal element is $p(x_i; \beta)(1 - p(x_i; \beta))$, $i = 1, 2, ..., N$.
5. $\mathbf{z} \leftarrow \mathbf{X}\beta + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$.
6. $\beta \leftarrow (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z}$.
7. If the stopping criteria is met, stop; otherwise go back to step 3.

- The Newton-Raphson step is

$$\begin{aligned} \beta^{new} &= \beta^{old} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \mathbf{p}) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}(\mathbf{X}\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z} , \end{aligned}$$

  where $\mathbf{z} \triangleq \mathbf{X}\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$.
- If $\mathbf{z}$ is viewed as a response and $\mathbf{X}$ is the input matrix, β^{new} is the solution to a weighted least square problem:

$$\beta^{new} \leftarrow \arg\min_{\beta} (\mathbf{z} - \mathbf{X}\beta)^T\mathbf{W}(\mathbf{z} - \mathbf{X}\beta) .$$

- Recall that linear regression by least square is to solve

$$\arg\min_{\beta} (\mathbf{z} - \mathbf{X}\beta)^T(\mathbf{z} - \mathbf{X}\beta) .$$

- $\mathbf{z}$ is referred to as the *adjusted response*.
- The algorithm is referred to as *iteratively reweighted least squares* or *IRLS*.

http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/logit.pdf

- The iteration can be expressed compactly in matrix form.
  - Let $\mathbf{y}$ be the column vector of y_i.
  - Let $\mathbf{X}$ be the $N \times (p+1)$ input matrix.
  - Let $\mathbf{p}$ be the N-vector of fitted probabilities with ith elemer $p(x_i; \beta^{old})$.
  - Let $\mathbf{W}$ be an $N \times N$ diagonal matrix of weights with ith element $p(x_i; \beta^{old})(1 - p(x_i; \beta^{old}))$.
  - Then

$$\frac{\partial L(\beta)}{\partial \beta} = \mathbf{X}^T(\mathbf{y} - \mathbf{p})$$
$$\frac{\partial^2 L(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T\mathbf{W}\mathbf{X} .$$

Pseudo Code

1. $0 \to \beta$
2. Compute $\mathbf{y}$ by setting its elements to
$$y_i = \begin{cases} 1 & \text{if } g_i = 1 \\ 0 & \text{if } g_i = 2 \end{cases} ,$$
$i = 1, 2, ..., N$.
3. Compute $\mathbf{p}$ by setting its elements to
$$p(x_i; \beta) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \quad i = 1, 2, ..., N.$$
4. Compute the diagonal matrix $\mathbf{W}$. The ith diagonal element is $p(x_i; \beta)(1 - p(x_i; \beta))$, $i = 1, 2, ..., N$.
5. $\mathbf{z} \leftarrow \mathbf{X}\beta + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$.
6. $\beta \leftarrow (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z}$.
7. If the stopping criteria is met, stop; otherwise go back to step 3.

- The Newton-Raphson Hessian Gradient

$$\begin{aligned} \beta^{new} &= \beta^{old} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \mathbf{p}) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}(\mathbf{X}\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z} , \end{aligned}$$

  where $\mathbf{z} \triangleq \mathbf{X}\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$.
- If $\mathbf{z}$ is viewed as a response and $\mathbf{X}$ is the input matrix, β^{new} is the solution to a weighted least square problem:

$$\beta^{new} \leftarrow \arg\min_{\beta} (\mathbf{z} - \mathbf{X}\beta)^T\mathbf{W}(\mathbf{z} - \mathbf{X}\beta) .$$

- Recall that linear regression by least square is to solve

$$\arg\min_{\beta} (\mathbf{z} - \mathbf{X}\beta)^T(\mathbf{z} - \mathbf{X}\beta) .$$

- $\mathbf{z}$ is referred to as the *adjusted response*.
- The algorithm is referred to as *iteratively reweighted least squares* or *IRLS*.

http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/logit.pdf

IRLS Algorithm

1 $\mathbf{w} = \mathbf{0}_D$
2 $w_0 = \log(\overline{y}/(1-\overline{y}))$
3 **repeat**
4 $\quad \eta_i = w_0 + \mathbf{w}^T \mathbf{x}_i$
5 $\quad \mu_i = \text{sigm}(\eta_i)$
6 $\quad z_i = \eta_i + \frac{y_i - \mu_i}{\mu_i(1-\mu_i)}$
7 $\quad s_i = \mu_i(1-\mu_i)$
8 $\quad \mathbf{S} = \text{diag}(s_{1:N})$
9 $\quad \mathbf{w} = (\mathbf{X}^T \mathbf{S} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S} \mathbf{z}$
10 **until** *converged*

CPSC 540

Probabilistic Machine Learning

HOME LECTURES ASSIGNMENTS PROJECT

Lectures

011/

After each lecture, you can watch the videos here.

Lecture 1: Introduction.

Lecture 2: Classification.

Lecture 3: Maximum likelihood.

Lecture 4: Linear regression.

Lecture 5: Optimization: Gradient descent, line search, stochastic gradient descent for massive datasets and streaming data.

Lecture 6: Second order methods: Newton, L-BFGS, and iterative reweighted least squares.

Lecture 7: Constrained optimization: Lagrangians and duality. Application to penalized maximum likelihood and Lasso.

Lecture 8: Bayesian learning: Priors, posterior, predictive distributions, conjugate models, and cross-validation Vs marginal likelihood.

Lecture 9: Multivariate Gaussian models.

Lecture 10: Gaussian processes.

Lecture 11: Directed probabilistic graphical models.

Lecture 12: Undirected probabilistic graphical mod

deep learning and log-linear models.

Lecture 13: Monte Carlo methods.

Lecture 14: The EM algorithm, mixtures and clustering.

http://www.cs.ubc.ca/~nando/540-2011/lectures.php

Common Implementations in Map-Reduce

- **Gradient Descent**
  - Jacobian Vector
  - First order
- **BFGS Algorithm in Map-Reduce**
  - Jacobian Vector
  - Hessian Matrix (approximation)
  - (Second Order)

- **(Hadoop and Spark)**

Logistic Regression via Gradient Descent

Let $\mathrm{W} = (0,0,....)$

Repeat

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

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

the term inside the parentheses is simply the prediction error;
pulling the W weight vector closer to the example
Batch LR: do a batch update of W after a sweep of the data

Logistic Regression using optim()

```
#-----------------------------------------------
# Logistic Regression using optim to
# maximize the likelihood
#-----------------------------------------------
Minimize the negative log likelihood as loss function (max loglikehood)
lreg.2 <- function(X, y, method='BFGS'){
    X <- cbind(1, X)
    negLogL <- function(b, X, y) {
       p = as.vector(1/(1 + exp(-X %*% b)));  #predicted probability
       - sum(y*log(p) + (1 - y)*log(1 - p))
    }
    grad <- function(b, X, y){
       p <- as.vector(1/(1 + exp(-X %*% b)))
       - colSums((y - p)*X)
    }
     result <- optim(rep(0, ncol(X)), negLogL, gr=grad,
           hessian=TRUE, method=method, X=X, y=y)
     list(coefficients=result$par, var=solve(result$hessian),
       deviance=2*result$value, converged=result$convergence == 0)
}
```

See lreg.2 () in R file

Maximize Likelihood using Newton-Rhapson On a SINGLE computer: optimization package

$$\log_e L = \sum y_i \log_e p_i + (1 - y_i) \log_e (1 - p_i)$$

- Optimizers work by evaluating the *gradient* (vector of partial derivatives) of the 'objective function' (the log-likelihood) at the current estimates of the parameters, iteratively improving the parameter estimates using the information in the gradient; iteration ceases when the gradient is sufficiently close to zero.
- For the logistic-regression model, the gradient of the log-likelihood is

$$\frac{\partial \log_e L}{\partial \mathbf{b}} = \sum (y_i - p_i)\mathbf{x}_i$$

The covariance matrix of the coefficients is the inverse of the matrix of second derivatives. The matrix of second derivatives, called the *Hessian*, is

$$\frac{\partial \log_e L}{\partial \mathbf{b} \partial \mathbf{b}'} = \mathbf{X}'\mathbf{V}\mathbf{X}$$

The `optim` function in R, however, calculates the Hessian numerically (rather than using an analytic formula).

Estimate Weights using Newton-Raphson

The *Newton-Raphson method* is a common iterative approach to estimating a logistic-regression model:

1. Choose initial estimates of the regression coefficients, such as $\mathbf{b}_0 = \mathbf{0}$.
2. At each iteration t, update the coefficients:

$$\mathbf{b}_t = \mathbf{b}_{t-1} + (\mathbf{X}'\mathbf{V}_{t-1}\mathbf{X})^{-1}\mathbf{X}'(\mathbf{y} - \mathbf{p}_{t-1})$$

   where
   - $\mathbf{X}$ is the model matrix, with $\mathbf{x}_i'$ as its ith row;
   - $\mathbf{y}$ is the response vector (containing 0's and 1's);
   - $\mathbf{p}_{t-1}$ is the vector of fitted response probabilities from the previous iteration, the ith entry of which is

$$p_{i,t-1} = \frac{1}{1 + \exp(-\mathbf{x}_i'\mathbf{b}_{t-1})}$$

   - $\mathbf{V}_{t-1}$ is a diagonal matrix, with diagonal entries $p_{i,t-1}(1 - p_{i,t-1})$.
3. Step 2 is repeated until $\mathbf{b}_t$ is close enough to $\mathbf{b}_{t-1}$. The estimated asymptotic covariance matrix of the coefficients is given by $(\mathbf{X}'\mathbf{V}\mathbf{X})^{-1}$

Solve a System of Equations in R

- **Example 1: Solve the system of linear equations.**

  -2x + 3y = 8
  3x - y = -5

- **multiply all terms in the second equation by 3**
  -2x + 3y = 8
  9x - 3y = -15
  add the two equations
  7x = -7

  Note: y has been eliminated, hence the name: method of elimination solve the above equation for x
  x = -1
  substitute x by -1 in the first equation
  -2(-1) + 3y = 8
  solve the above equation for y

  2 + 3y = 8
  3y = 6
  y = 2

  write the solution to the system as an ordered pair

  (-1,2)

```
➢A <- matrix(c(-2,3, 3,-1 ), 2)
> A
     [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr(A), b)
[1] -1  2
```

Using solve() to compute the inverse in R

- **#compute the inverse of t(X)VX (inverse of Hessian)**
- **var.b <- solve(t(X) %*% V %*% X) #computes inverse**
- **Normally use solve(a, b, ...) BUT …..**
  - If missing, b is taken to be an identity matrix and solve will return the inverse of a.

Logistic Regression using Newton-Raphson

```
lreg.NewtonRaphson <- function(X, y, max.iter=10, tol=1E-6){
   # X is the model matrix
   # y is the response vector of 0s and 1s
   # max.iter is the maximum number of iterations
   # tol is a convergence criterion
   X <- cbind(1, X)  # add constant
   b <- b.last <- rep(0, ncol(X))  # initialize weight vector
   it <- 1  # iteration index
   while (it <= max.iter){
      p <- as.vector(1/(1 + exp(-X %*% b)))
      V <- diag(p * (1 - p))
      var.b <- solve(t(X) %*% V %*% X)  #compute the inverse of t(X)VX
      b <- b + var.b %*% t(X) %*% (y - p)
      if (max(abs(b - b.last)/(abs(b.last) + 0.01*tol)) < tol)
         break
      b.last <- b
      it <- it + 1  # increment index
      }
   if (it > max.iter) warning('maximum iterations exceeded')
   list(coefficients=as.vector(b), var=var.b, iterations=it)
}
```

See lreg.* () in R file

predicted

Update the weights

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

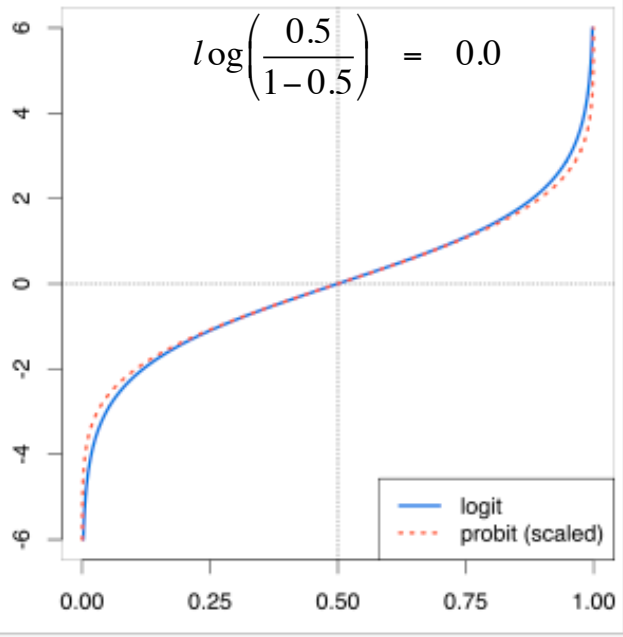
$$\log\left(\frac{p}{1-p}\right) = W^T X$$

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

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

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

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

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

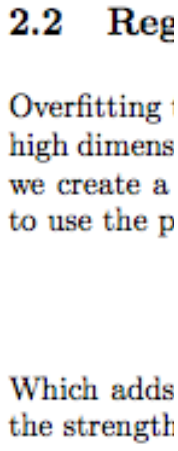
Plug in equation (27), we can derive

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

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

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 extensio 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 Reg

