Data X

About Me:

Gradient Descent, Classification & Logistic Regression

Alexander Fred-Ojala, Ikhlaq Sidhu, Kevin Bozhe Li

Data-X Fall 2018
Lecture 7: Outline

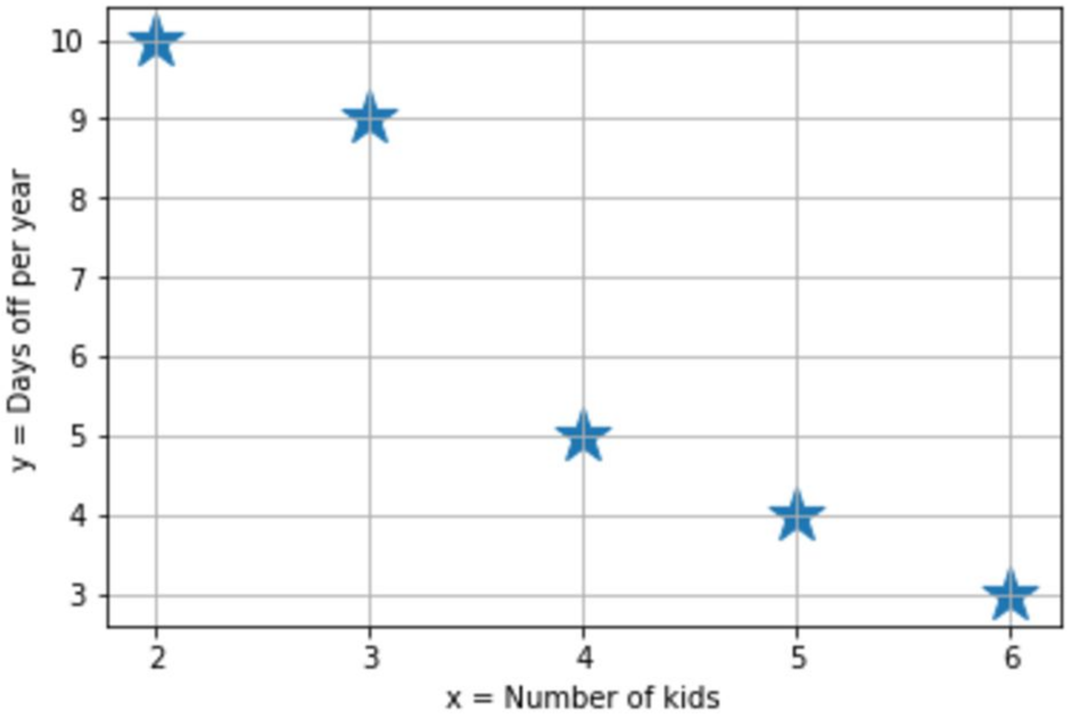
1. Linear Regression recap
2. Gradient Descent
3. Feature scaling
4. Intro to Classification
5. Logistic Regression

Recap: Linear Regression

Recap: Prediction

Given some data:
[(x_1,y_1) , (x_2 , y_2) … (x_m , y_m)]

x	**y**
2	10
4	5
3	9
5	4
6	3



Objective: Be able to predict y given new input x

Recap: Simple Linear Regression

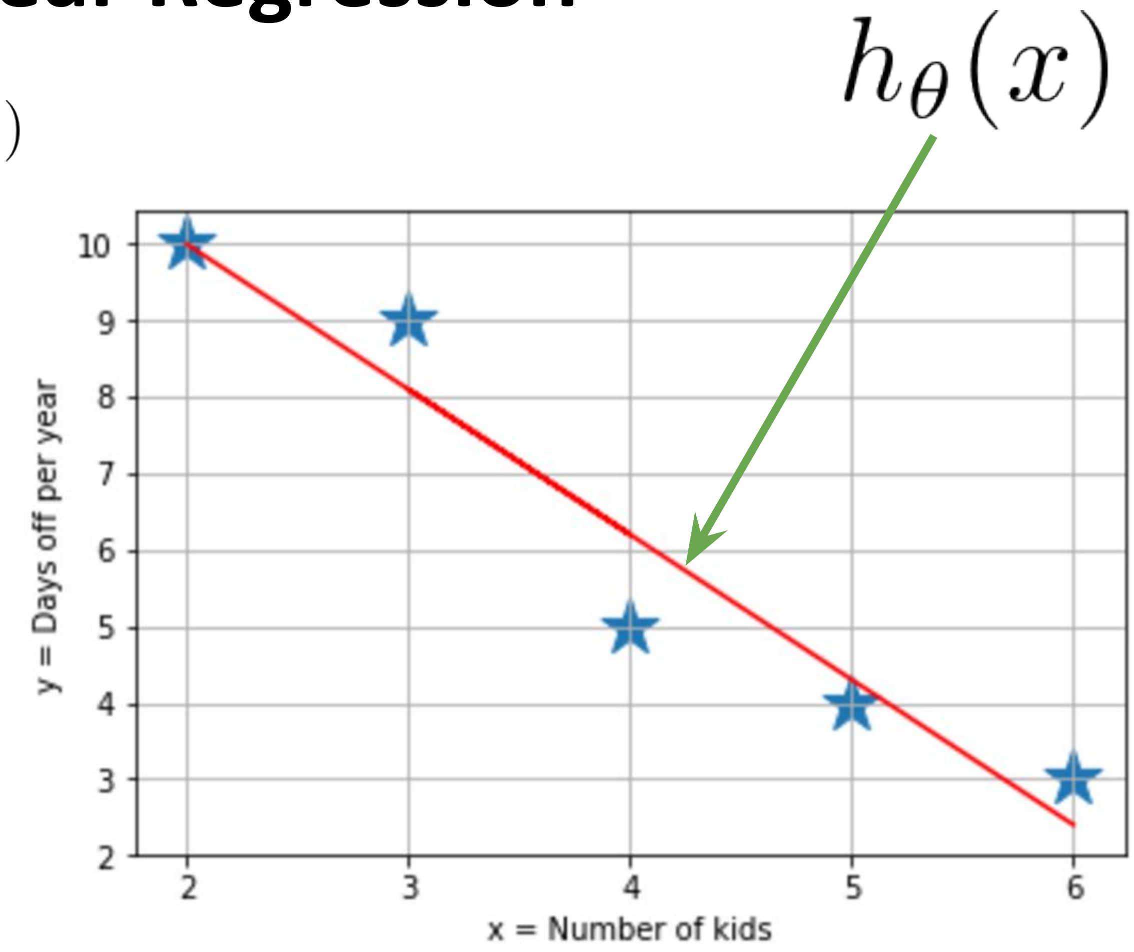
Simple Linear Regression: Hypothesis function $h_\theta(x)$

$$\hat{y} = f(x, \theta) = h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

$$x = \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$ **x is given input**

Objective: fit the best linear function to the training data, i.e. find optimal parameters θ

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



Recap: Multiple Linear Regression

Multiple Linear Regression: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \theta^T X$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdot \\ \cdot \\ \cdot \\ \theta_n \end{bmatrix} \text{ is the parameter vector and}$$

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdot & \cdot & \cdot & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdot & \cdot & \cdot & x_n^{(2)} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_0^{(m)} & x_1^{(m)} & \cdot & \cdot & \cdot & x_n^{(m)} \end{bmatrix} \text{ is the feature vector and}$$

