About Me:

Data ^{X}

Gradient Descent & Classification

Alexander Fred Ojala, Ikhlaq Sidhu, Kevin Bozhe Li

Alexander Fred Ojala
Visiting Scholar, IEOR UC Berkeley
afo@berkeley.edu

Lecture 5 pt.1: Outline

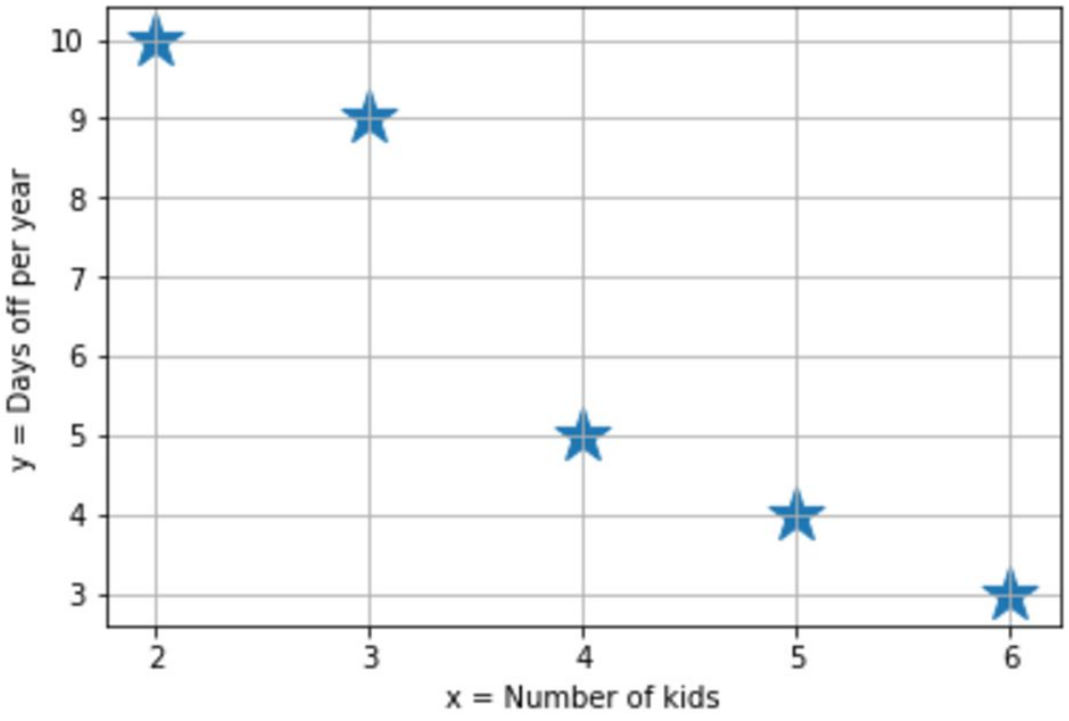
1. Linear Regression recap
2. Gradient Descent
3. Feature scaling
4. Intro to Classification
5. Logistic Regression
6. Example Code