Overfitting
high dimens
we create a
to use the p

Which adds
the strength

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/

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

Metrics are key

- Metrics are key
- Diagnostics

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

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

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

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

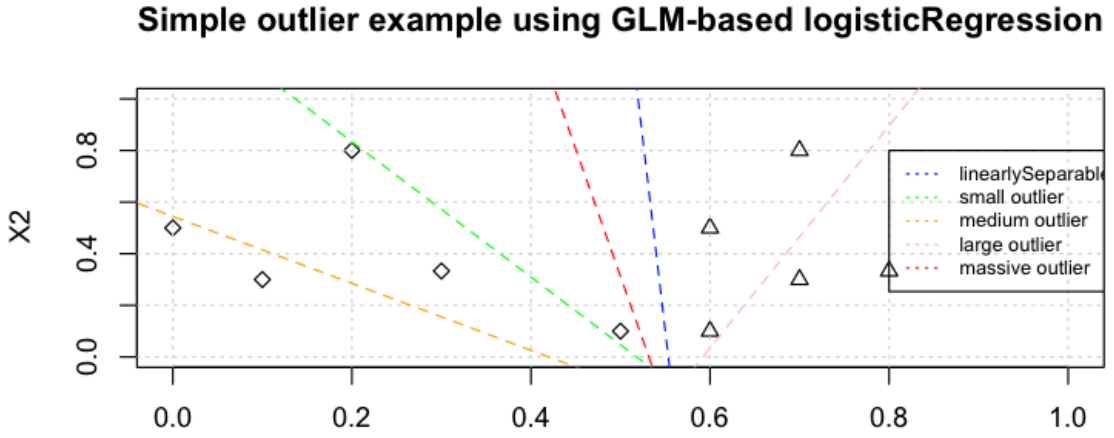
simplifies to

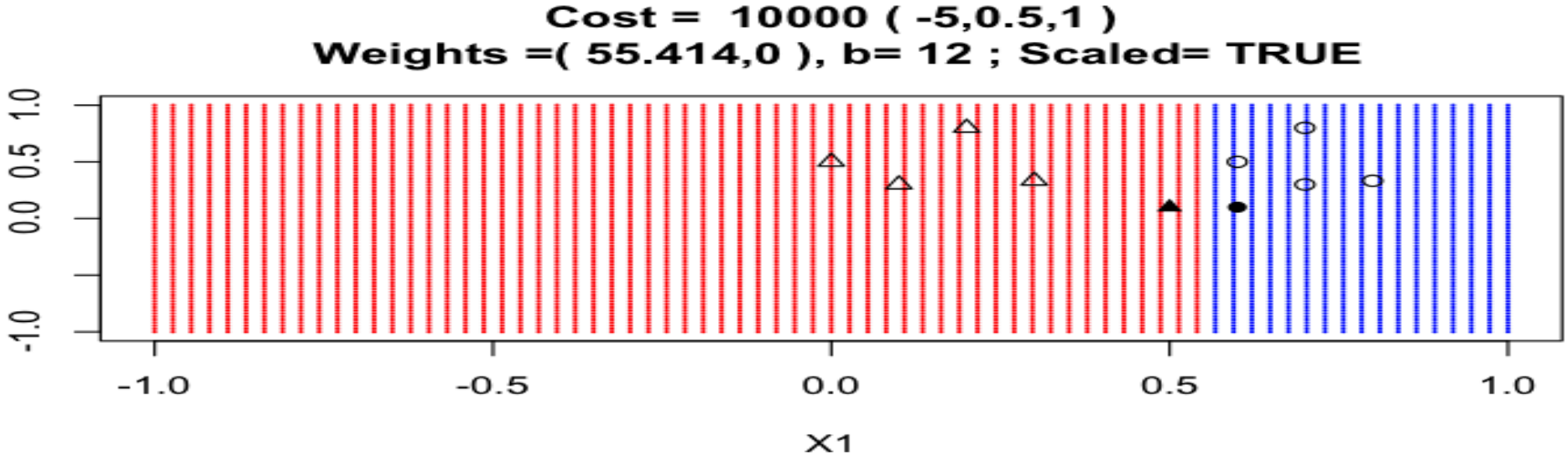
$y = 0$ if $1 < \exp(W^T X^i)$

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**

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

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

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

vs

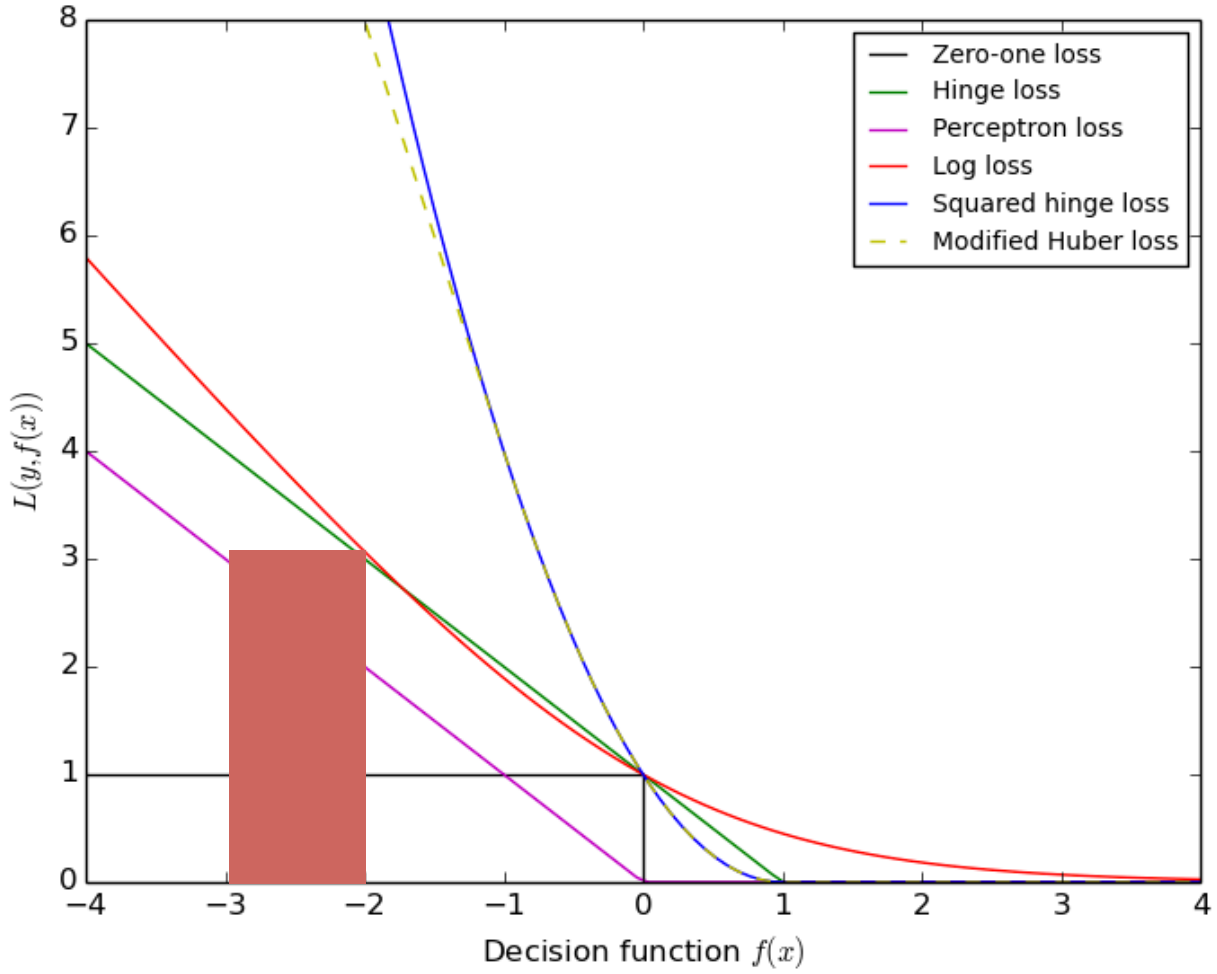
-1 * log(0.9)
-1*-0.1 = 0.1 Error

- **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

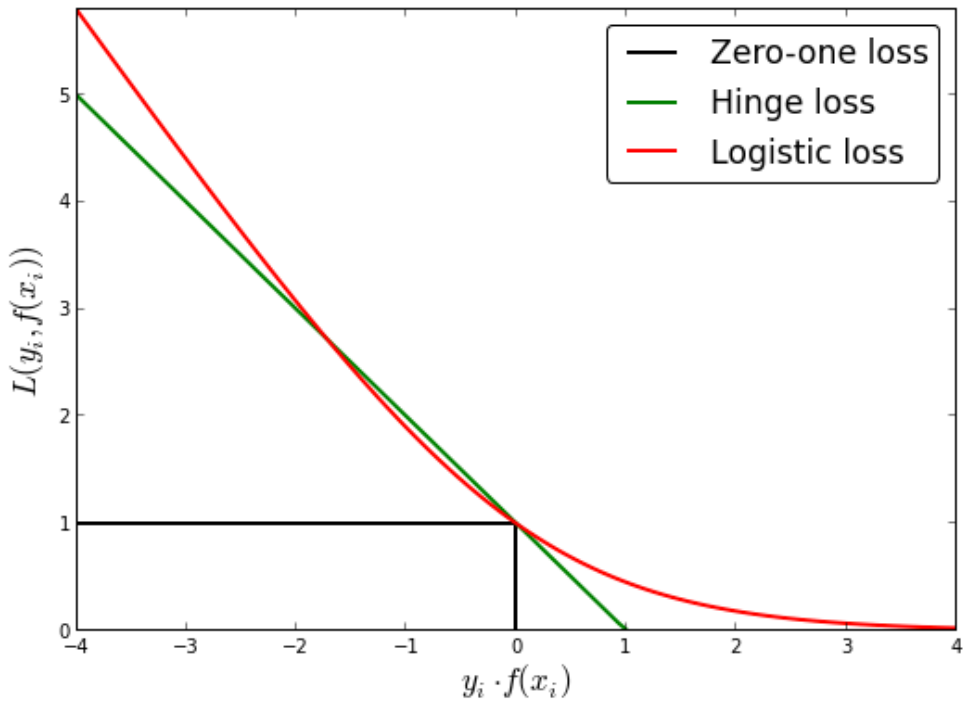
regression loss function

In machine learning it is common to formulate the classification task as a 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

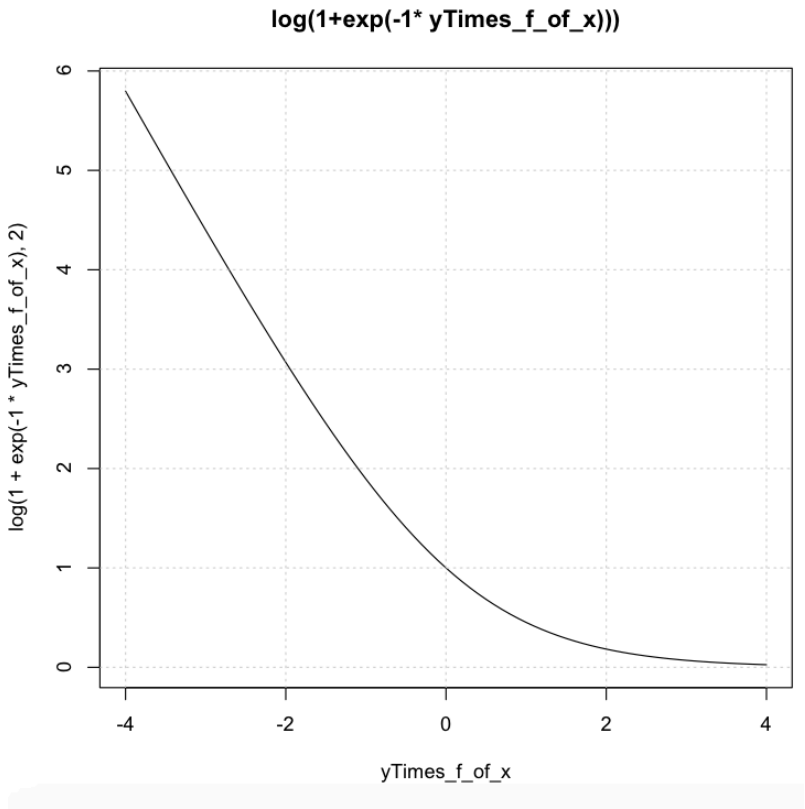
$$L(x, y) = \sum_{i}^{n} \text{loss}(f(x_i), y_i)$$

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, $\log(1 + \exp f(x_i)y_i)$



Log loss function (base 2)



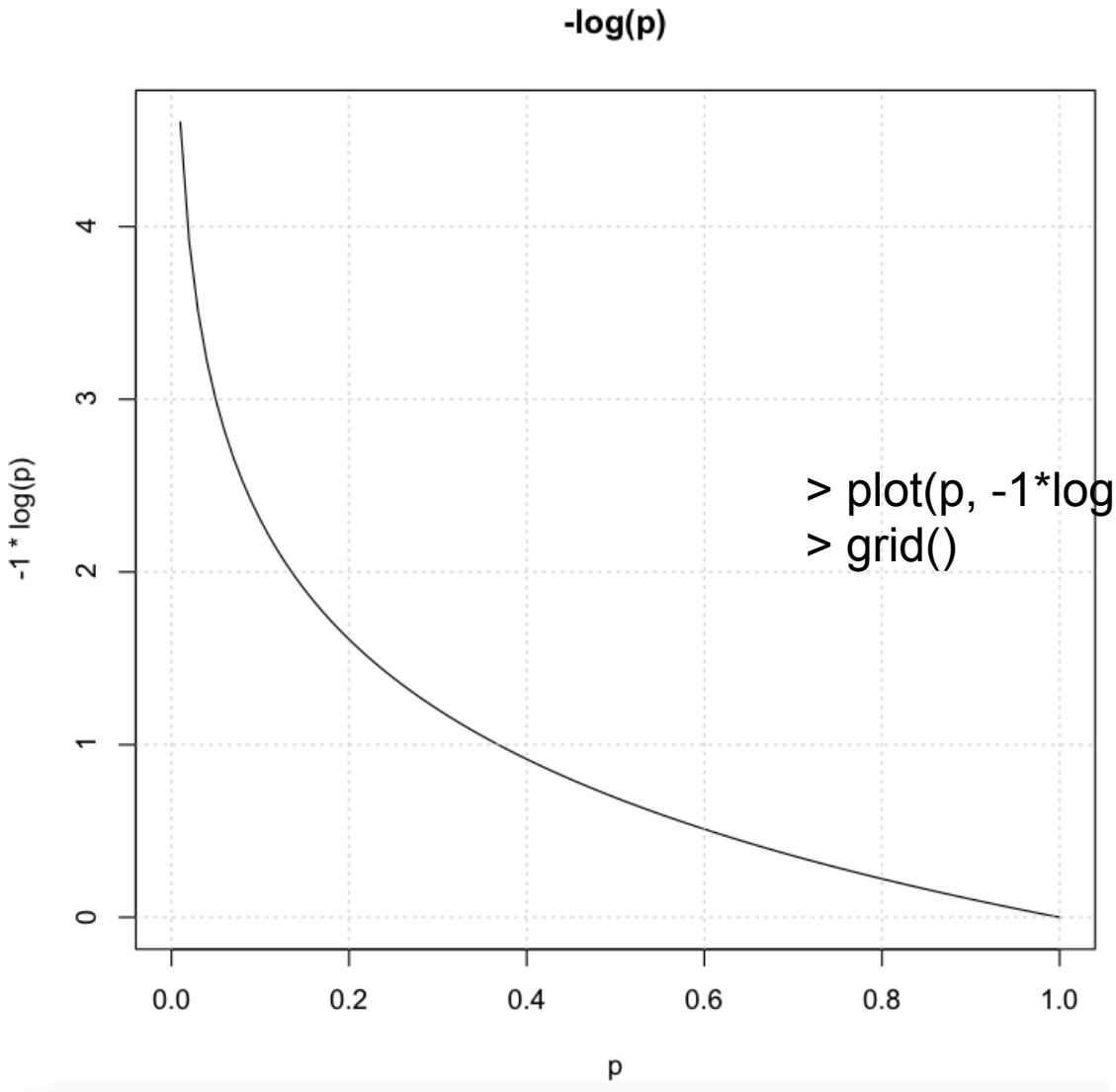
Convex so the loss function is also convext (sum of convex functions)

```
> 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()
```

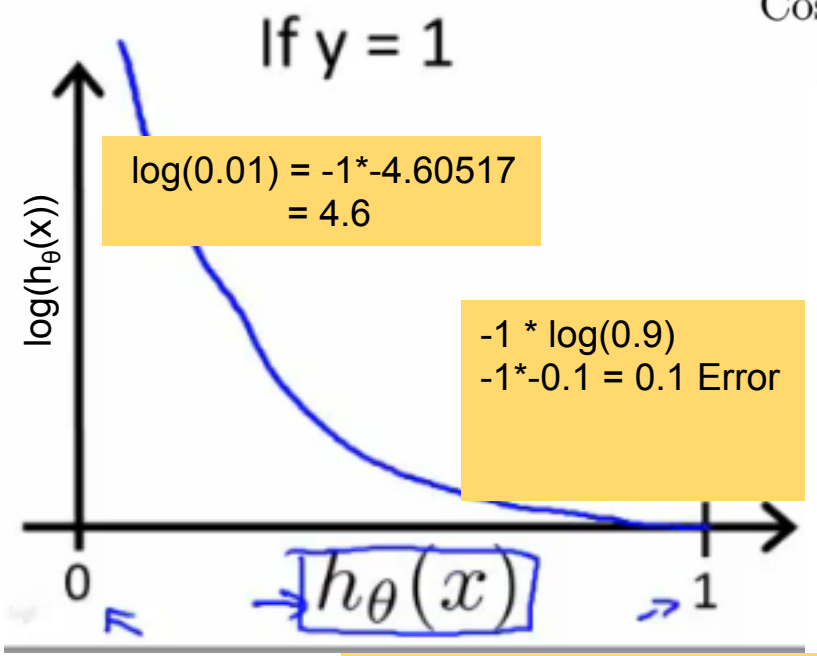
A convex logistic regression cost function