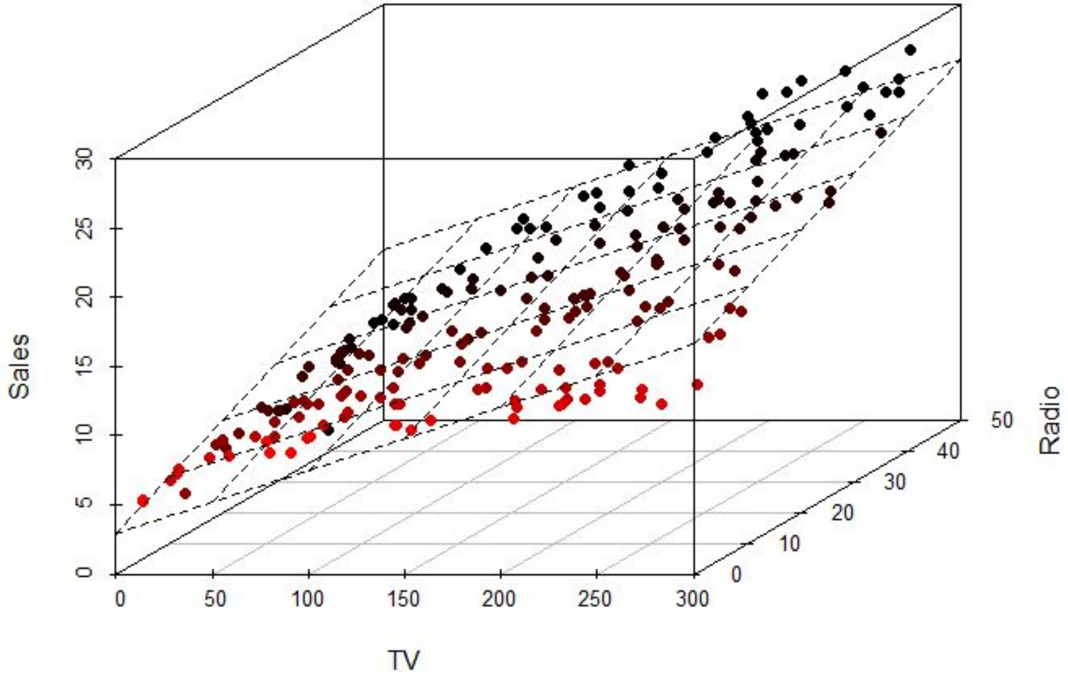
$$h_\theta(X) = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \cdot \\ \cdot \\ h_\theta(x^{(m)}) \end{bmatrix} \text{ is the hypotheses vector}$$

Example of multiple linear regression (2 features)

x_1 = TV advertising

x_2 = Radio advertising

y = Sales

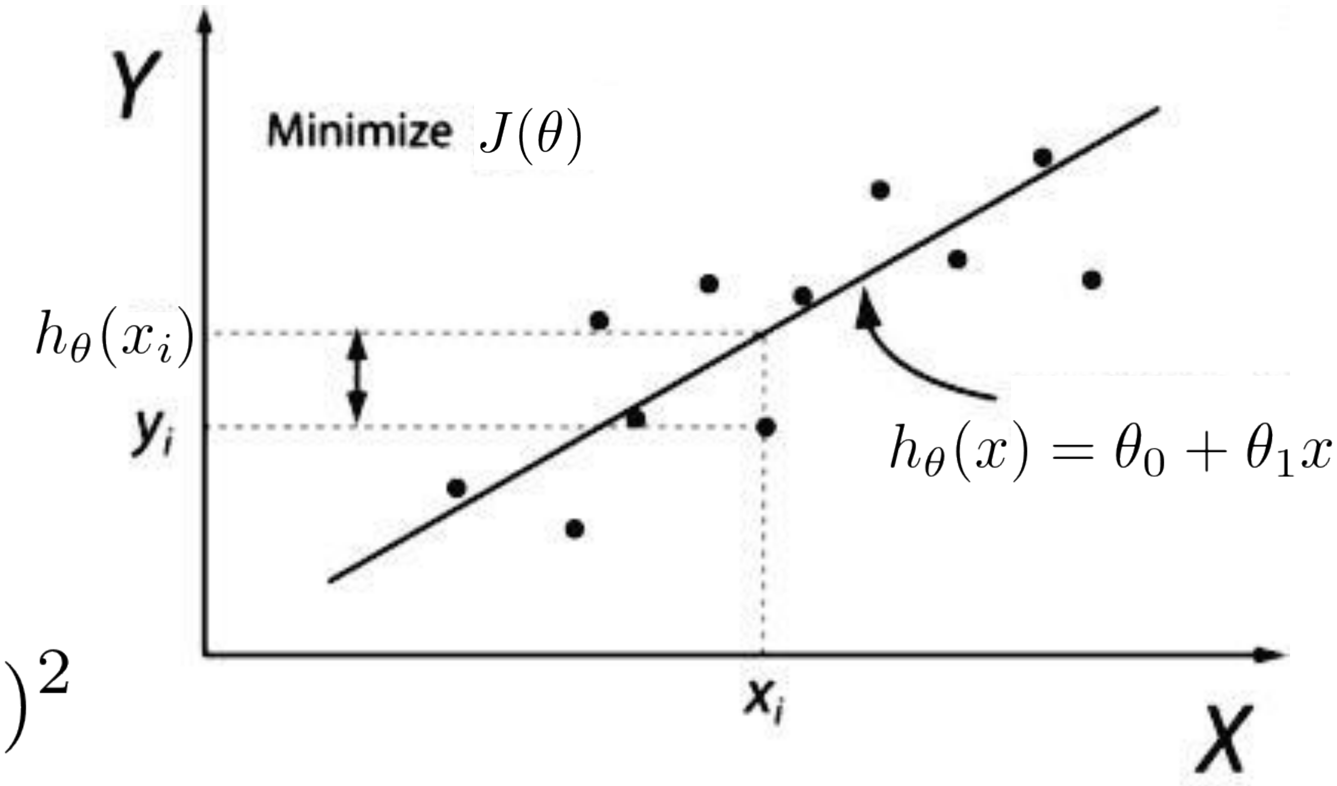


Source: 3.bp.blogspot.com/

Recap: Cost function (MSE)

Simple Linear Regression

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1$$



Cost function:

Measures how good our predictions are (MSE)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Recap: Minimize cost function

Optimal model parameters minimizes J(θ)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

$$\min_\theta \; J(\theta) \quad \Longrightarrow \quad \nabla_\theta J(\theta) = 0$$

$$\frac{\partial J}{\partial \theta} = 2X^TX\theta - 2X^Ty = 0$$

Minimize by taking the [derivative w.r.t. θ] = 0

<u>**Normal equation** for Linear Regression</u>

Closed form, analytical solution.