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

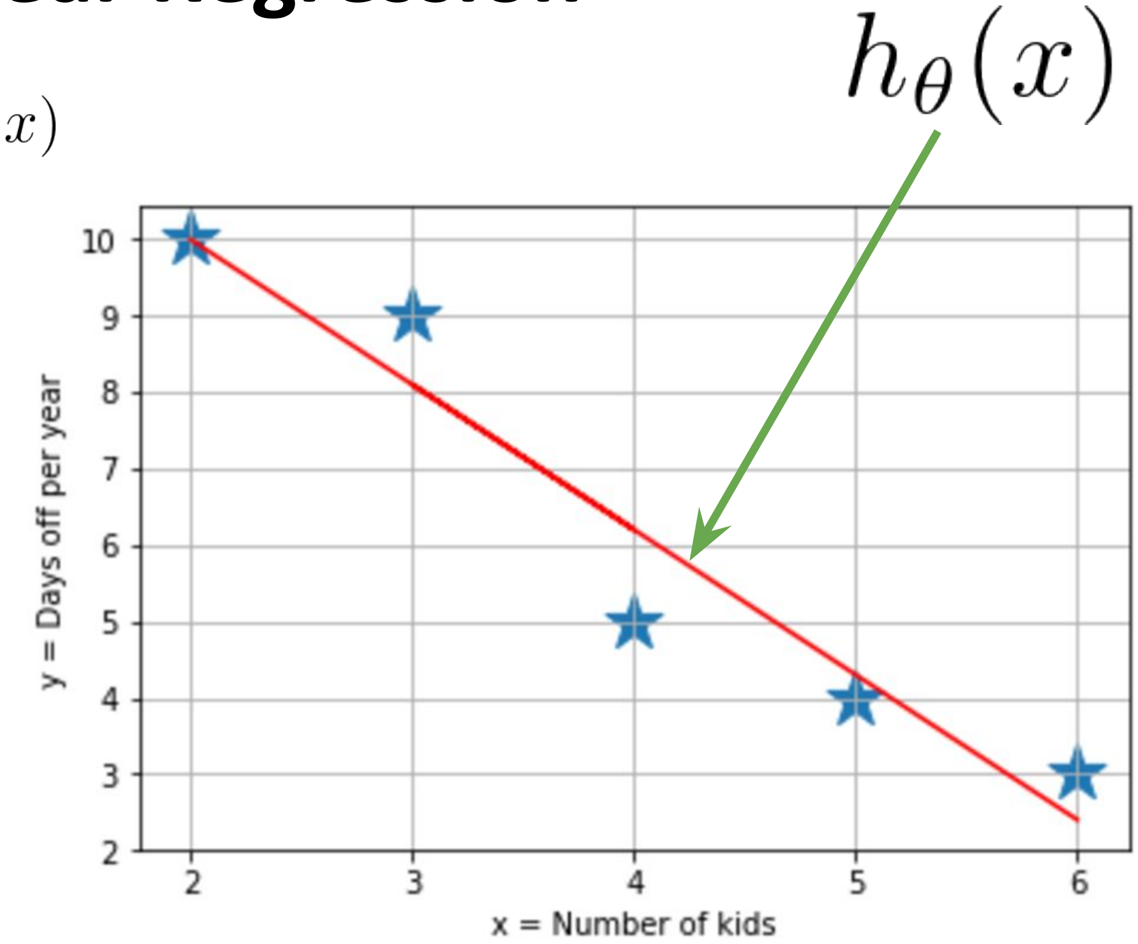
Simple Linear Regression: Prediction hypothesis $h_\theta(x)$

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

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

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

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



Recap: Multiple Linear Regression

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

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

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

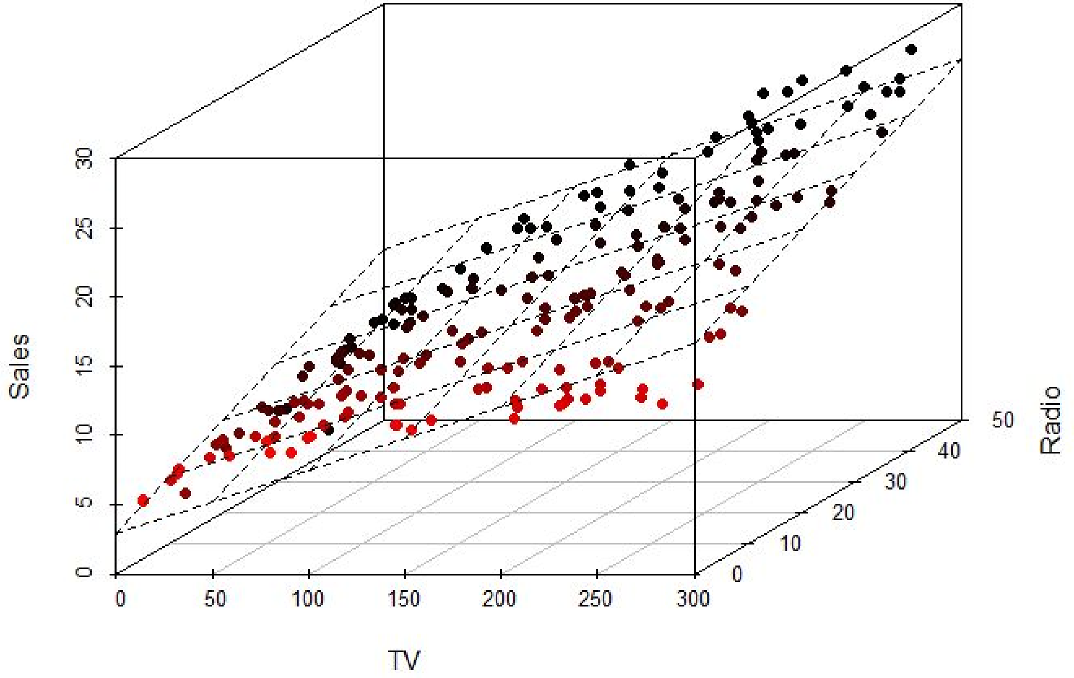
$$h_\theta(X) = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ . \\ . \\ h_\theta(x^{(m)}) \end{bmatrix} \text{ is the hypotheses vector}$$