Regularization Loss

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

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

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



log(0.01) = -1*-4.60517
= 4.6

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

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

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

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

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

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

- **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

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

- **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

Logistic Regression in R

- **Explore Logistic Regression in R**
  - Using Newton-Raphson
  - Using general optimization
  - Using GLM built-in function
  - Using Gradient Descent (homework)

- **Book: John Fox (2002), Sage,**
  An R and S-PLUS Companion to Applied Regression
  - http://socserv.mcmaster.ca/jfox/Courses/R-course/
- **Accessing man pages in R**
  - ?glm
  - ?solve
  - help.search("solve system") in R

LR: Maximum Likelihood Estimates

- **Because logistic regression predicts probabilities, rather than just classes, we can fit it using likelihood.**
  - For each training data-point, we have a vector of features, X_i, and an observed class, y_i. The probability of that class was either p, if $y_i = 1$, or $1 - p$, if $y_i = 0$. The likelihood of the model (W,b) of generating the training data is then
- **where $W = \langle w_0, w_1 \ldots, w_n \rangle$ is the vector of parameters to be estimated, Y^l denotes the observed value of Y in the l^{th} training example and *X^l denotes the observed* value of *X in the l^{th} training example***

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

- **The expression to the right of the *argmax* is the conditional data likelihood**

$$L(\vec{w}, b) = \prod_{i=1}^{n} p(\vec{x}_i)^{y_i} (1 - p(\vec{x}_i)^{1-y_i}$$

Find best W via Maximum Likelihood

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

$$\Pr(y^i = 0 \mid X^i; W) = 1 - \frac{1}{1 + \exp(-W^T X^i)} = \frac{\exp(-W^T X^i)}{1 + \exp(-W^T X^i)}$$

More compactly for each example

$$P(y \mid X; W) = p^y (1 - p)^{1-y} \quad \text{where } p = \Pr(y^i = 1 \mid X^i; W)$$

LR: Maximum Likelihood Estimates

- **The expression to the right of the *argmax* is the conditional data likelihood.**

$$W \leftarrow \arg\max_{W} \prod_{l} P(Y^l | X^l, W)$$

Select W s:t likelihood of W generating the data is maximized

$$L(\vec{w}, b) = \prod_{i=1}^{n} p(\vec{x}_i)^{y_i} (1 - p(\vec{x}_i)^{1-y_i}$$

Y can take only values 0 or 1, so only one of the two terms in the expression will be non-zero for any given Y^l; *recall m^0 =1 .*

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

Working with logs is simpler and more effective computationally; amenable to off-the-shelf optimization approaches; concave function in W so gradient ascent will converge to global maximum (though many may exist). L(W) continuous, differentiable

Conditional Data Likelihood

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

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

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l=1|X^l,W) + (1-Y^l)\ln P(Y^l=0|X^l,W) \\ &= \sum_l Y^l \ln \frac{P(Y^l=1|X^l,W)}{P(Y^l=0|X^l,W)} + \ln P(Y^l=0|X^l,W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1+\exp(w_0+\sum_i^n w_i X_i^l)) \end{aligned}$$

Estimating Parameters using Gradient Descent

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

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

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

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

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

Find best W via Maximum Likelihood

$$P(y \mid X;W) = p^{y}(1-p)^{1-y} \quad \text{where p} = \Pr(y^{i} = 1 \mid X^{i};W)$$

More compactly for each example

$$L(W) = \Pr(Y \mid X;W) \quad \text{for all m training examples}$$

$$= \prod_{i=1}^{m} \Pr(y^{i} \mid X^{i};W)$$

$$= \prod_{i=1}^{m} p^{y^{i}}(1-p)^{1-y^{i}} \quad \text{where p} = \Pr(y^{i} = 1 \mid X^{i};W)$$

$$\text{l(W)} = \log(\text{L(W)})$$

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

=

Derive Gradient….in notes; must finish

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

$$= \prod_{i=1}^{m} \Pr(y^i \mid X^i; W)$$

$$= \prod_{i=1}^{m} p^{y^i} (1 - p)^{1 - y^i} \qquad \text{where p} = \Pr(y^i = 1 \mid X^i; W)$$

$$\text{l(W)} = \log(\text{L(W)})$$

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

$$\text{Notice z} = W^T X$$

$$\frac{\partial \text{logit}(W^T X)}{\partial W} = \frac{\partial \left(\frac{1}{1 + \exp(-W^T X)} \right)}{\partial W}$$

$$= \text{logit}(W^T X)\left(1 - \text{logit}(W^T X)\right)$$

Derive Gradient….in notes; must finish 2/2

- **W=**

See Tom Mitchell extra book chapter

$$
\begin{aligned}
l(W) &= \sum_{i=1}^{m} y^i p - (1 - y^i)(1 - p) \\
&= \\
&= \\
l(W) &= \log(L(W)) \\
&= \sum_{i=1}^{m} y^i p - (1 - y^i)(1 - p)
\end{aligned}
$$

Logistic Regression via Gradient Descent

Stochastic Gradient Descent

- **Stochastic update**

$\nabla_{wj} l(W_t)$
Partial derivate WRT to variable w_j of error function l(W) at point W_t

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

Repeat

For j in 0..n # each variable

For i in 1..m # each example

$$W_{j,t+1} = W_{j,t} + \alpha * \nabla_W l(W)$$

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

Logistic Regression via Gradient Descent

Stochastic Gradient Descent

- **Stochastic update**

$Let\ \ \mathrm{W} = (0,0,....)$

Repeat

$\nabla_{wj}\, l(W_t)$
Partial derivate WRT to variable w_j of error function l(W) at point W_t

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 term inside the parentheses is simply the prediction error; pulling the W weight vector closer to the example
Batch LR: do a batch update of W_j after a sweep of the data

Logistic Regression has a global optimum

- **Good news:** ***l(w) is concave function of w; so local optimum is a global optimum!!!***
- **Bad news: no closed-form solution to maximize** ***l(w); gradient equations are non-linear***
- **Good news: concave functions easy to optimize**

Maximize Likelihood using Newton-R

$$\log_e L = \sum y_i \log_e p_i + (1 - y_i) \log_e (1 - p_i)$$

- Optimizers work by evaluating the *gradient* (vector of partial derivatives) of the 'objective function' (the log-likelihood) at the current estimates of the parameters, iteratively improving the parameter estimates using the information in the gradient; iteration ceases when the gradient is sufficiently close to zero.
- For the logistic-regression model, the gradient of the log-likelihood is

$$\frac{\partial \log_e L}{\partial \mathbf{b}} = \sum (y_i - p_i)\mathbf{x}_i$$

The covariance matrix of the coefficients is the inverse of the matrix of second derivatives. The matrix of second derivatives, called the *Hessian*, is

$$\frac{\partial \log_e L}{\partial \mathbf{b} \partial \mathbf{b}'} = \mathbf{X}'\mathbf{V}\mathbf{X}$$

The `optim` function in R, however, calculates the Hessian numerically (rather than using an analytic formula).

Estimate Weights using Newton-Raphson

The *Newton-Raphson method* is a common iterative approach to estimating a logistic-regression model:

1. Choose initial estimates of the regression coefficients, such as $\mathbf{b}_0 = \mathbf{0}$.
2. At each iteration t, update the coefficients:

$$\mathbf{b}_t = \mathbf{b}_{t-1} + (\mathbf{X}'\mathbf{V}_{t-1}\mathbf{X})^{-1}\mathbf{X}'(\mathbf{y} - \mathbf{p}_{t-1})$$

   where
   - $\mathbf{X}$ is the model matrix, with $\mathbf{x}_i'$ as its ith row;
   - $\mathbf{y}$ is the response vector (containing 0's and 1's);
   - $\mathbf{p}_{t-1}$ is the vector of fitted response probabilities from the previous iteration, the ith entry of which is

$$p_{i,t-1} = \frac{1}{1 + \exp(-\mathbf{x}_i'\mathbf{b}_{t-1})}$$

   - $\mathbf{V}_{t-1}$ is a diagonal matrix, with diagonal entries $p_{i,t-1}(1 - p_{i,t-1})$.
3. Step 2 is repeated until $\mathbf{b}_t$ is close enough to $\mathbf{b}_{t-1}$. The estimated asymptotic covariance matrix of the coefficients is given by $(\mathbf{X}'\mathbf{V}\mathbf{X})^{-1}$

Solve a System of Equations in R

- **Example 1: Solve the system of linear equations.**

  -2x + 3y = 8
  3x - y = -5

- **multiply all terms in the second equation by 3**
  -2x + 3y = 8
  9x - 3y = -15
  add the two equations
  7x = -7

  Note: y has been eliminated, hence the name: elimination solve the above equation for x
  x = -1
  substitute x by -1 in the first equation
  -2(-1) + 3y = 8
  solve the above equation for y

  2 + 3y = 8
  3y = 6

```
➢A <- matrix(c(-2,3, 3,-1 ), 2)
> A
     [,1] [,2]
[1,]   -2    3
[2,]    3   -1
> b
[1] 8 5
> b=c(8,-5)
> qr.solve(A, b) # or solve(qr(A
[1] -1  2
```

Using solve() to compute the inverse in R

- **#compute the inverse of t(X)VX (inverse of Hessian)**
- **var.b <- solve(t(X) %*% V %*% X) #computes inverse**
- **Normally use solve(a, b, ...) BUT …..**
  - If missing, b is taken to be an identity matrix and solve will return the inverse of a.

Logistic Regression using Newton-Raphson

```
lreg.NewtonRaphson <- function(X, y, max.iter=10, tol=1E-6){
   # X is the model matrix
   # y is the response vector of 0s and 1s
   # max.iter is the maximum number of iterations
   # tol is a convergence criterion
   X <- cbind(1, X)  # add constant
   b <- b.last <- rep(0, ncol(X))  # initialize weight vector
   it <- 1  # iteration index
   while (it <= max.iter){
      p <- as.vector(1/(1 + exp(-X %*% b)))      predicted
      V <- diag(p * (1 - p))
      var.b <- solve(t(X) %*% V %*% X)  #compute the inverse of t(X)VX
      b <- b + var.b %*% t(X) %*% (y - p)          Update the weights
      if (max(abs(b - b.last)/(abs(b.last) + 0.01*tol)) < tol)
         break
      b.last <- b
      it <- it + 1  # increment index
      }
   if (it > max.iter) warning('maximum iterations exceeded')
   list(coefficients=as.vector(b), var=var.b, iterations=it)
```

Negative Log Likelihood of Learnt Model W

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

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

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

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

Feature Significance

Residual Deviance -2l(W)

AIC

Deviance of Null Model W

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

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

- **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

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

- **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

Assessing Logistic Regression Models

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

Logistic Regression in R

- **Explore Logistic Regression in R**
  - Using Newton-Raphson
  - Using general optimization
  - Using GLM built-in function
  - Using Gradient Descent (homework)

- **Book: John Fox (2002), Sage,**
  An R and S-PLUS Companion to Applied Regression
  - http://socserv.mcmaster.ca/jfox/Courses/R-course/
- **Accessing man pages in R**
  - ?solve
  - help.search("solve system") in R

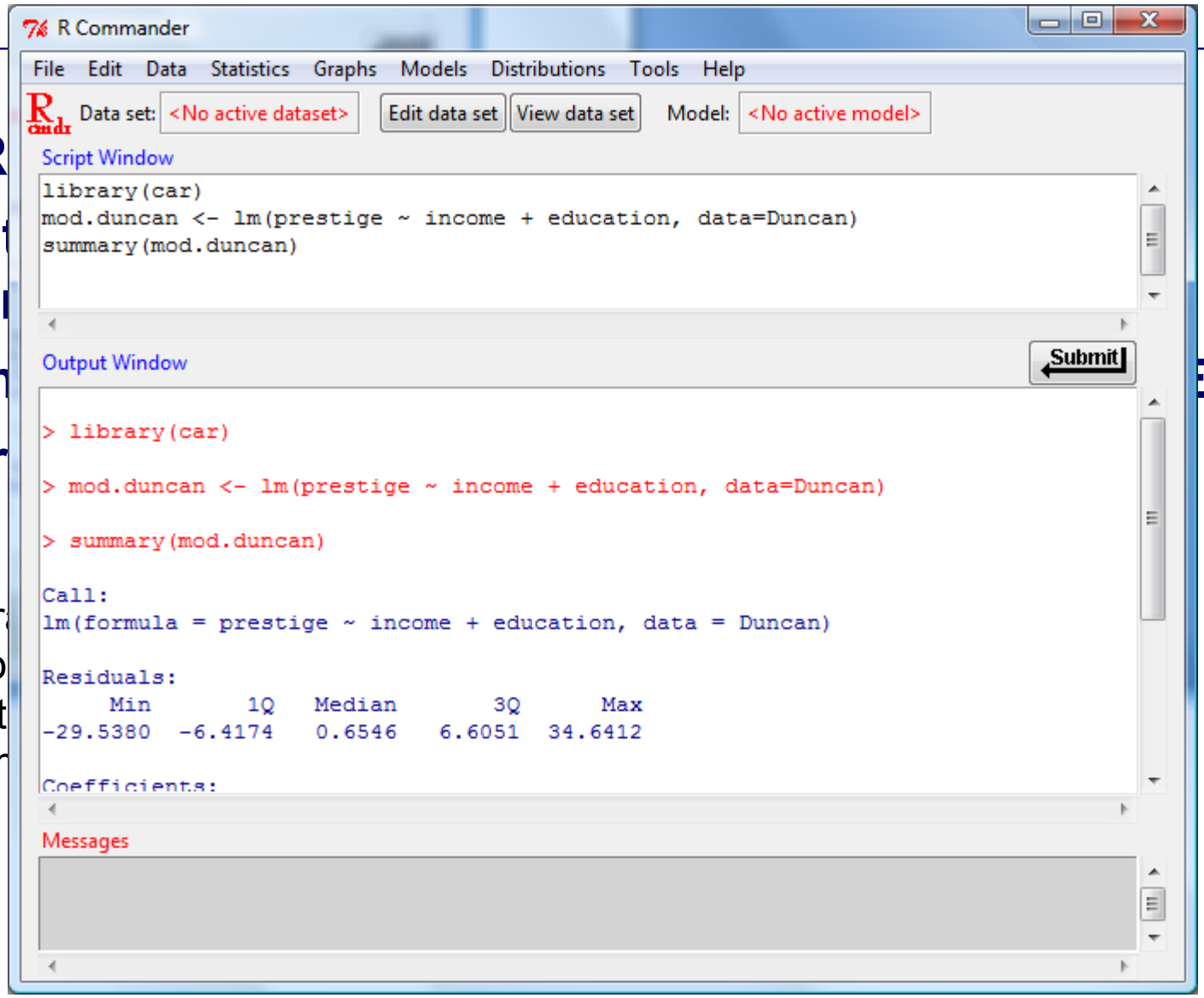
Rcmdr: a tool for demos and teaching

```
# Rcmdr
# http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/
    installation-notes.html
# install.packages("Rcmdr", dependencies=TRUE)
library(Rcmdr)
```