$$\theta = (X^TX)^{-1}X^Ty$$

<u>Pros:</u>

- Finds optimum in one calculation
- Really quick for small data sets

<u>Cons:</u>

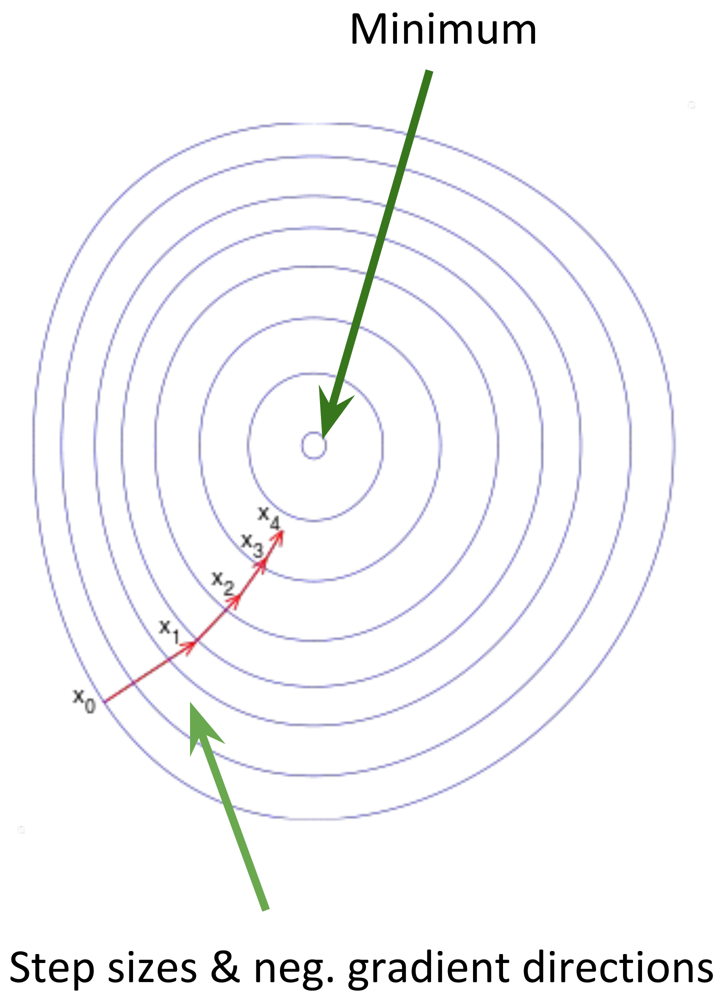
- $O(n^2m)$ complexity, to calculate (X^TX) can be slow if X has many features n.
- $(X^TX)^{-1}$ might not be invertible, ie singular (can be solved by using the pseduoinverse)

Gradient Descent

Introducing Gradient Descent

Gradient descent is a an iterative optimization algorithm for finding the minimum of a function.

To reach minima one takes steps proportional to the negative gradient (or approximate gradient) of the function at the current point.



Introducing Gradient Descent

Alternative way of minimizing the cost function:

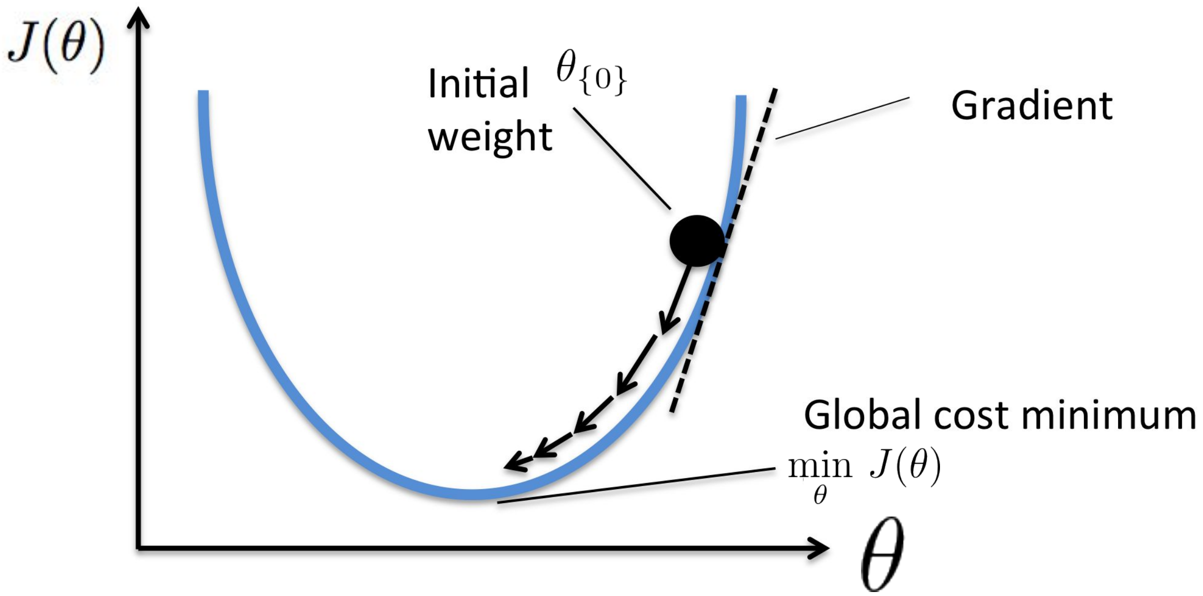
$$J(\theta) = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

- ***Will always converge because J(θ) is convex***

- Start with / initialize θ_0, θ_1 . E.g. $(\theta_0, \theta_1) = (0, 0)$
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$,

Illustration of Gradient Descent

for one parameter θ



Source: https://sebastianraschka.com

Gradient Descent Algorithm: Linear Regression

1. **Calculate the partial derivative** $\frac{\partial}{\partial \theta_j} J(\theta)$ for all j
2. Form the **update rule** for every parameter:

$$\theta_{j,iter+1} := \theta_{j,iter} - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_{j,iter} - \alpha/m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

3. **Choose a step size/ *learning rate*** α *(*often between 10^-6 and 10^2 -- not too big, then divergence).
4. **Update all the parameters** $\theta_0 \ldots \theta_n$ **by feeding in all training samples in X** ("batch" Gradient descent)
5. Stop when the error has converged.

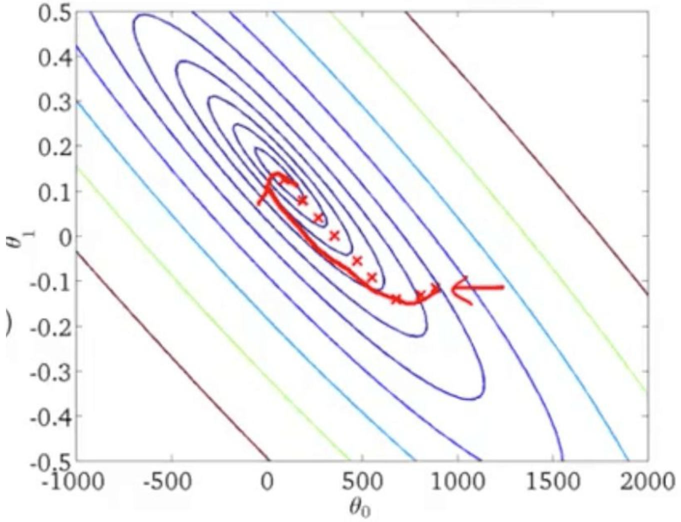
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

}



Source:Ritchie Ng

Gradient Descent Tips

Feature Scaling:

Gradient Descent will be quicker and more stable if the features are scaled.

Standardization

For all features:

- Subtract mean
- Divide by st.dev.