Example of multiple linear regression (2 features)

x_1 = TV advertising

x_2 = Radio advertising

y = Sales



Recap: Cost function (MSE)

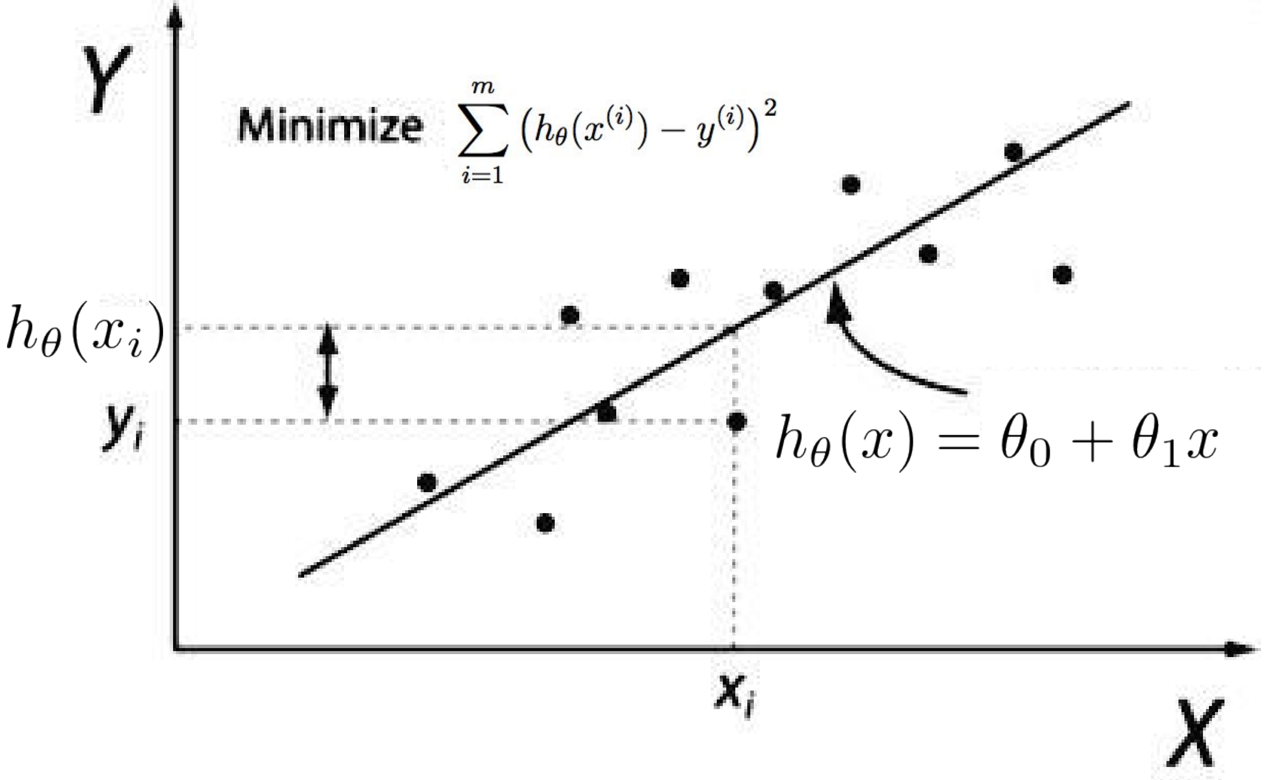
Simple Linear Regression

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

Cost function:

Measures how good the fit is (MSE)

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



Recap: Minimize cost function

Optimal parameters are found when the cost function / the error J(θ) is minimized

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

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

$$\min_\theta \; J(\theta)$$

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

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

Normal equation for Linear Regression

Closed form, analytical solution. Finds θ that minimizes J(θ)

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

Pros:

- Finds optimal answer with one calculation
- Really quick for small data sets

Cons:

- $\mathcal{O}(n^3)$ complexity, ***slow,*** because of matrix inverse
- $(X^TX)^{-1}$ might not be invertible (can be solved by taking pseduo-inverse)