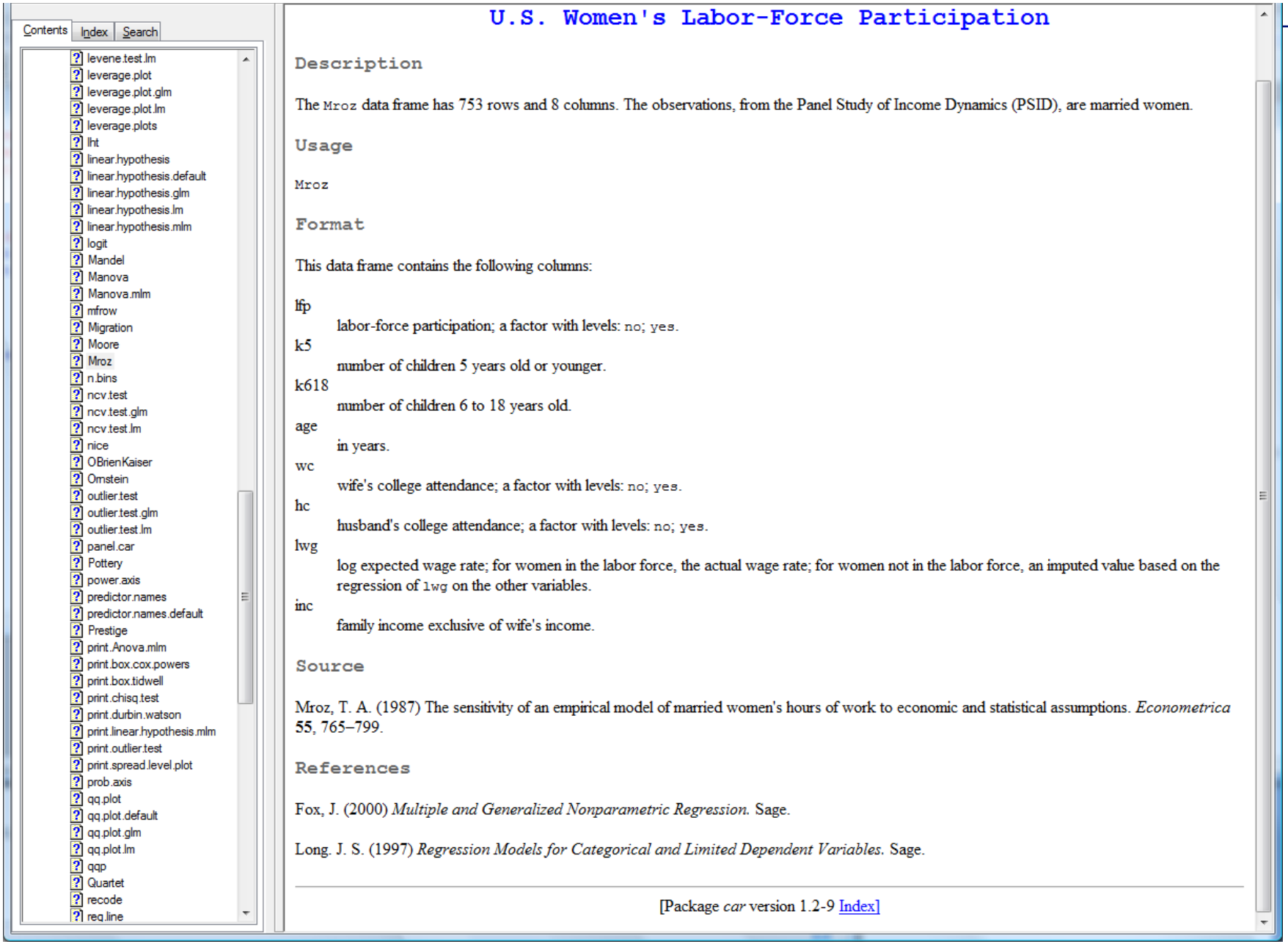
```
library(car)
mod.duncan <- lm(prestige ~ income + education,
data=Duncan)
summary(mod.duncan)
```

R
ht
i
in
libr

libr
mo
dat
sur



Mroz Dataset in R



U.S. Women's Labor-Force Participation

Description

The `Mroz` data frame has 753 rows and 8 columns. The observations, from the Panel Study of Income Dynamics (PSID), are married women.

Usage

```
Mroz
```

Format

This data frame contains the following columns:

lfp
: labor-force participation; a factor with levels: `no`; `yes`.

k5
: number of children 5 years old or younger.

k618
: number of children 6 to 18 years old.

age
: in years.

wc
: wife's college attendance; a factor with levels: `no`; `yes`.

hc
: husband's college attendance; a factor with levels: `no`; `yes`.

lwg
: log expected wage rate; for women in the labor force, the actual wage rate; for women not in the labor force, an imputed value based on the regression of `lwg` on the other variables.

inc
: family income exclusive of wife's income.

Source

Mroz, T. A. (1987) The sensitivity of an empirical model of married women's hours of work to economic and statistical assumptions. *Econometrica* **55**, 765–799.

References

Fox, J. (2000) *Multiple and Generalized Nonparametric Regression.* Sage.

Long. J. S. (1997) *Regression Models for Categorical and Limited Dependent Variables.* Sage.

[Package *car* version 1.2-9 Index]

Mroz Dataset

Description

The `Mroz` data frame has 753 rows and 8 columns. The observations, from the Panel Study of Income Dynamics (PSID), are married women.

Usage

```
Mroz
```

Format

This data frame contains the following columns:

lfp
: labor-force participation; a factor with levels: `no`; `yes`.

k5
: number of children 5 years old or younger.

k618
: number of children 6 to 18 years old.

age
: in years.

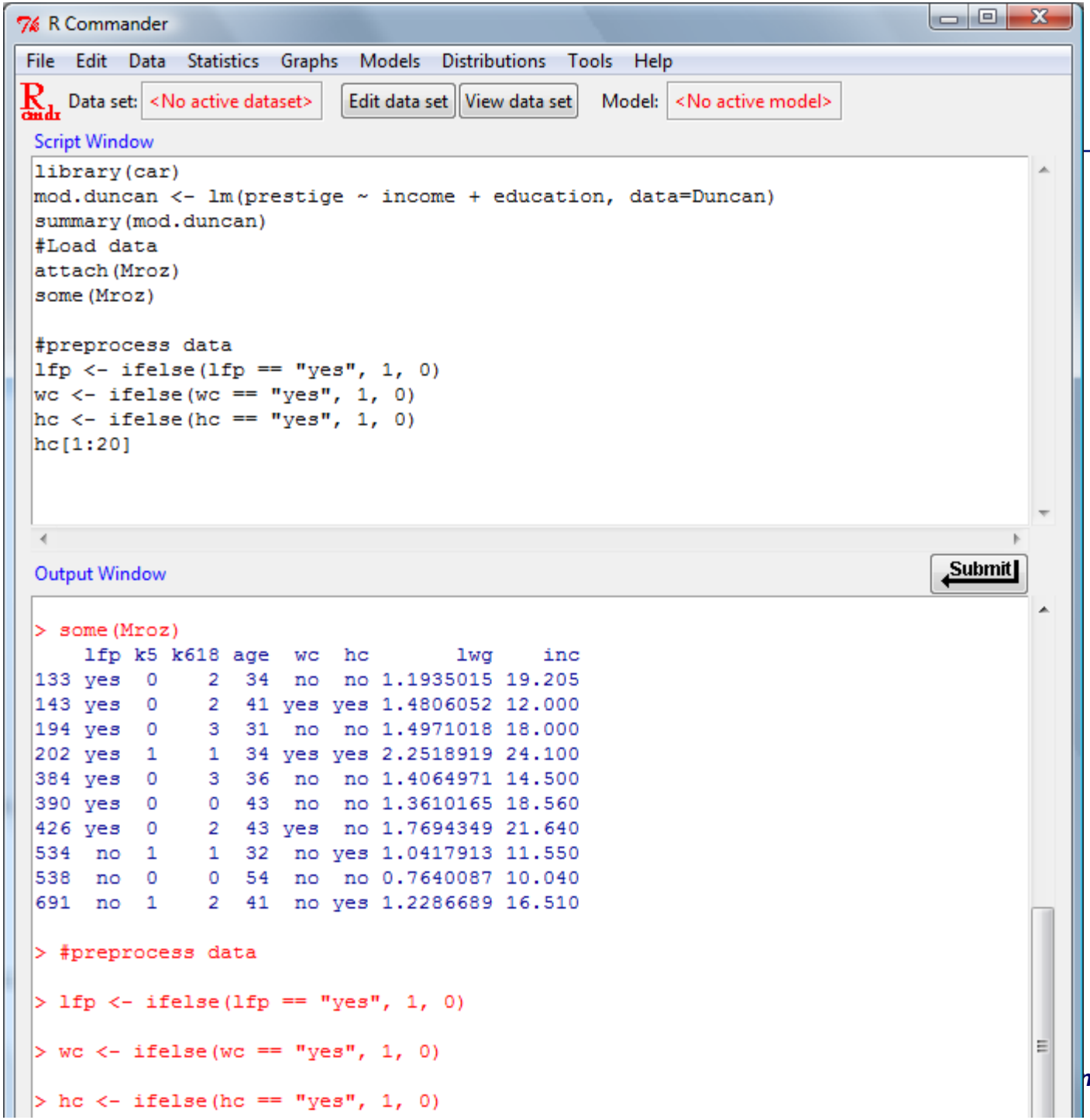
wc
: wife's college attendance; a factor with levels: `no`; `yes`.

hc
: husband's college attendance; a factor with levels: `no`; `yes`.

lwg
: log expected wage rate; for women in the labor force, the actual wage rate; for women not in the labor force, an imputed value based on the regression of `lwg` on the other variables.

inc
: family income exclusive of wife's income.

Preprocess Data



Logistic Regression using optim()

```
#-----------------------------------------------
# Logistic Regression using optim to
# maximize the likelihood
#-----------------------------------------------
Minimize the negative log likelihood as loss function (max loglikehood)
lreg.2 <- function(X, y, method='BFGS'){
    X <- cbind(1, X)
    negLogL <- function(b, X, y) {
       p = as.vector(1/(1 + exp(-X %*% b)));  #predicted probability
       - sum(y*log(p) + (1 - y)*log(1 - p))
     }
    grad <- function(b, X, y){
       p <- as.vector(1/(1 + exp(-X %*% b)))
       - colSums((y - p)*X)
     }
     result <- optim(rep(0, ncol(X)), negLogL, gr=grad,
           hessian=TRUE, method=method, X=X, y=y)
     list(coefficients=result$par, var=solve(result$hessian),
       deviance=2*result$value, converged=result$convergence == 0)
}
```

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

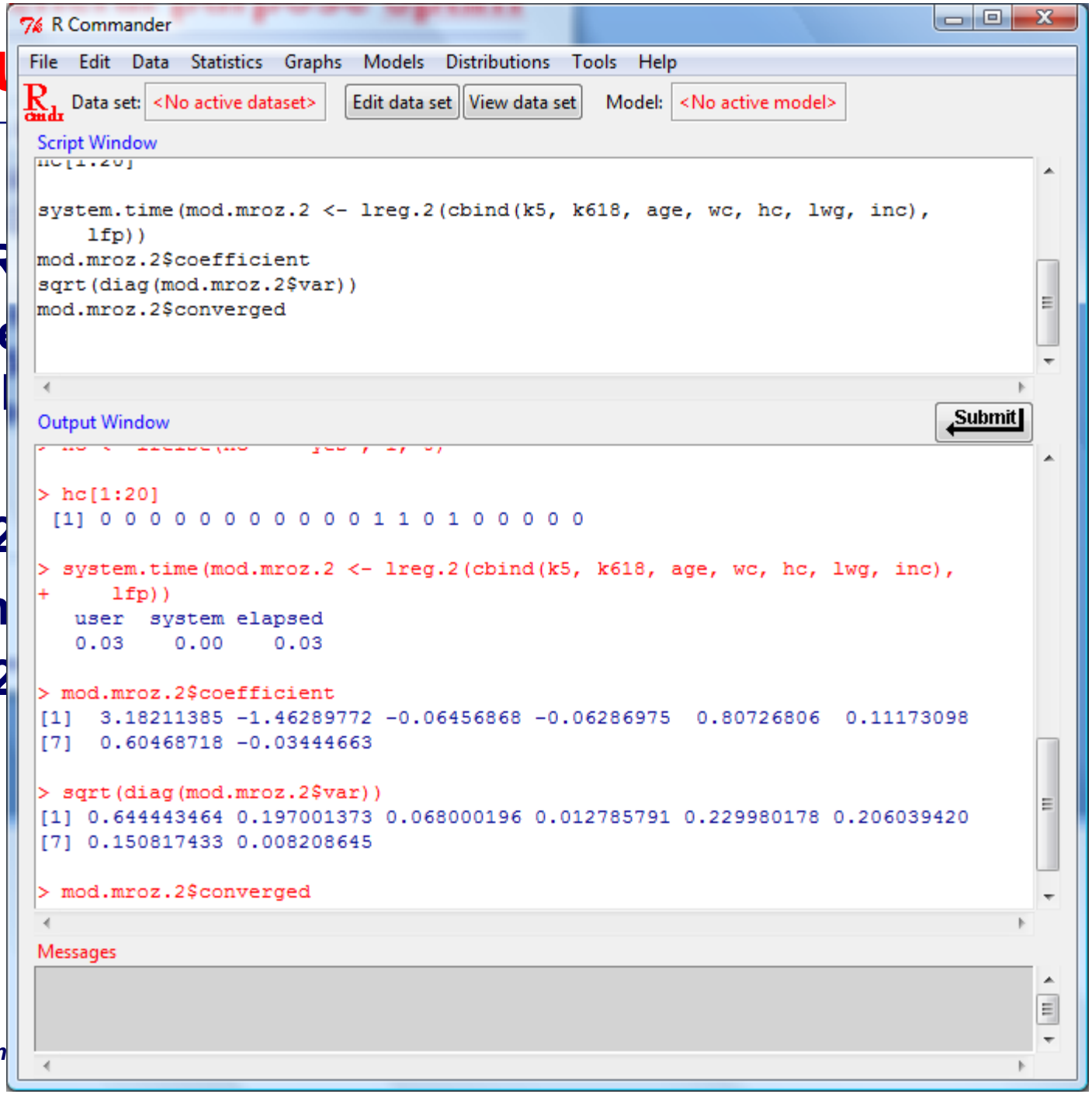
Feature Significance

Residual Deviance

AIC

LR u

```
# Logistic R
system.time
    age, wc, h
    lfp))
mod.mroz.2
sqrt(diag(m
mod.mroz.2
```



Exercise: solve LR using gradient descent

- Test it LR.GradientDescent withMroz dataset
- How many iterations does it take to convergence
- Compute the residual deviance of the learnt model
- Compare the coefficients (W's) with those learnt using glm()

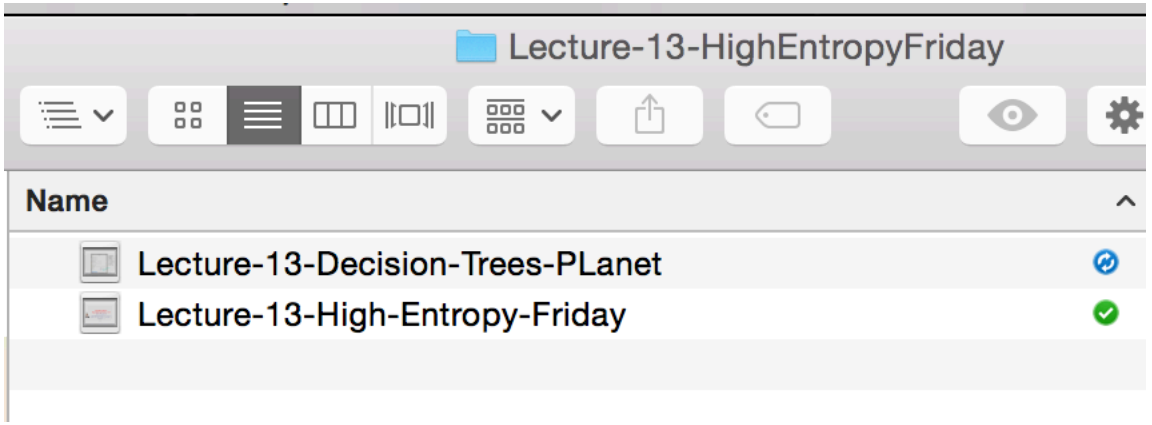
- HINTS

```
help(Mroz) #get a description of the dataset
#Load data
attach(Mroz)
#preprocess data
lfp <- ifelse(lfp == "yes", 1, 0);   wc <- ifelse(wc == "yes", 1, 0)
hc <- ifelse(hc == "yes", 1, 0)
# Logistic Regression using GLM
system.time(mod.mroz.glm <-     # check!
   glm(lfp ~ k5 + k618 + age +wc + hc + lwg + inc, family=binomial))
coefficients(mod.mroz.glm)
```

Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering (HHH)**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

Decision Trees



Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering (HHH)**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

Tutorial Outline

- **Part 1: Introduction**
  - Welcome Survey
  - Install Spark
  - Background and motivation
- **Part 2: Spark Intro and basics**
  - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
  - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, *etc.*
- **Part 4: Wrap up**
  - Spark 1.5 and beyond
  - Summary

Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks
- **Spark frameworks**
- **Spark 1.6 (2.0) and beyond**
- **Summary**

DataFrames (new API introduced in 2/2015)

- **A DataFrame is a distributed collection of data organized into named columns.**
- **It is conceptually equivalent to a table in a relational database or a data frame in R/Python, or a table in MySql, but with richer optimizations under the hood.**
- **DataFrames can be constructed from a wide array of sources such as:**
  - structured data files, tables in Hive,
  - external databases, or existing RDDs
  - JSON, Parquet and more
- **The DataFrame API is available in Scala, Java, Python, and R.**

- **The entry point into all relational functionality in Spark is the SQLContext class, or one of its decedents. To create a basic SQLContext, all you need is a SparkContext.**

- ..