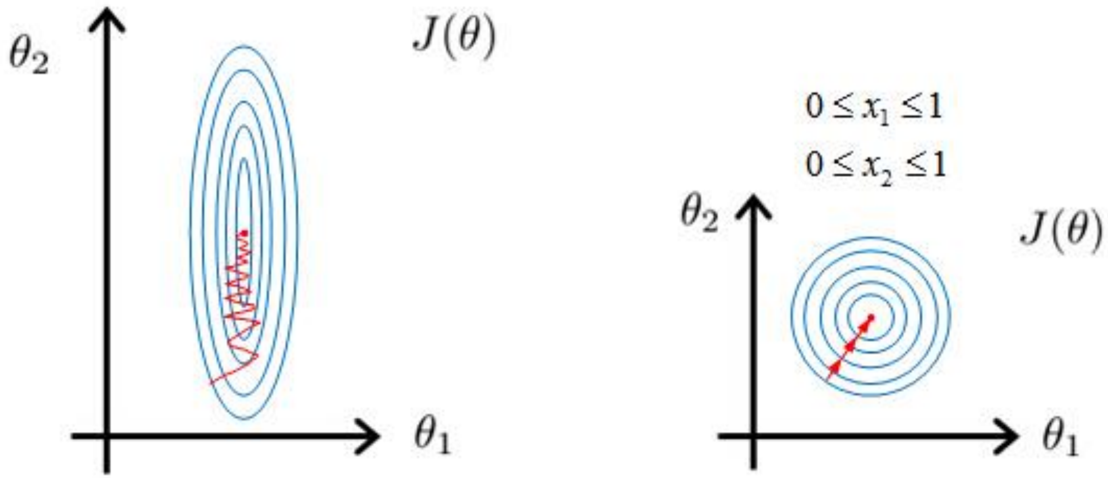
$$x_i \leftarrow \frac{x_i - \mu(x_i)}{\sigma(x_i)}$$

Min-max scaling

For all features:

- Subtract min(x_i)
- max(x_i)-min(x_i)

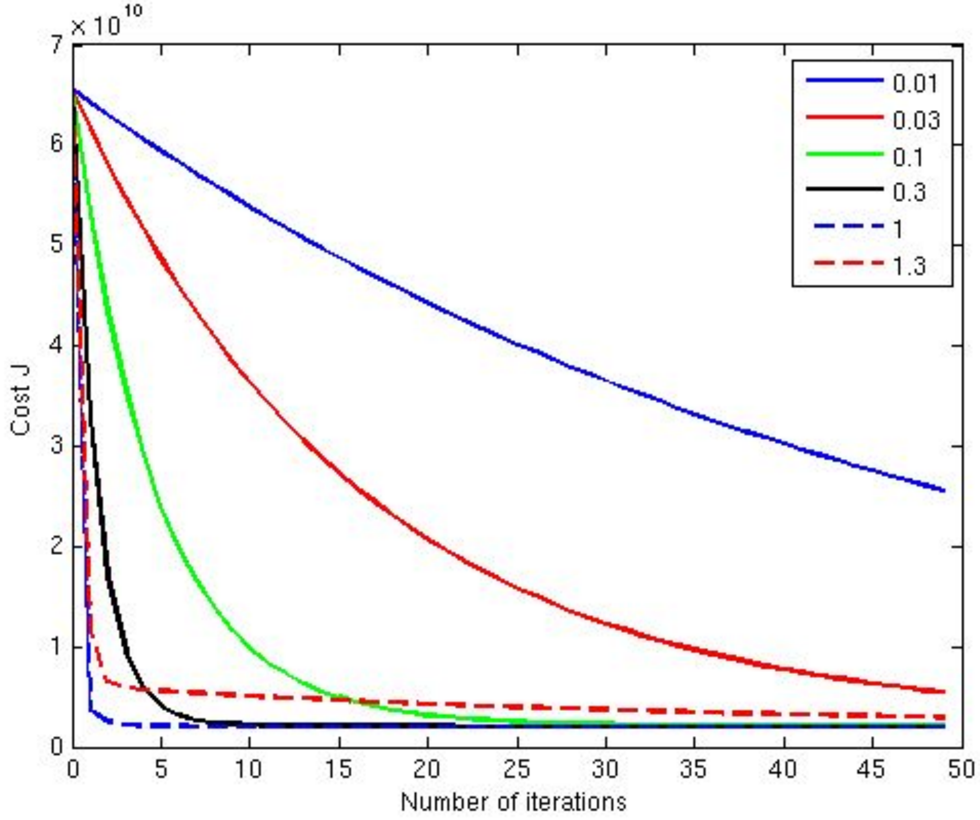
$$x_i \leftarrow \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}$$



Gradient Descent Tips

Monitor convergence

Plot the value of the error function *J(θ)* at every iteration. *Check that the error becomes smaller. Plot for different learning rates to find the best one.*



Gradient Descent

Pros

- **Will always converge** if learning rate α is chosen correctly
- **Fast** (time complexity is $O(m)$)
- Supports out of sample training (stochastic / mini batch G.D.)

Cons

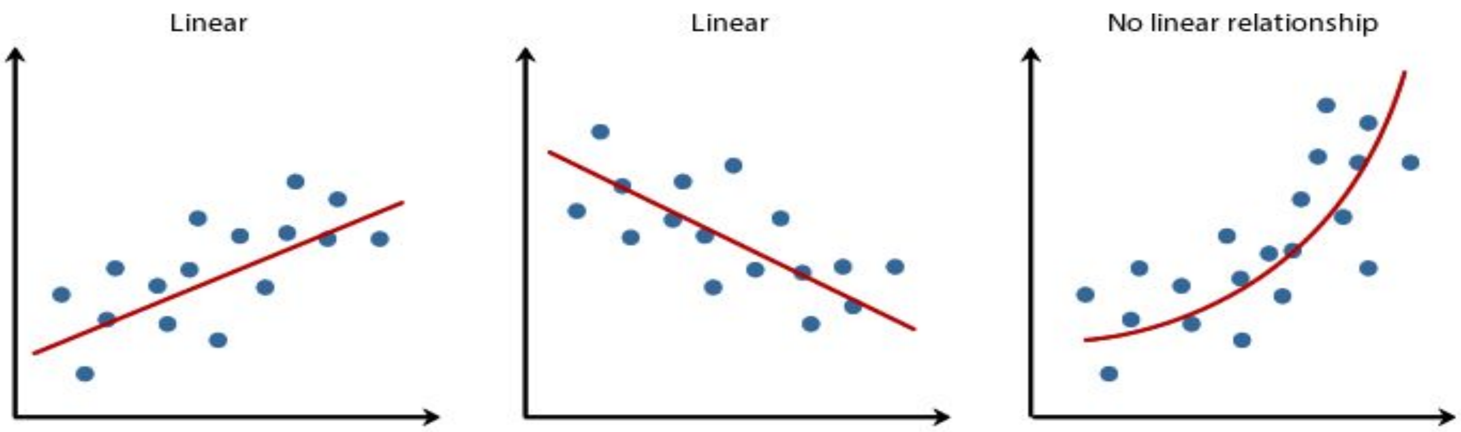
- **We have to choose learning rate** and initialize model parameters
- **Often takes many iterations**

Classification

Regression vs. Classification

Regression:

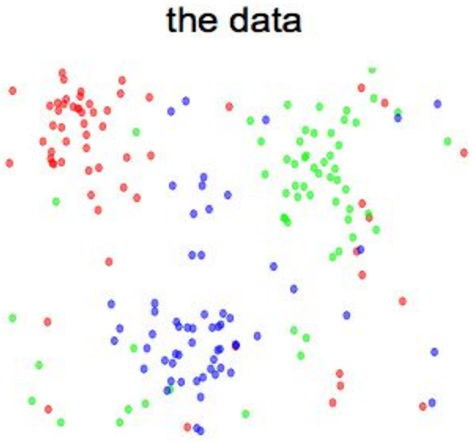
- Continuous output y
- Quantitative approach
- Linear or Non-linear



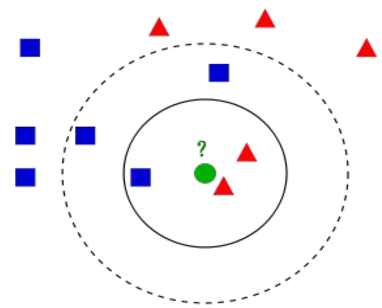
Classification:

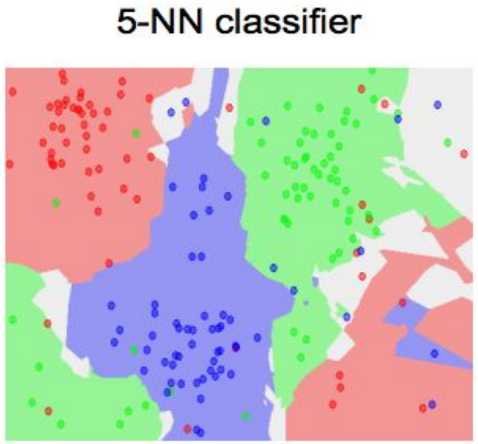
- Discrete output y
- Qualitative approach
- Linear or Non-linear

Ex. KNN,
Logitstic, SVM, ..