Data X

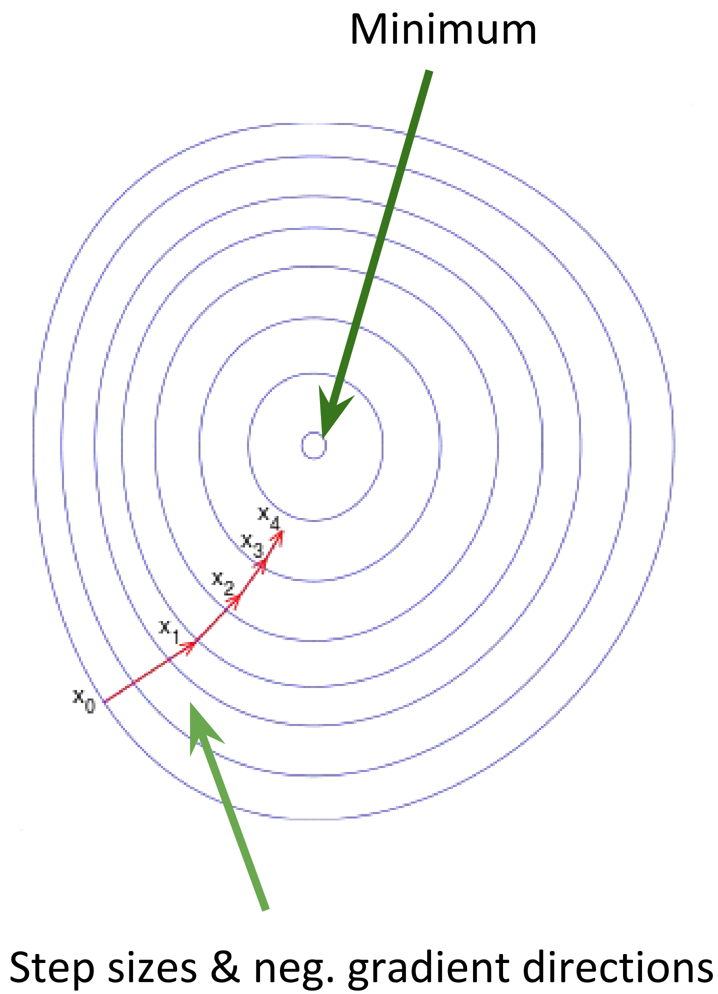
Gradient Descent

Introducing Gradient Descent

WIKIPEDIA:

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 of the gradient (or approximate gradient) of the function at the current point.



Introducing Gradient Descent

Alternative way of minimizing the cost function:

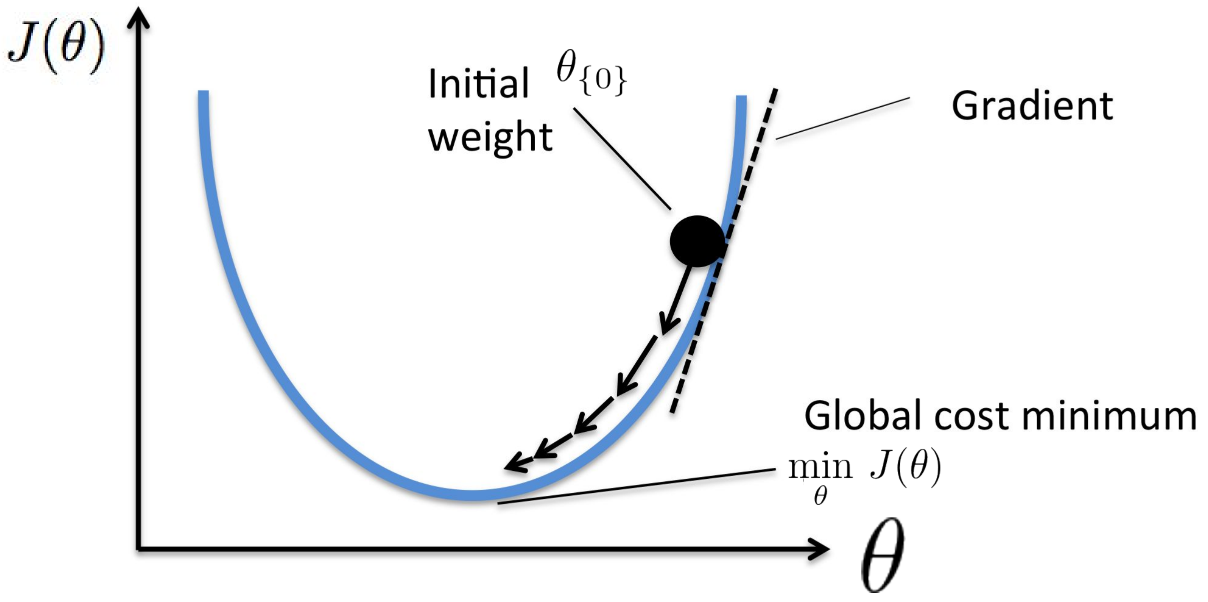
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- ***Gradient Descent minimizes J(θ) iteratively,***
- ***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 gradient** $\frac{\partial}{\partial \theta_j} J(\theta)$ 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** $\theta_1 .. \theta_n$ **at once**