Branch: master
spark / examples / src / main / resources / people.json
yhuai on Jun 17, 2014 [SPARK-2060][SQL] Querying JSON Datasets with SQL and DSL in Spark SQL
1 contributor
4 lines (3 sloc) 0.073 kB
Raw
1 {"name":"Michael"}
2 {"name":"Andy", "age":30}
3 {"name":"Justin", "age":19}

- As an example, the following creates a `DataFrame` based on the content of a JSON file:

Scala **Java** **Python** **R**

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.json("examples/src/main/resources/people.json")

# Displays the content of the DataFrame to stdout
df.show()
```

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

# Create the DataFrame
df = sqlContext.read.json("examples/src/main/resources/people.json")

# Show the content of the DataFrame
df.show()
## age  name
## null Michael
## 30   Andy
## 19   Justin

# Print the schema in a tree format
df.printSchema()
## root
## |-- age: long (nullable = true)
## |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
## name
## Michael
## Andy
## Justin
```

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
## name    (age + 1)
## Michael null
## Andy    31
## Justin  20

# Select people older than 21
df.filter(df['age'] > 21).show()
## age name
## 30  Andy

# Count people by age
df.groupBy("age").count().show()
## age  count
## null 1
## 19   1
## 30   1
```

JSON Datasets

Scala Java Python R Sql

```
CREATE TEMPORARY TABLE jsonTable
USING org.apache.spark.sql.json
OPTIONS (
  path "examples/src/main/resources/people.json"
)

SELECT * FROM jsonTable
```

DataFrames

people.json

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

Load data from json file and show the schema

```
df = sqlContext.read.json("people.json")
df.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
df.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

DataFrames

```
df.select("name").show()
```

```
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
df.select(df['name'], df['age'] + 1).show()
```

```
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+
```

DataFrames

```
+----+-------
| age|    name
+----+-------
|null|Michael
|  30|   Andy
|  19| Justin
+----+-------
```

```
: df.filter(df['age'] > 21).show()
```

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

```
: df.groupBy("age").count().show()
```

```
+----+-----+
| age|count|
+----+-----+
|null|    1|
|  19|    1|
|  30|    1|
+----+-----+
```

DataFrames

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
df.select("name").show()
```

```
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
```

```
df.select(df['name'], df['age'] + 1).show()
```

```
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+
```

```
: df.filter(df['age'] > 21).show()
```

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

```
: df.groupBy("age").count().show()
```

```
+----+-----+
| age|count|
+----+-----+
|null|    1|
|  19|    1|
|  30|    1|
+----+-----+
```

DataFrames

people.txt

```
Michael, 29
Andy, 30
Justin, 19
```

Load data from a text file

Load data into a PythonRDD and then transform an PythonRDD to an DataFrame

```
from pyspark.sql import Row
# Load a text file and convert each line to a Row.
lines = sc.textFile("people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
people
```

Create an RDD

```
PythonRDD[152] at RDD at PythonRDD.scal
```

```
[Row(age=29, name=u'Michael'),
 Row(age=30, name=u'Andy'),
 Row(age=19, name=u'Justin')]
```

```
schemaPeople = sqlContext.createDataFrame(people)
# schemaPeople = people.toDF() is another way to create DataFrame
schemaPeople
```

Transform RDD to a dataframe

```
DataFrame[age: bigint, name: string]
```

```
[Row(age=29, name=u'Michael'),
 Row(age=30, name=u'Andy'),
 Row(age=19, name=u'Justin')]
```

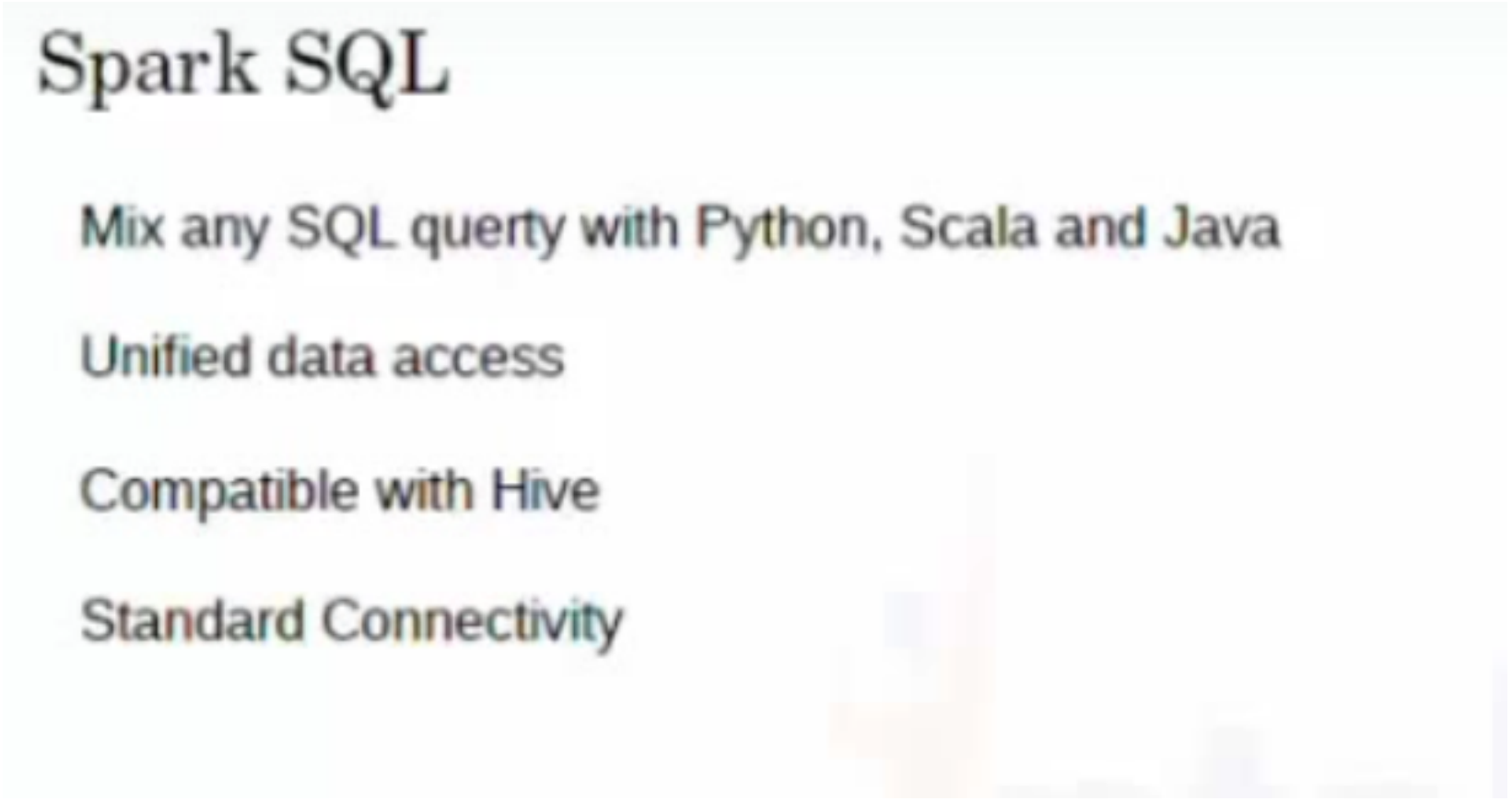
DataFrames

- **Register the DataFrame as a table and then run SQL queries**

```
# Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

# SQL can be run over DataFrames that have been registered as a table.
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

# The results of SQL queries are RDDs and support all the normal RDD operations.
teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
    print teenName
```

```
Name: Justin
```

Spark SQL


Spark SQL
Mix any SQL querty with Python, Scala and Java
Unified data access
Compatible with Hive
Standard Connectivity

Spark SQL

- Native SQL Interface to Spark
- Hive, DataFrame, JSON, RDD, etc...
- JDBC Server (B.I. Interface)
- UDFs (Spark SQL and Hive)
- Columnar storage, Predicate Pushdowns, Tuning options

Spark SQL Example

```
hiveCtx = HiveContext(sc)
allData = hiveCtx.jsonFile(filein)
allData.registerTempTable("customers")

query1 = hiveCtx.sql("
                SELECT last, first
                FROM customers
                ORDER BY last
                LIMIT 50")
```

Spark SQL

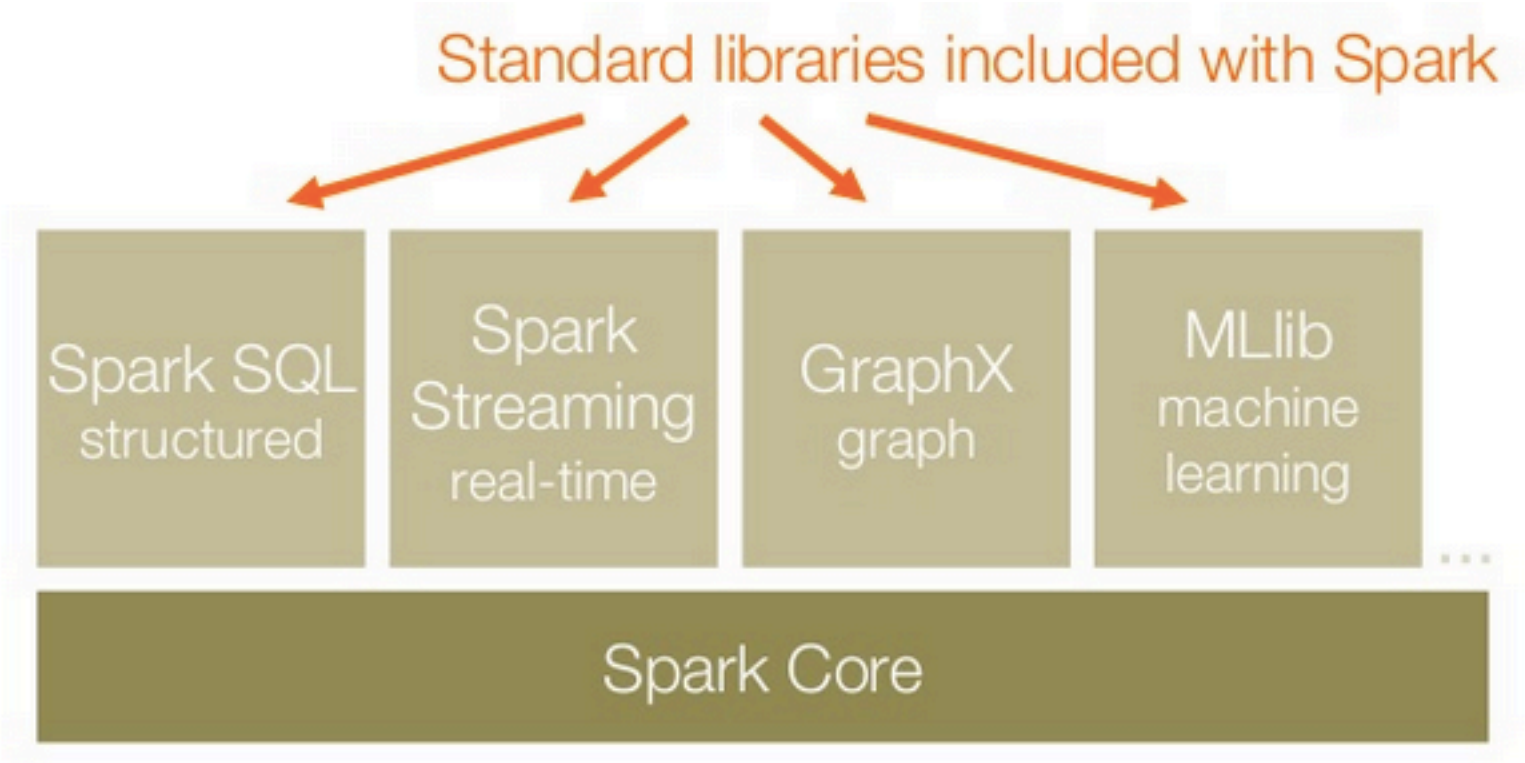
Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks

- **Machine Learning in Spark**
  - Mllib
  - Write your own!

Spark Machine Learning

- ..



Spark MLLib Functionality 1/2

MLlib is Apache Spark's scalable machine learning library

- **Supervised ML**
  - linear SVM and logistic regression
  - classification and regression tree
  - multinomial naive Bayes
  - random forest and gradient-boosted trees
  - linear regression with L1- and L2-regularization
  - isotonic regression
- **recommendation via alternating least squares**
- **Unsupervised learning**
  - clustering via k-means, Gaussian mixtures, and power iteration clustering
  - topic modeling via latent Dirichlet allocation
  - singular value decomposition
- **frequent itemset mining via FP-growth**
- **Statistics**
  - Summary statistics, Correlation, Chi-square
- **Feature transformations**

Spark MLLib Functionality 2/2

- **Pipeline API (Train, Evaluation, Testing)**
  - EDA
  - Feature engineering
  - Data engineering
  - Learning
  - Hyperparameter tuning
  - Testing

- ..

Machine Learning Library (MLlib)