KNN Method: Find the k nearest images and have them vote on the label (i.e. take the mode)

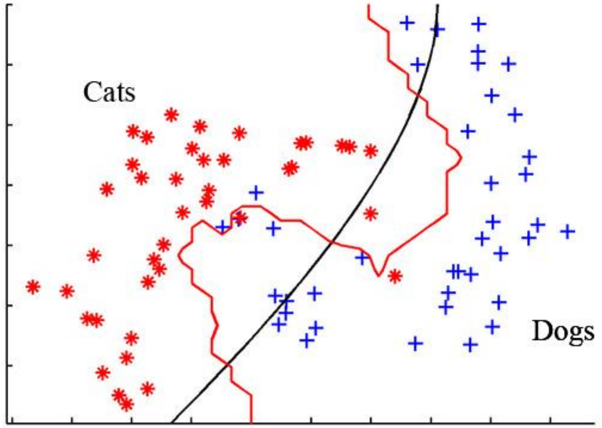


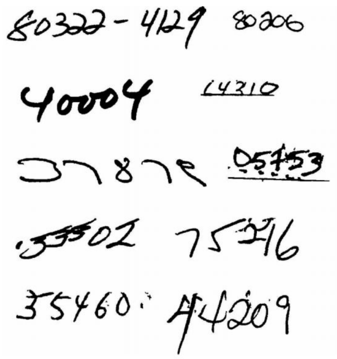


Examples of classification

Examples

- Weather: Sunny / Rainy
- Spam Detection
- Image Classification: Cats VS Dogs
- Image Classification: Recognizing Digits





Our Goal: Classify items

i.e. find the best hypothesis function $h_\theta(x)$ *that maps x to y*

xi $\xrightarrow{}$

Model

$$\hat{y} = f(x, \theta) = h_\theta(x)$$

Binary classification (cat vs dog):

y = 1 if picture is dog

Y = 0 if picture is cat

$$y \in \{0, 1\}$$

We have this data:
(X,Y): (x1,y1), (x2, y2) .. (xn,yn)

- xi and yi are arrays for each data element
- **Example:** xi = [12 15] = [height, weight], yi = male / female
- **For a picture:** xi = [32 x 32 x 3] multidim array, yi = cat / dog

Multi-class classification:

$y_i = [\, y_{i,0}, y_{i,1}, .. y_{i,k} \,]$

$y_i = [\, 1, 0, .. 0 \,]$

$$y \in \{0, 1, 2..k\}$$

Y(i,0) = 1 if picture is a dog
Y(i,1) = 1 if picture is a cat
Y(i,2) = 1 if picture is a elephant
etc.

Our Goal: To classify items.

We have this: (X,Y)

xi $\longrightarrow$ Model: $h_\theta(x)$

Actual Results:

$y_i = [\, y_{i,1}, y_{i,2}, ..y_{i,k} \,]$
$y_i = [\, 0 , 1, .. 0 \,]$

Machine Learning Steps to train a classifier model

1. Choose model: $h_\theta(x)$ = estimate of Y
2. Define a loss function (J(θ)) = which is a function of **f**(Y_actual, Y_estimated)
3. Optimize across the parameter space (θ) to minimize the loss function

Linear Regression for Classification? (Not so good!)

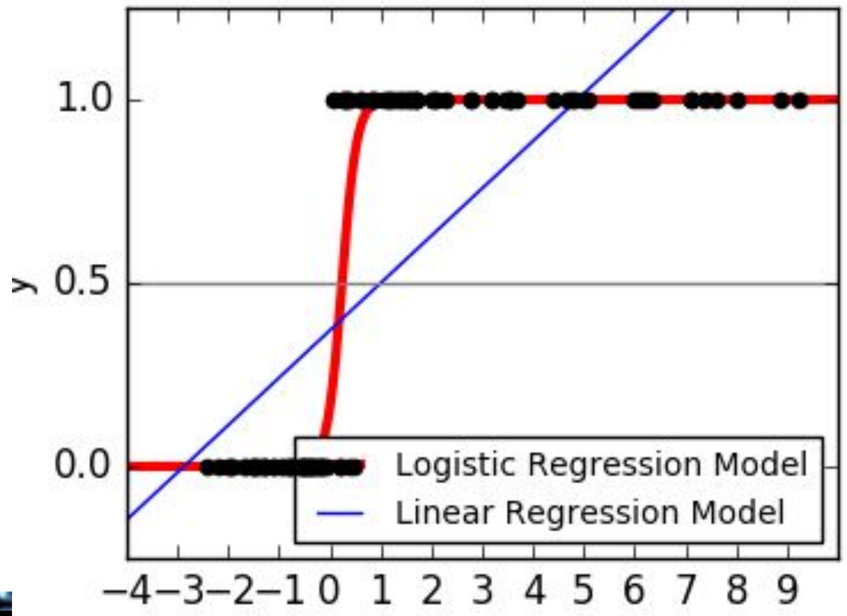
Why not choose a Linear model for classification?

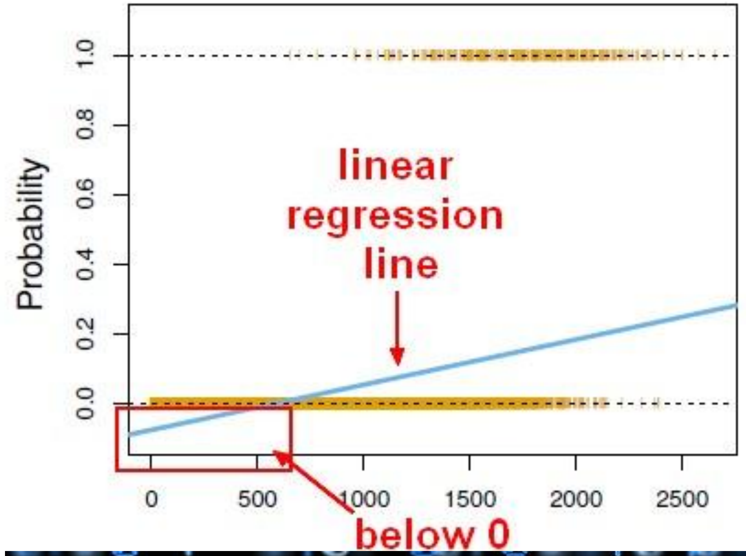
Because a line is not a good estimator for binary results (classification)

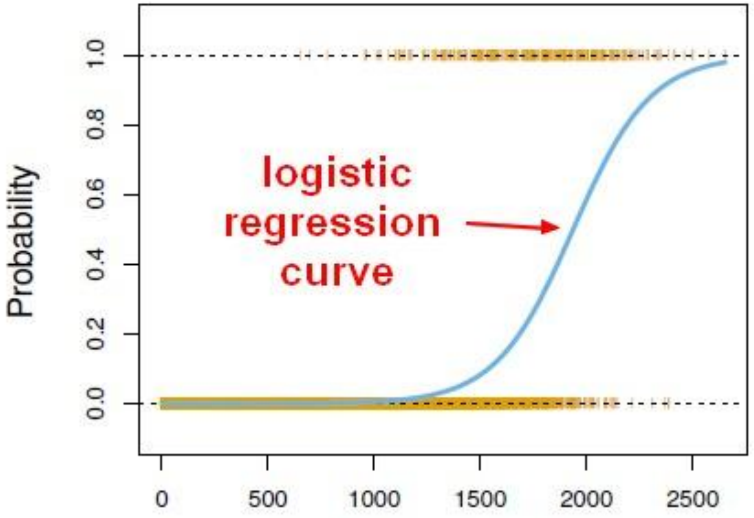
Linear model: $f(x,\theta) = h_\theta(x) = \theta x_i = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \ldots$

Negative probabilities and biased towards majority class

Instead we use Logistic Regression!