```
points = context.sql("select latitude, longitude from tweets")
model = KMeans.train(points, 10)
```

MLLib

- **MLlib is Apache Spark's scalable machine learning library.**

https://spark.apache.org/mllib/

Algorithms

MLlib 1.3 contains the following algorithms:

- linear SVM and logistic regression
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and pov iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- linear regression with L_1- and L_2-regularization
- isotonic regression
- multinomial naive Bayes
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

Refer to the MLlib guide for usage examples.

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Tre
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)
- PMML model export

1.4

data types, algorithms, and utilities

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - streaming significance testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - bisecting k-means
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
  - association rules
  - PrefixSpan
- Evaluation metrics
- PMML model export
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

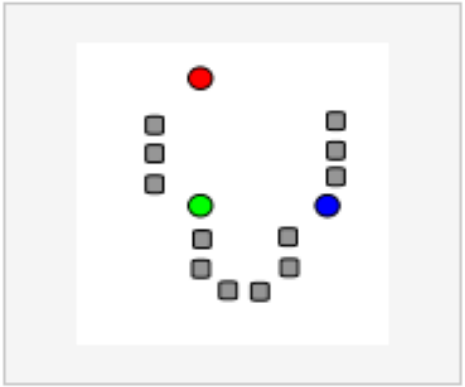
Spark 1.6

MLlib Example

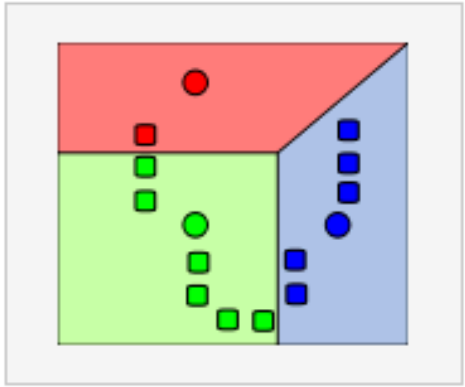
```
# Linear Regression Model Build

model = LinearRegressionWithSGD.train(
            data,
            iterations = 100,
            intercept = True
```

K-Means Clustering

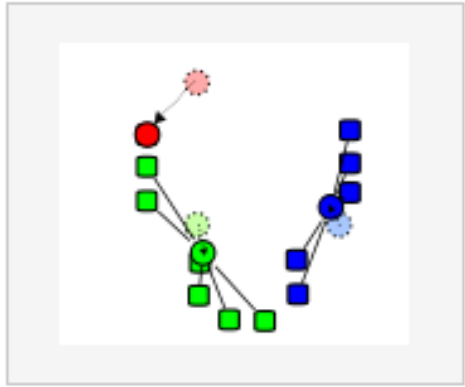
- **k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.**



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Kmeans in MLLib

The following examples can be tested in the PySpark shell.

In the following example after loading and parsing data, we use the KMeans object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm. We then compute Within Set Sum of Squared Error (WSSSE). You can reduce this error measure by increasing *k*. In fact the optimal *k* is usually one where there is an "elbow" in the WSSSE graph.

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Evaluate clustering by computing Within Set Sum of Squared Errors
Assign point to cluster centre

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Evaluate clustering by computing Within Set Sum of Squared Errors
Assign point to cluster centre

Gaussian Model: Estimate Parameters

Suppose the following are marks in a course

55.5, 67, 87, 48, 63

Marks typically follow a Normal distribution whose density function is

$$N(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2\sigma}(x-\mu)^2}$$

Now, we want to find the best μ,σ such that

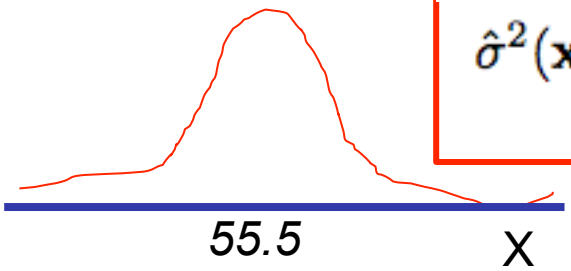
$$argmax_{\mu,\sigma} p(Data|\mu, \sigma)$$

$$\hat{\mu}(\mathbf{x}) = \bar{x}$$

$$\hat{\sigma}^2(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2.$$

$\hat{\mu}(\mathbf{x})\ \hat{\sigma}^2(\mathbf{x})$ **are MLE for μ, σ²**

> mean(55.5, 67, 87, 48, 63)
[1] 55.5
> sd(c(55.5, 67, 87, 48, 63))
[1] 14.72413



Maximum-likelihood estimation

- **Maximum-likelihood estimation (MLE) is a method of estimating the parameters of a statistical model. When applied to a data set and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters.**
- **Expectation–maximization (EM) algorithm is an iterative method for finding maximum likelihood, where the model depends on unobserved latent variables.**
  - The EM iteration alternates between performing
  - an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters,
  - and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the Estep.
    - These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

MLE Estimates for GMM

If we knew z: class assignments for each example

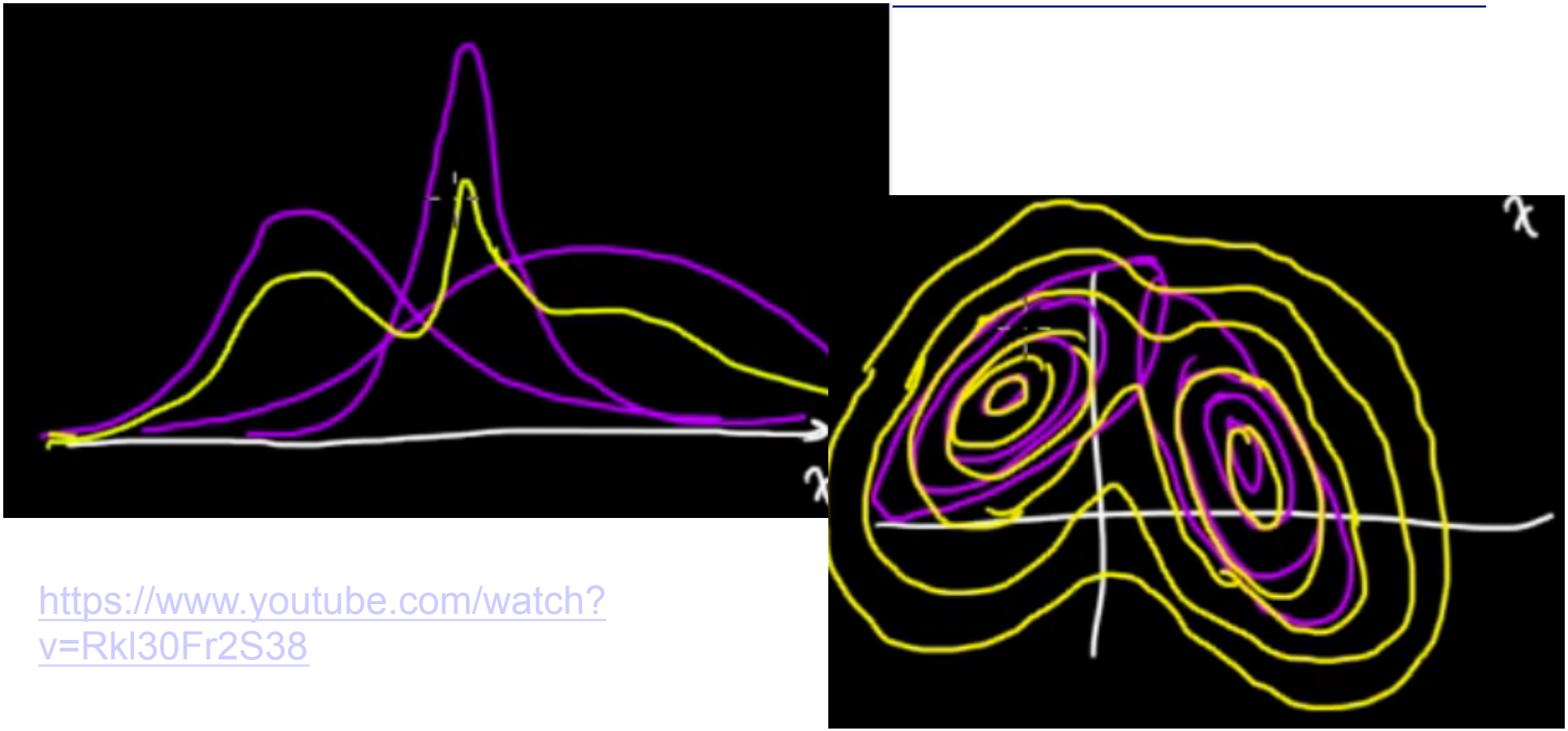
likelihood problem would have been easy. Specifically, we could then write down the likelihood as

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{m} \log p(x^{(i)}|z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi).$$

Maximizing this with respect to ϕ, μ and Σ gives the parameters:

$$\begin{aligned}
\phi_j &= \frac{1}{m} \sum_{i=1}^{m} 1\{z^{(i)} = j\}, \\
\mu_j &= \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}, \\
\Sigma_j &= \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}.
\end{aligned}$$

PDF of a mixture of Gaussians



https://www.youtube.com/watch?v=Rkl30Fr2S38

Yellow curve is convex combination of the individual Gaussian

But $z^{(i)}$'s are not known: so use EM

- **However, in our density estimation problem, the $z^{(i)}$'s are not known.**
- **What can we do?**
- **The EM algorithm is an iterative algorithm that has two main steps.**
  - Applied to our problem, in the E-step, it tries to "guess" the values of the $z_{(i)}$'s.
  - In the M-step, it updates the parameters of our model based on our guesses. Since in the M-step we are pretending that the guesses in the first part were correct, the maximization becomes easy. Here's the algorithm:

EM for GMM

Driver: Initialize μ, Σ, ϕ

- ..

Repeat until convergence: {

(E-step) For each i, j, set $w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$

J class; I example index

(M-step) Update the parameters:

phi $\phi_j := \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$,

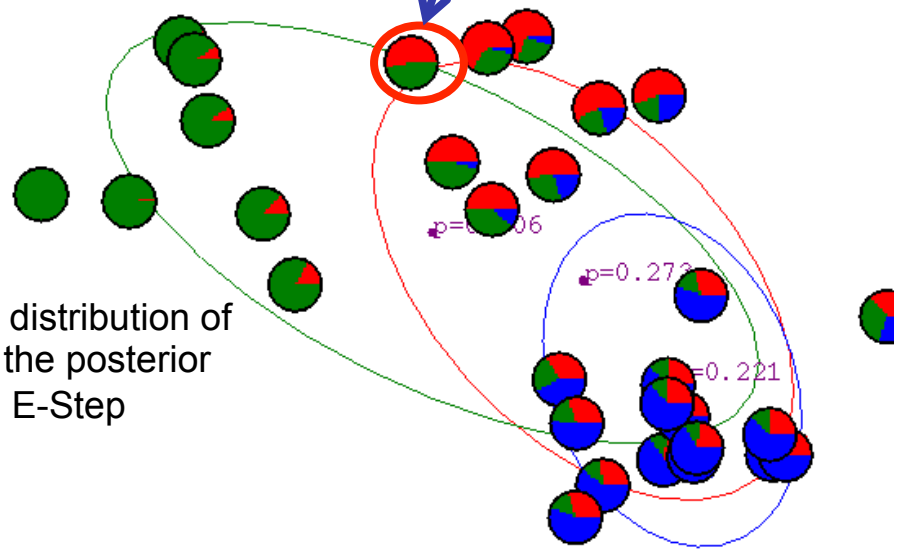
Compute relative distribution of each class using the posterior probabilities from E-Step

$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$,

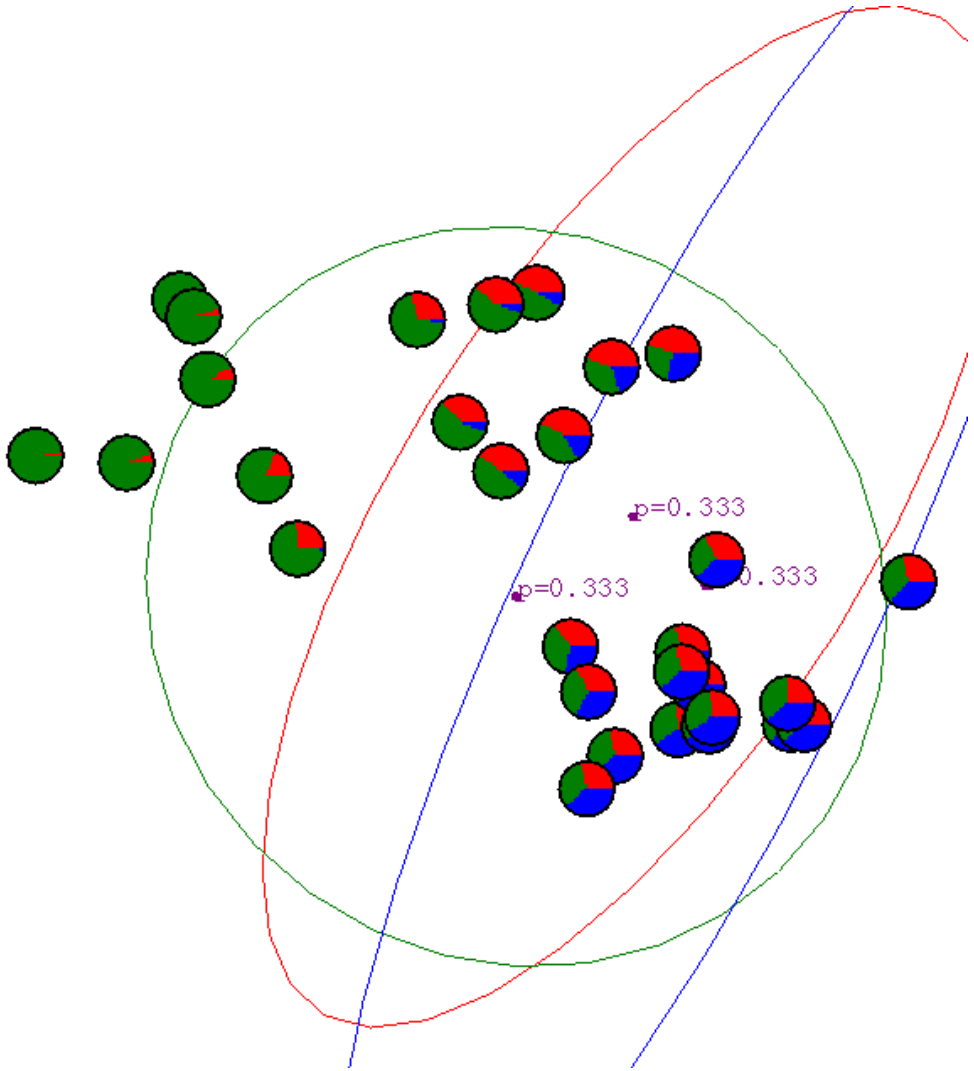
$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$

E-Step

Pr(Class = Red|X) = W_{red}=0.5;
W_{green}=0.5;
W_{blue}=0.0



Random init: and assignment



EM for GMM: write the mapper

Commutative and associative ops are great candidates for Mapper/combiner

Driver: Initialize μ, Σ, ϕ

- ..

Repeat until convergence: {

(E-step) For each i, j, set J class; I example index

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

phi

$$\phi_j := \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)},$$

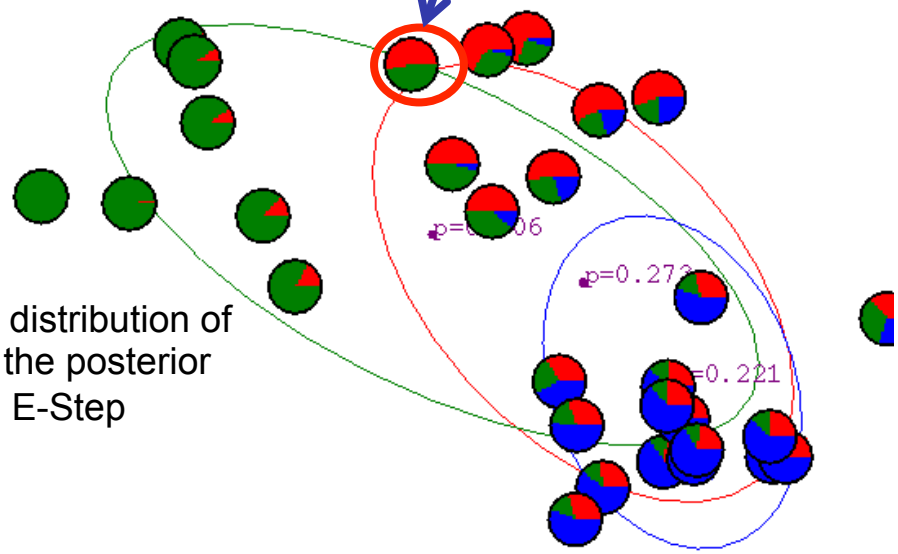
Compute relative distribution of each class using the posterior probabilities from E-Step

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

E-Step

Pr(Class = Red|X) = W_{red}=0.5;
W_{green}=0.5;
W_{blue}=0.0



EM for GMM: write the REDUCER

Commutative and associative do we have great candidates for Mapper/combiner

Driver: Initialize μ_j, Σ_j, ϕ_j

- ..

Repeat until convergence: {

(E-step) For each j, j, set j class, i example index

$p_j := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$

(M-step) Update the parameters:

Partition key for reducer?

Partial sums for

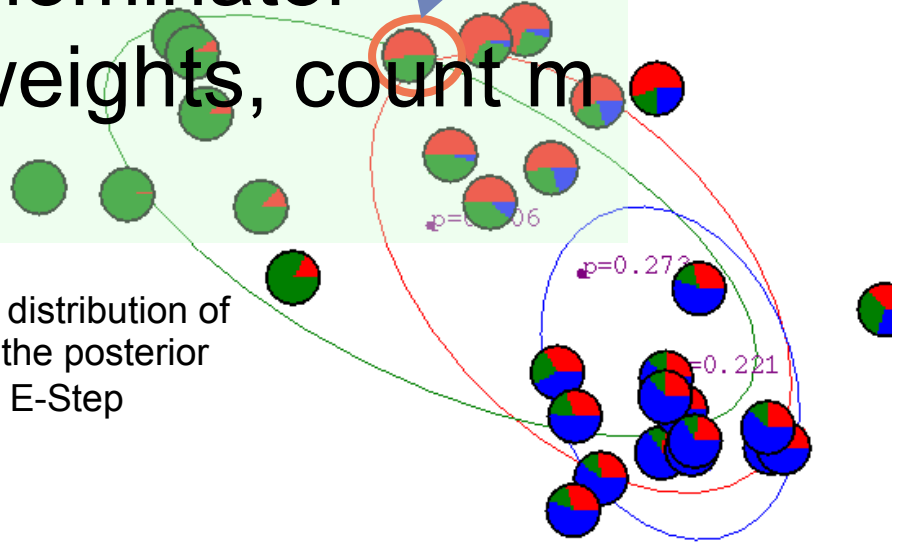
μ, numerator, denominator

Σ, numerator, denominator

ϕ partial sum of weights, count m

E-Step

Pr(Class = Red|X) = W_{red}=0.5;
W_{green}=0.5;
W_{blue}=0.0



phi $\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)}$,

Compute relative distribution of each class using the posterior probabilities from E-Step

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

In the E-step, we calculate the posterior probability of our parameters the $z^{(i)}$'s, given the $x^{(i)}$ and using the current setting of our parameters. I.e., using Bayes rule, we obtain:

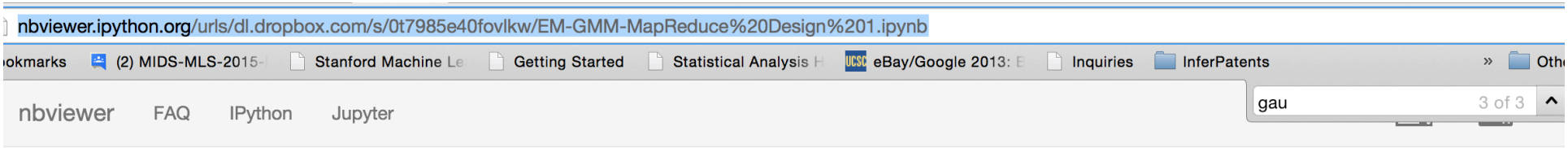
- $$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

Here, $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$ is given by evaluating the density of a Gaussian with mean μ_j and covariance Σ_j at $x^{(i)}$; $p(z^{(i)} = j; \phi)$ is given by ϕ_j, and so on. The values $w_j^{(i)}$ calculated in the E-step represent our "soft" guesses[2] for the values of $z^{(i)}$.

Also, you should contrast the updates in the M-step with the formulas we had when the $z^{(i)}$'s were known exactly. They are identical, except that instead of the indicator functions "$1\{z^{(i)} = j\}$" indicating from which Gaussian each datapoint had come, we now instead have the $w_j^{(i)}$'s.

The EM-algorithm is also reminiscent of the K-means clustering algorithm, except that instead of the "hard" cluster assignments $c(i)$, we instead have the "soft" assignments $w_j^{(i)}$. Similar to K-means, it is also susceptible to local optima, so reinitializing at several different initial parameters may be a good idea.

EM Algorithm for GMM



Introduction

This is a map-reduce version of expectation maximization algo for a mixture of Gaussians model. There are two mrJob MR packages, mr_GMixEmIterate and mr_GMixEmInitialize. The driver calls the mrJob packages and manages the iteration.

E Step: Given mean vector and covariance matrix, calculate the probability of that each data point belongs to a class

$P(Cluster_k | X^i; \theta)$

$$p(\omega_k|\boldsymbol{x}^{(i)},\theta) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}$$

Estimate class assignments (responsibities) or weights
Use the current model to estimate the class assignment

M Step: Given probabilities, update mean and covariance

$Centroid_k$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k}\sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)\boldsymbol{x}^{(i)}$$

Centroid is just a weighted sum of soft assigned examples

$Covariance_k$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k}\sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)(\boldsymbol{x}^{(i)}-\hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}^{(i)}-\hat{\boldsymbol{\mu}}_k)^T$$

Weighted covariance

$Prior_k$

$$\hat{\pi}_k = \frac{n_k}{n} \text{ where } n_k = \sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)$$

Class prior is based on the sum of the partial weights

Gaussian Mixture Model

- ..

2001 lines (2000 sloc) | 62.5 KB

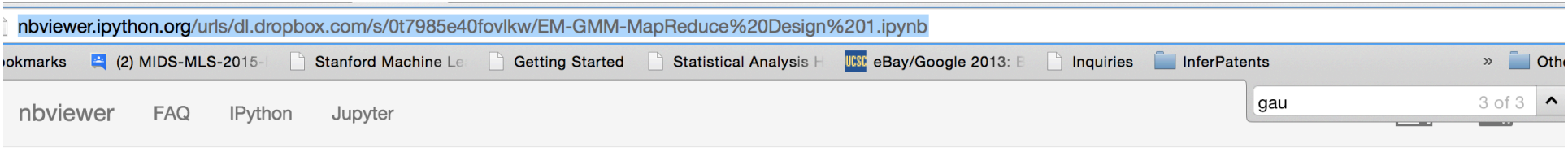
```
2.59470454e+00 2.12298217e+00
1.15807024e+00 -1.46498723e-01
2.46206638e+00 6.19556894e-01
-5.54845070e-01 -7.24700066e-01
-3.23111426e+00 -1.42579084e+00
3.02978115e+00 7.87121753e-01
1.97365907e+00 1.15914704e+00
```

In the following example after loading and parsing data, we use a GaussianMixture object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm. We then output the parameters of the mixture model.

Refer to the `GaussianMixture` Python docs and `GaussianMixtureModel` Python docs for more details on the API.