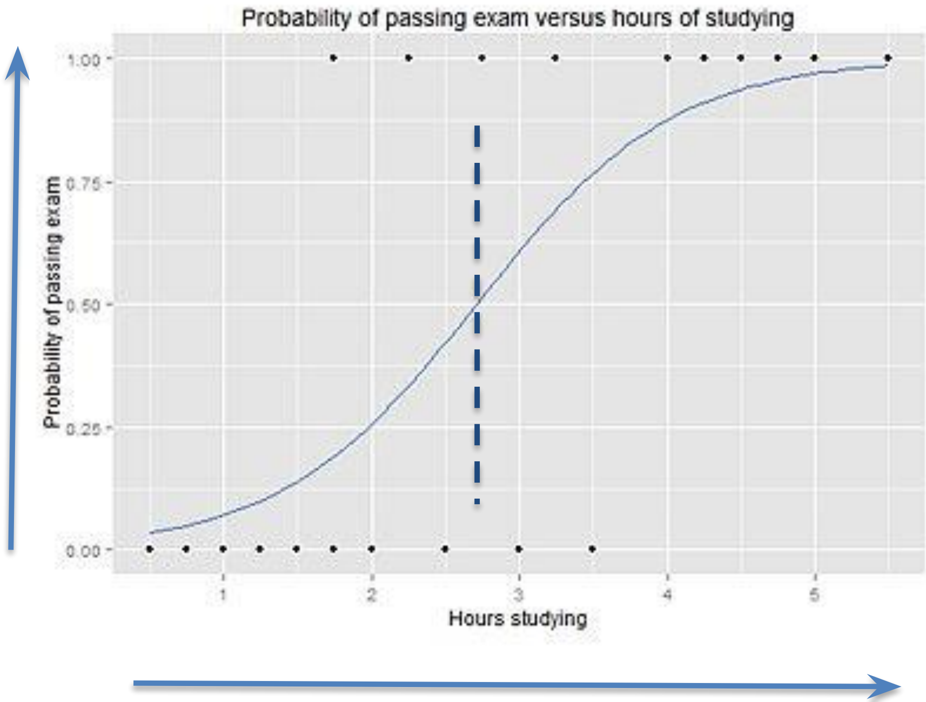
Logistic Regression

Classification Example

Data: students study for an exam
(x= hours studied, y = pass/not pass)

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

y, binary output
0 = fail
1 = pass



x

Problem:
We want to find a model that can predict the probability that the student passes given x hours of study

$$h_\theta(x) = P(y = 1|x; \theta)$$

If Prob >= 0.5, predict student will pass, y=1
If Prob < 0.5, predict student will fail, y=0

$$P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)$$

The logistic / sigmoid function

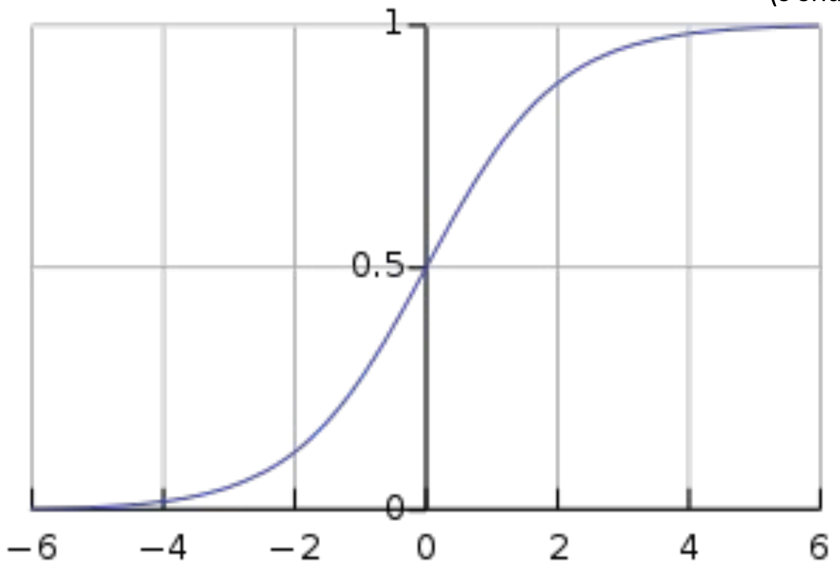
The sigmoid function:

$$z(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large positive t
z -> 1
Large negative t
z -> 0

This function only evaluates to values between 0 and 1 for all real numbers (like a probability)

(s shaped curve)



t, sum of weighted inputs + bias
$t = \theta_0 + x_1\theta_1$ →

$$z(t) = z(\theta^T x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}} = h_\theta(x)$$

If θ_1 is small → slow rise
If θ_1 is large → fast rise

- **$z(\theta x)$ is the probability that y = 1 given any x**
- The **decision boundary**, where the probability = 50%
- **$z(\theta x)$ = ½** when $e^{-(\theta_0 + x_{i,1}\theta_1)} = 1$, ie $\theta_0 + x_1\theta_1 = 0$

Decision Boundary

The decision boundary separates our predicted categories from one another, in the feature space.

If we have two inputs, x_1 and x_2, the decision boundary is the line when the predicted probability for y=0 and y=1 is equal to 50%

$$h_\theta(x) = z(\theta^T X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}} = 0.5$$

$$\Leftrightarrow$$

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

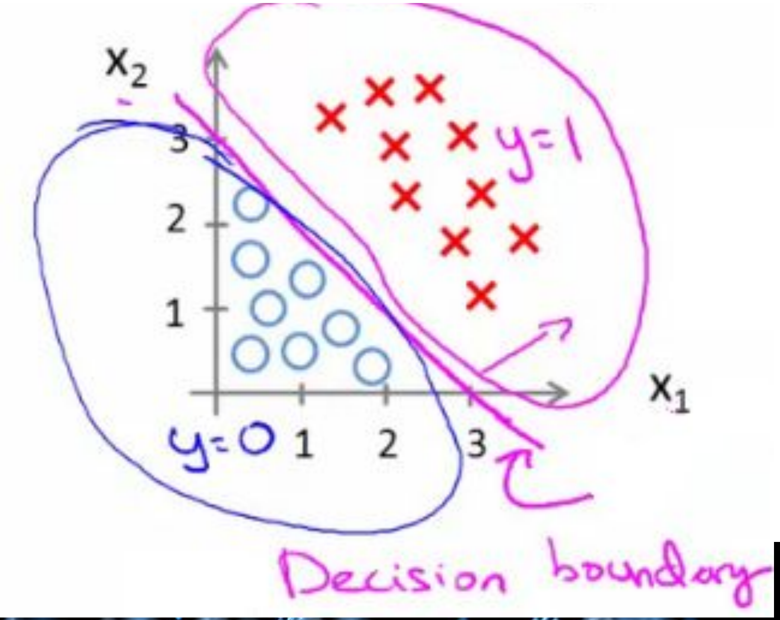
Example

θ_0 = -3 θ_1 = 1 θ_2 = 1

Then $x_1 + x_2 - 3 \geq 0$

will predict y=1 and vice versa

(see example below)



Data X

Derivation of the logistic cost function

- How to choose parameters θ to find the best $h_\theta(x)$
- We need a **cost function J(θ)** that measures performance, then find best θ.