```
from pyspark.mllib.clustering import GaussianMixture
from numpy import array

# Load and parse the data
data = sc.textFile("data/mllib/gmm_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.strip().split(' ')]))

# Build the model (cluster the data)
gmm = GaussianMixture.train(parsedData, 2)

# output parameters of model
for i in range(2):
    print ("weight = ", gmm.weights[i], "mu = ", gmm.gaussians[i].mu,
        "sigma = ", gmm.gaussians[i].sigma.toArray())
```

EM Algorithm for GMM



Introduction

This is a map-reduce version of expectation maximization algo for a mixture of Gaussians model. There are two mrJob MR packages, mr_GMixEmIterate and mr_GMixEmInitialize. The driver calls the mrJob packages and manages the iteration.

E Step: Given mean vector and covariance matrix, calculate the probability of that each data point belongs to a class

$P(Cluster_k | X^i; \theta)$

$$p(\omega_k|\boldsymbol{x}^{(i)},\theta) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}$$

Estimate class assignments (responsibities) or weights
Use the current model to estimate the class assignment

M Step: Given probabilities, update mean and covariance

$Centroid_k$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k}\sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)\boldsymbol{x}^{(i)}$$

Centroid is just a weighted sum of soft assigned examples

$Covariance_k$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k}\sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)(\boldsymbol{x}^{(i)}-\hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}^{(i)}-\hat{\boldsymbol{\mu}}_k)^T$$

Weighted covariance

$Prior_k$

$$\hat{\pi}_k = \frac{n_k}{n} \text{ where } n_k = \sum_{i=1}^{n} p(\omega_k|\boldsymbol{x}^{(i)},\theta)$$

Class prior is based on the sum of the partial weights

Classification

The example below demonstrates how to load a LIBSVM data file, parse it as an RDD of `LabeledPoint` and then perform classification using a Random Forest. The test error is calculated to measure the algorithm accuracy.

Scala Java Python

Classification using Random Forests

```python
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     numTrees=3, featureSubsetStrategy="auto",
                                     impurity='gini', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = RandomForestModel.load(sc, "myModelPath")
```

Classification using Random Forests

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     numTrees=3, featureSubsetStrategy="auto",
                                     impurity='gini', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = RandomForestModel.load(sc, "myModelPath")
```

Gradient Boosted DTs

Gradient-Boosted Trees (GBTs) are ensembles of decision trees. GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, GBTs handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

MLlib supports GBTs for binary classification and for regression, using both continuous and categorical features. MLlib implements GBTs using the existing decision tree implementation. Please see the decision tree guide for more information on trees.

Note: GBTs do not yet support multiclass classification. For multiclass problems, please use decision trees or Random Forests.

Basic algorithm

Gradient boosting iteratively trains a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label. The dataset is re-labeled to put more emphasis on training instances with poor predictions. Thus, in the next iteration, the decision tree will help correct for previous mistakes.

The specific mechanism for re-labeling instances is defined by a loss function (discussed below). With each iteration, GBTs further reduce this loss function on the training data.

Losses

The table below lists the losses currently supported by GBTs in MLlib. Note that each loss is applicable to one of classification or regression, not both.

Notation: N = number of instances. y_i = label of instance i. x_i = features of instance i. $F(x_i)$ = model's predicted label for instance i.

Loss	Task	Formula	Description
Log Loss	Classification	$2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$	Twice binomial negative log likelihood.
Squared Error	Regression	$\sum_{i=1}^{N}(y_i-F(x_i))^2$	Also called L2 loss. Default loss for regression tasks.
Absolute Error	Regression	$\sum_{i=1}^{N}\lvert y_i-F(x_i)\rvert$	Also called L1 loss. Can be more robust to outliers than Squared Error.

Gradient Boosted DTs

Gradient-Boosted Trees (GBTs) are ensembles of decision trees. GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, re able to capture non-li

MLlib supports GBTs using the existing de

Note: GBTs do

Basic alg

Gradient boos ct the label of each training i ng instances with poor prec

The specific m educe this loss function c

Losses

The table belo gression, not both.

Notation: N = i.

Losses

The table below lists the losses currently supported by GBTs in MLlib both.

Notation: N = number of instances. y_i = label of instance i. x_i = featu

Loss	Task	Formula
Log Loss	Classification	$2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$
Squared Error	Regression	$\sum_{i=1}^{N}(y_i-F(x_i))^2$
Absolute Error	Regression	$\sum_{i=1}^{N}\lvert y_i-F(x_i)\rvert$

Loss	Task	Formula	Description
Log Loss	Classification	$2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$	Twice binomial negative log likelihood.
Squared Error	Regression	$\sum_{i=1}^{N}(y_i-F(x_i))^2$	Also called L2 loss. Default loss for regression tasks.
Absolute Error	Regression	$\sum_{i=1}^{N}\lvert y_i-F(x_i)\rvert$	Also called L1 loss. Can be more robust to outliers than Squared Error.

The example below demonstrates how to load a LIBSVM data file, parse it as an RDD of `LabeledPoint` and then perform classification using Gradient-Boosted Trees with log loss. The test error is calculated to measure the algorithm accuracy.

Scala Java Python

Decision Tree Ensembles

```python
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file.
data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a GradientBoostedTrees model.
#  Notes: (a) Empty categoricalFeaturesInfo indicates all features are continuous.
#         (b) Use more iterations in practice.
model = GradientBoostedTrees.trainClassifier(trainingData,
    categoricalFeaturesInfo={}, numIterations=3)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification GBT model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = GradientBoostedTreesModel.load(sc, "myModelPath")
```

Decision Tree Ensembles

```
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file.
data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a GradientBoostedTrees model.
#  Notes: (a) Empty categoricalFeaturesInfo indicates all features are continuous.
#         (b) Use more iterations in practice.
model = GradientBoostedTrees.trainClassifier(trainingData,
    categoricalFeaturesInfo={}, numIterations=3)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification GBT model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = GradientBoostedTreesModel.load(sc, "myModelPath")
```

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks

- **Pipelines**

Pipelines

- **In machine learning, it is common to run a sequence of algorithms to process and learn from data.**
- **E.g., a simple text document processing workflow might include several stages:**
  - Split each document's text into words.
  - Convert each document's words into a numerical feature vector.
  - Normalization
  - Learn a prediction model using the feature vectors and labels.
  - Spark ML represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this simple workflow as a running example in this section.

Pipeline-Training data

In Python, The single asterisk form (*args) is used to pass a non-keyworded, variable-length argument list in function definition. It can also unpacks the sequence/ collection into positional arguments

Example of Unpacking

```
def sum(a, b):
    return a + b

values = (1, 2)

s = sum(*values)
```

```
# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0, "a b c d e spark", 1.0),
                           (1, "b d", 0.0),
                           (2, "spark f g h", 1.0),
                           (3, "hadoop mapreduce", 0.0),
                           (4, "b spark who", 1.0),
                           (5, "g d a y", 0.0),
                           (6, "spark fly", 1.0),
                           (7, "was mapreduce", 0.0),
                           (8, "e spark program", 1.0),
                           (9, "a e c l", 0.0),
                           (10, "spark compile", 1.0),
                           (11, "hadoop software", 0.0)
                           ]) \
    .map(lambda x: LabeledDocument(*x)).toDF()
training.collect()
```

```
[Row(id=0, text=u'a b c d e spark', label=1.0),
 Row(id=1, text=u'b d', label=0.0),
 Row(id=2, text=u'spark f g h', label=1.0),
 Row(id=3, text=u'hadoop mapreduce', label=0.0),
 Row(id=4, text=u'b spark who', label=1.0),
 Row(id=5, text=u'g d a y', label=0.0),
 Row(id=6, text=u'spark fly', label=1.0),
 Row(id=7, text=u'was mapreduce', label=0.0),
 Row(id=8, text=u'e spark program', label=1.0),
 Row(id=9, text=u'a e c l', label=0.0),
 Row(id=10, text=u'spark compile', label=1.0),
 Row(id=11, text=u'hadoop software', label=0.0)]
```

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

There are several variants on the definition of term frequency and document frequency. In MLlib, we separate TF and IDF to make them flexible.

Our implementation of term frequency utilizes the hashing trick. A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1,048,576$.

Note: MLlib doesn't provide tools for text segmentation. We refer users to the Stanford NLP Group and scalanlp/chalk.

Scala | Python

TF and IDF are implemented in HashingTF and IDF. `HashingTF` takes an RDD of list as the input. Each record could be an iterable of strings or other types.

```
from pyspark import SparkContext
from pyspark.mllib.feature import HashingTF

sc = SparkContext()

# Load documents (one per line).
documents = sc.textFile("...").map(lambda line: line.split(" "

hashingTF = HashingTF()
tf = hashingTF.transform(documents)
```

While applying `HashingTF` only needs a single pass to the data, applying scale the term frequencies by IDF.

```
from pyspark.mllib.feature import IDF

# ... continue from the previous example
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

MLLib's IDF implementation provides an option for ignoring terms which IDF for these terms is set to 0. This feature can be used by passing the m

```
# ... continue from the previous example
tf.cache()
idf = IDF(minDocFreq=2).fit(tf)
tfidf = idf.transform(tf)
```

A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is 220=1,048,576220=1,048,576.

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

There are several variants on the definition of term frequency and document frequency. In MLlib, we separate TF and IDF to make them flexible.

Our implementation of term frequency utilizes the hashing trick. A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1,048,576$.

Note: MLlib doesn't provide tools for text segmentation. We refer users to the Stanford NLP Group and scalanlp/chalk.

Scala | Python

TF and IDF are implemented in HashingTF and IDF. `HashingTF` takes an RDD of list as the input. Each record could be an iterable of strings or other types.

```
from pyspark import SparkContext
from pyspark.mllib.feature import HashingTF

sc = SparkContext()

# Load documents (one per line).
documents = sc.textFile("...").map(lambda line: line.split(" "))

hashingTF = HashingTF()
tf = hashingTF.transform(documents)
```

While applying `HashingTF` only needs a single pass to the data, applying `IDF` needs two passes: first to compute the IDF vector and second to scale the term frequencies by IDF.

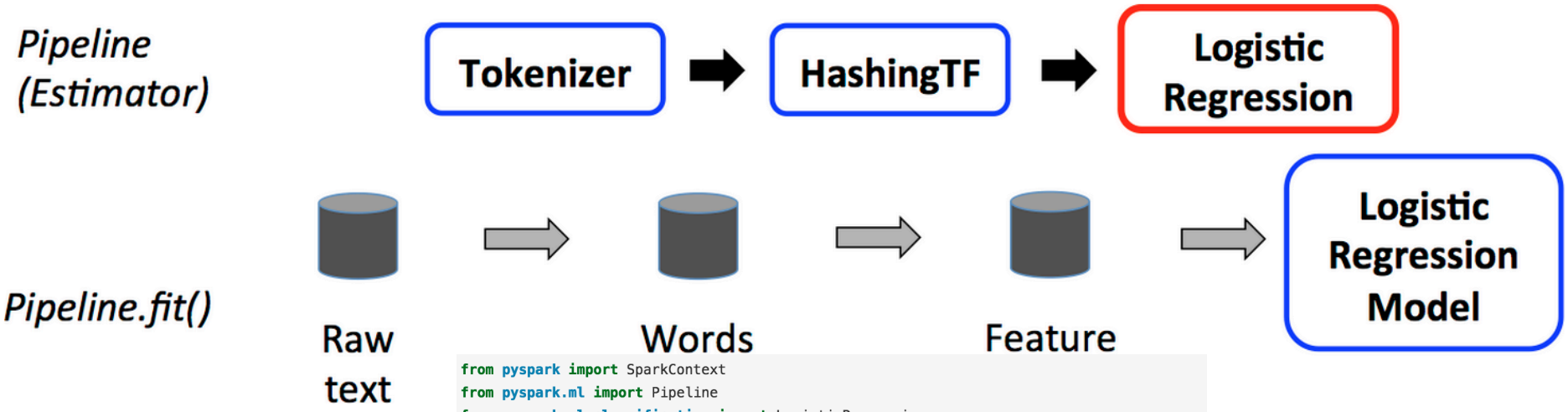
```
from pyspark.mllib.feature import IDF

# ... continue from the previous example
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

MLLib's IDF implementation provides an option for ignoring terms which occur in less than a minimum number of documents. In such cases the IDF for these terms is set to 0. This feature can be used by passing the `minDocFreq` value to the IDF constructor.

```
# ... continue from the previous example
tf.cache()
idf = IDF(minDocFreq=2).fit(tf)
tfidf = idf.transform(tf)
```

A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is 220=1,048,576220=1,048,576.

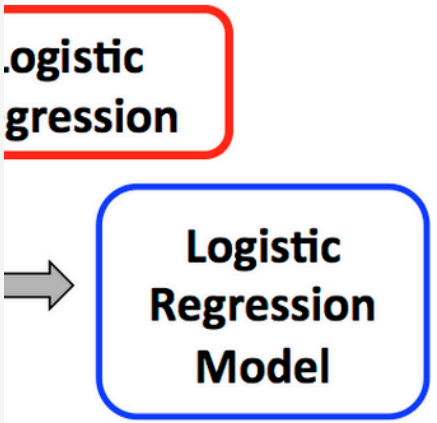
Training Pipeline



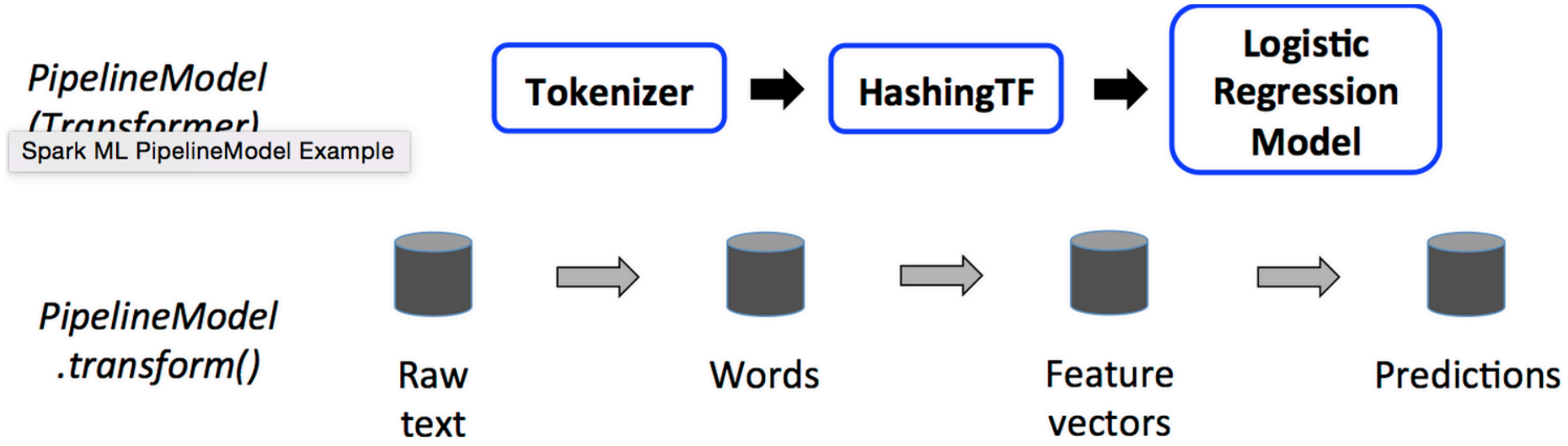

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

Training Pipeline

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```



Testing pipeline



above, the PipelineModel has the same number of stages as the original Pipeline, but all Estimators in the original Pipeli

```
# Prepare test documents, which are unlabeled.
Document = Row("id", "text")
test = sc.parallelize([(4L, "spark i j k"),
                       (5L, "l m n"),
                       (6L, "mapreduce spark"),
                       (7L, "apache hadoop")]) \
    .map(lambda x: Document(*x)).toDF()

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
selected = prediction.select("id", "text", "prediction")
for row in selected.collect():
    print row
```

ML Pipelines: use pipeline-based CV

Hyper-parameter Tuning

```
// Build a parameter grid.
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))
  .build()

// Set up cross-validation.
val cv = new CrossValidator()
  .setNumFolds(3)
  .setEstimator(pipeline)
  .setEstimatorParamMaps(paramGrid)
  .setEvaluator(new BinaryClassificationEvaluator)

// Fit a model with cross-validation.
val cvModel = cv.fit(trainingDataset)
```

Pipeline - Stages&Cross Validation


```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

Pipeline - Stages&Cross Validation

3 hashingTF.number of features: 10, 100, 1000

2 logistic regression regularization parameters: 0.1, 0.001

```
# Use a ParamGridBuilder to construct a grid of parameters to search over.
paramGrid = ParamGridBuilder() \
        .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \
        .addGrid(lr.regParam, [0.1, 0.01]) \
        .build()
```

regParam is for L2 regularization

```
# Run cross-validation, and choose the best set of parameter
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=2)  # use 3+ folds in practice
cvModel = crossval.fit(training)
```

For each iteration in cross validation, follow this pipeline

By default, BinaryclassificationEvaluator is areaUnderROC

Pipeline-Prediction

```
# Prepare test documents, which are unlabeled.
Document = Row("id", "text")
test = sc.parallelize([(4, "spark i j k"),
                       (5, "l m n"),
                       (6, "spark hadoop spark"),
                       (7, "apache hadoop")]) \
    .map(lambda x: Document(*x)).toDF()

# Make predictions on test documents and print columns of interest.
prediction = cvModel.transform(test)
selected = prediction.select("id", "text", "prediction")
for row in selected.collect():
    print(row)

sc.stop()
```

```
Row(id=4, text=u'spark i j k', prediction=1.0)
Row(id=5, text=u'l m n', prediction=0.0)
Row(id=6, text=u'spark hadoop spark', prediction=1.0)
Row(id=7, text=u'apache hadoop', prediction=0.0)
```

Outline

- **Classification Algorithms**
  - Perceptron, SVMs
  - Logistic Regression
- **Feature engineering and data engineering**
- **Non-gradient descent Algos**
  - Decision Trees (for regression and for Classification, CART)
- **Spark**
  - DataFrames
  - MLLib
- **EDA (Titanic Example)**
- **Next Steps**
- **Debrief over beer at 2:30**

- ..

DAY 1						
6/13/16	09:00-10:30	Review & stateless Naïve Bayes				
6/13/16	10:30-11:00	COFFEE BREAK				
6/13/16	11:00-12:30	Total Sorts				
6/13/16	12:30-01:30	LUNCH				
6/13/16	01:30-03:00	Gradient Descent				
6/13/16	03:00-03:30	COFFEE BREAK				
6/13/16	03:30-05:00	Linear Regression				
	5:00	Homework				
DAY 2						
6/14/16	09:00-10:30	Spart Intro		Gradient Descent Part 2 + Linear Regressior		
	10:30-11:00	COFFEE BREAK				
	11:00-12:30	Programming in Spark				
	12:30-01:30	LUNCH				
	01:30-03:00	Linear Regression in Spark				
	03:00-03:30	COFFEE BREAK				
	03:30-05:00	Kmeans in Spark		as HW tonight		
	5:00	Homework				
Day 3						
6/15/16	09:00-10:30	Classification and Loss Functions				
	10:30-11:00	COFFEE BREAK				
	11:00-12:30	Logistic Regression				
	12:30-01:30	LUNCH				
	01:30-03:00	Perceptron				
	03:00-03:30	COFFEE BREAK				
	03:30-05:00	SVMs				
	5:00	Homework				
DAY 4						
6/16/16	09:00-10:30	Feature engineering and data engineering				
	10:30-11:00	COFFEE BREAK				
	11:00-12:30	EDA and project planning				
	12:30-01:30	LUNCH				
	01:30-03:00	Predicting CTR or the length of stay of a patient (Healthcare)				
	03:00-03:30	COFFEE BREAK				
	03:30-05:00	Wrapup				

Gradient is our compass

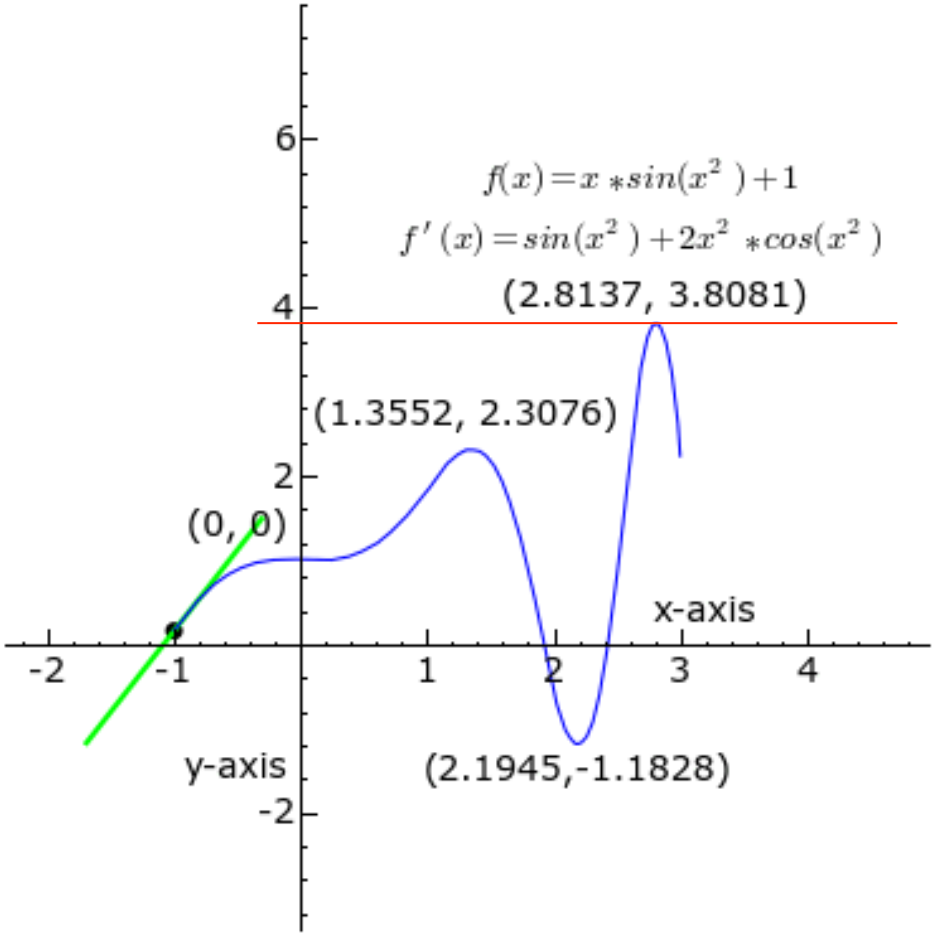
- **The gradient will act like a compass and always point us downhill. To compute it, we will need to differentiate our error function. Since our function is defined by two parameters (m and b), we will need to compute a partial derivative for each.**

Machine Learning Algorithms *(sample)*

	Unsupervised	Supervised

Continuous	• Clustering & Dimensionality Reduction
	  ○ SVD
	  ○ PCA
	  ○ K-means
		• Regression
		  ○ Linear
		  ○ Polynomial
		• Decision Trees
		• Random Forests

Categorical	• Association Analysis
	  ○ Apriori
	  ○ FP-Growth
	• Hidden Markov Model
		• Classification
		  ○ KNN
		  ○ Trees
		  ○ Logistic Regression
		  ○ Naive-Bayes
		  ○ SVM



Y=mx+c
y-y0=m(x-x0)

Next Steps

- **Next Steps**
  - Grades
  - Master Solutions
  - Homework 3 and 4
  - Project