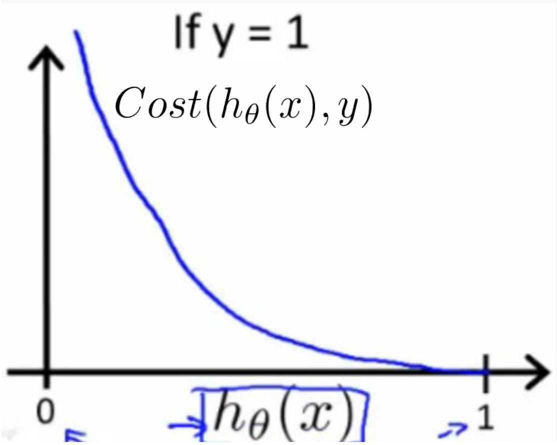
- Output y is binary and can only take on two values (0 or 1)
- Construct **J(θ)** that penalizes wrong predictions

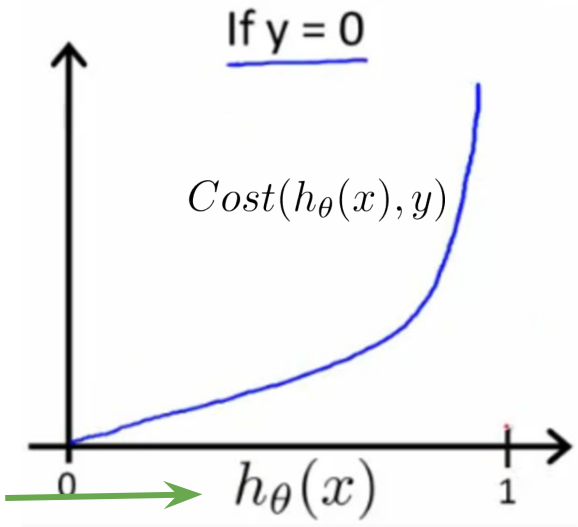
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } y = 1$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } y = 0$$

Cost plotted against predicted class probability when the true value is y=1 (top) or y=0 (bottom)





$$\frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}} \rightarrow h_\theta(x)$$

Logistic cost function

Cross Entropy for binary classification =

$$Cost(h_\theta(x), y) = -ylog(h_\theta(x)) - (1-y)log(1-h_\theta(x))$$

Actual output

y

Estimated output

$h_\theta(x)$

Note: Loss Function on the former slide can be added to form cross entropy for binary classification.

This cost function can be derived from the Maximum Likelihood estimation of the parameters.

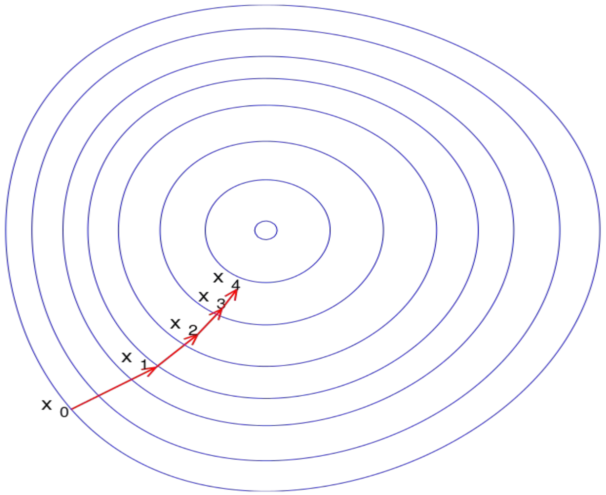
Gradient Descent & Logistic Regression

J(θ) = is a cost a function comparing our estimate $h_\theta(x)$ and the true y.

Find optimal θ (first initialize θ with some random value)

Take small steps in the direction where J(θ) is decreasing

Update rule: $\theta_{j+1} = \theta_j$ – [(step size α) x gradient of **J(θ)**]



Formal update rule

looks exactly like Linear Regression, but note that $h_\theta(x)$ has changed)

$$J(\theta) = \frac{-1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Same as:

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

X Data

Multi-class Logistic Regression: *One-vs-all* $y \in \{0,1...k\}$

Sigmoid function:

$$z(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large t -> 1, Small t -> 0

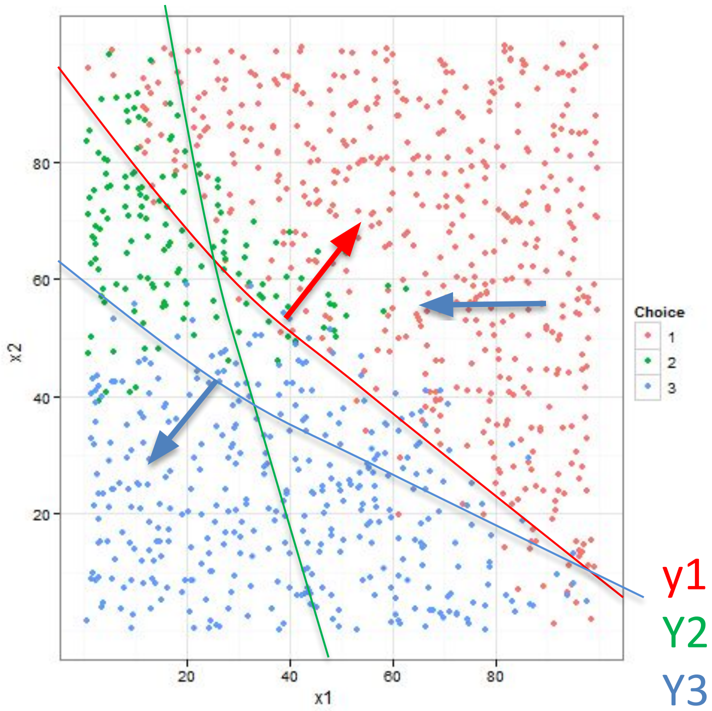
And if t has this form
$t = \theta x_i$ (in matrix form) $= \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \ldots$

$$z(t) = z(\theta^T x) = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1 + e^{(-\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots)}}$$

- Easily extends to multiple features (x1, x2, x3..)
- And multiple parameter weights

One-vs-all

- Take i:th class (against all other grouped into an alternative class), create decision boundary and calculate probability
- Final prediction will be the class that had the highest probability against all others.



y1 boundary
Y2 boundary
Y3 boundary

$$h_\theta^{(0)}(x) = P(y = 0|x; \theta)$$

$$h_\theta^{(1)}(x) = P(y = 1|x; \theta)$$

...

$$h_\theta^{(k)}(x) = P(y = k|x; \theta)$$

$$prediction = max_i(h_\theta^{(i)}(x))$$

Read about Softmax Regression for Multiclass Classification

p. 139 - 142 in the Textbook

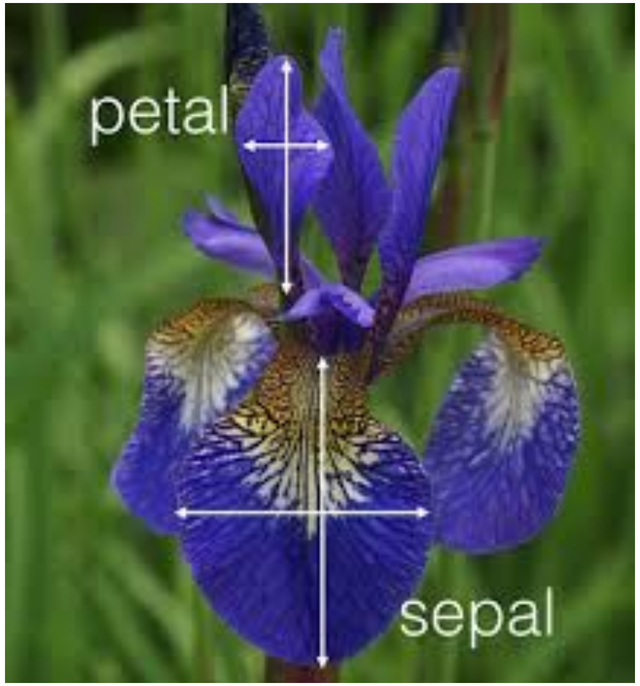
End of Section

References

- The material presented in this lecture references lecture material draws on the materials the following courses:
- UC Berkeley – CS 294-129 (Designing, Visualizing, and Understanding Deep Neural Networks): https://bcourses.berkeley.edu/courses/1453965/pages/cs294-129-designing-visualizing-and-understanding-deep-neural-networks
- Stanford – CS231n (Convolutional Neural Networks for Visual Recognition): http://cs231n.stanford.edu/
- Stanford – CS229 (Machine Learning) & Andrew Ng's Machine Learning at Coursera: http://cs229.stanford.edu/ & https://www.coursera.org/learn/machine-learning

Example Code: Logistic Regression in Scikit-learn

Example Code Sample with Logistic Regression Classifier



Input data

X: Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Y:
0 = 'setosa',
1 = 'versicolor',
2 = 'virginica’

```
print type (X)
print X[0:5]

<type 'numpy.ndarray'>
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]


print Y[0:5]
[0 0 0 0 0]
```

Example Code Sample with Logistic Regression Classifier

```
  import numpy as np
1 from sklearn import linear_model, datasets

  X = iris.data[:, 1:3]  # only the first two features.
  Y = iris.target

  # https://en.wikipedia.org/wiki/Logistic_regression
2 logreg = linear_model.LogisticRegression(C=1e5)

  # we create an instance of Neighbours Classifier and fit the data.
3 logreg.fit(X, Y)

  # predict a category for every row in X
4 Z = logreg.predict(X)
```

Class sklearn.linear_model.LogisticRegression

(penalty='l2',
dual=False,
tol=0.0001,
C=1.0,
fit_intercept=True,
intercept_scaling=1,
class_weight=None,
random_state=None,
solver='liblinear', max_iter=100,
multi_class='ovr', verbose=0,
warm_start=False,
n_jobs=1)

http://scikit-learn.org/stable/modules/generatedsklearn.linear_model.LogisticRegression.html

* Z[2] will be the predicted number for row X[2]

Code Samples with SciKit Learn

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
# numpy.ravel: Return a contiguous flattened array.
```

xx shape is (171, 231)
yy shape is (171, 231)

np.c returns shape (39501, 2)
[[3.8 1.5]
[3.82 1.5]
[3.84 1.5] …]
Z shape is (39501,)

1.5 yy -> X[:,1] 4.9

3.8 / xx-> X[:,0] / 8.4

3.8, 1.5	3.8, 1.7	3.8, 1.9
….		
4.0, 1.5	4.0, 1.7	4.0, 1.9….
4.2, 1.5	4.2, 1.7	4.2, 1.9….
…		

xx is a matrix of all the first values

1.5 4.9

3.8 / 8.4

3.8	3.8	3.8 ….
4.0	4.0	4.0 ….
4.2	4.2	4.2….
…		

yy is matric of only the second values

1.5 yy -> X[:,1] 4.9

3.8 / xx-> X[:,0] / 8.4

1.5	1.7	1.9 ….
1.5	1.7	1.9 ….
1.5	1.7	1.9….
…		

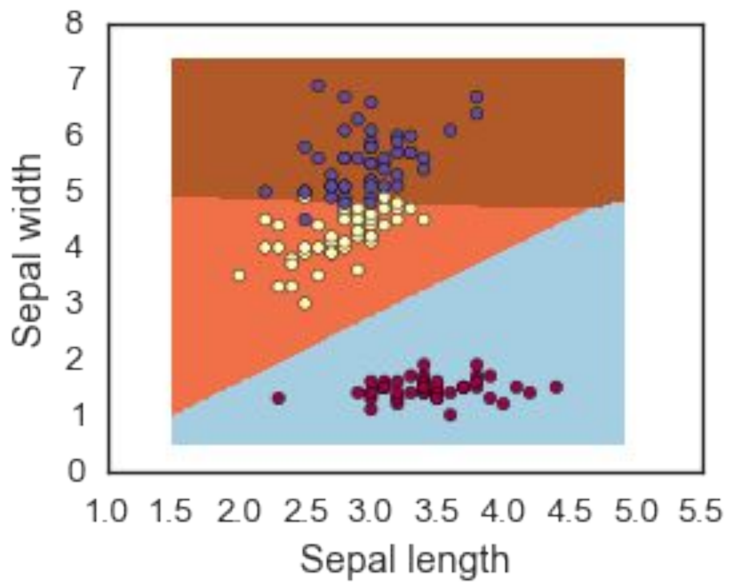
Plotting the Results

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k',
cmap=get_cmap("Spectral"))
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

#plt.xlim(xx.min(), xx.max())
#plt.ylim(yy.min(), yy.max())
#plt.xticks(())
#plt.yticks(())

plt.show()
```



Methods for LogisticRegression

Methods

`decision_function`(X)	Predict confidence scores for samples.
`densify`()	Convert coefficient matrix to dense array format.
`fit`(X, y[, sample_weight])	Fit the model according to the given training data.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict class labels for samples in X.
`predict_log_proba`(X)	Log of probability estimates.
`predict_proba`(X)	Probability estimates.
`score`(X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.
`sparsify`()	Convert coefficient matrix to sparse format.
`transform`(*args, **kwargs)	DEPRECATED: Support to use estimators as feature selectors will be removed in version 0.19.

fit(*X*, *y*, *sample_weight=None*) [source]

Fit the model according to the given training data.

Parameters: **X** : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vector, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape (n_samples,)

Target vector relative to X.

sample_weight : array-like, shape (n_samples,) optional

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: sample_weight support to LogisticRegression.

Returns: **self** : object

Returns self.

predict(*X*) [source]

Predict class labels for samples in X.

Parameters: **X** : {array-like, sparse matrix}, shape = [n_samples, n_features]

Samples.

Returns: **C** : array, shape = [n_samples]

Predicted class label per sample.

Fit and predict from ScikitLearn

Regularization

Why: To avoid over-fitting

How: You penalize your loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of your weights vector w

Your new loss function $= L(X,Y) + \lambda N(w)$

Tuning the regularization term λ: Cross-validation:

- divide your training data,
- train your model for a fixed value of λ and test it on the remaining subsets
- repeat this procedure while varying λ.
  Then you select the best λ that minimizes your loss function.

Shrinkage Methods II: An example