   (this is called "batch" Gradient descent)

5. Stop when the error has converged.

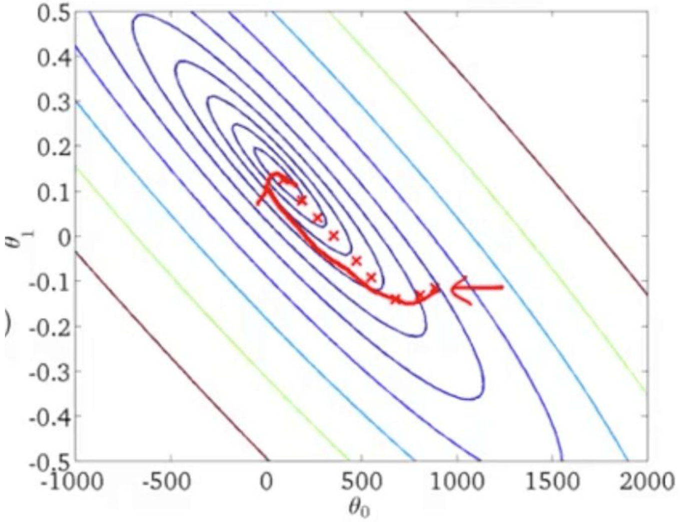
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

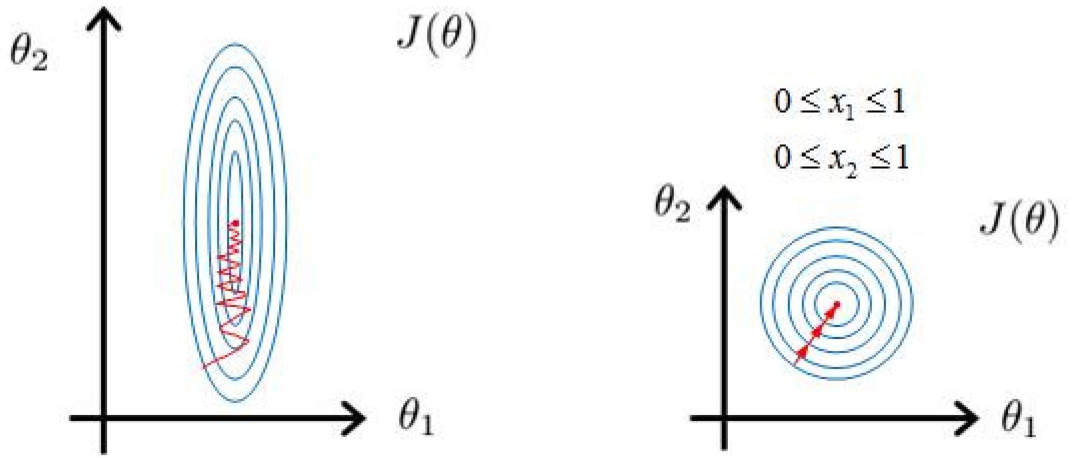
Feature Scaling

Gradient Descent will be more likely to converge and be faster if the features are scaled, ie $-1 \leq x_i \leq 1$

For all features:

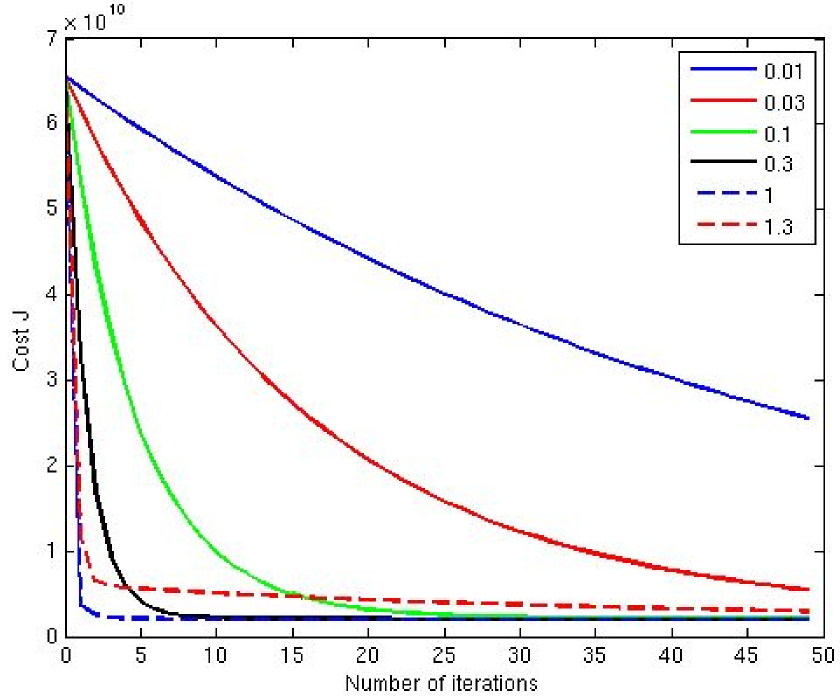
- Subtract mean
- Divide by st.dev.

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



Monitor convergence

Plot the error function *J(θ)* every iteration. *Check that the error becomes smaller. Also, plot this for different learning rates to find a suitable one.*



Gradient Descent Pros / Cons

Pros

- **Will always converge to minima** if learning rate α is chosen correctly
- **Fast** (time complexity is $\mathcal{O}(n)$)

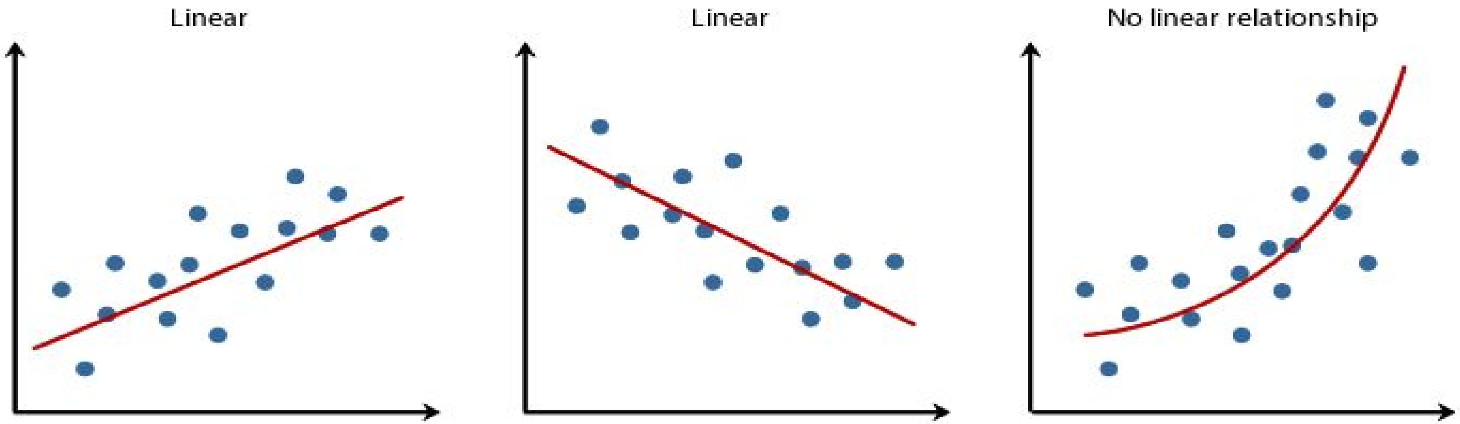
Cons

- **We have to choose a learning rate** α and initialize the parameters
- **Often takes A LOT of iterations** to reach global minima of the cost / objective function

Classification

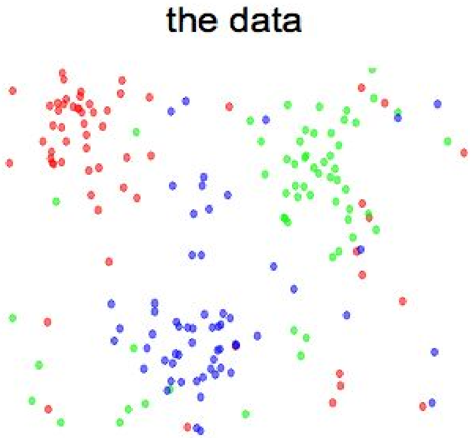
Regression vs. Classification

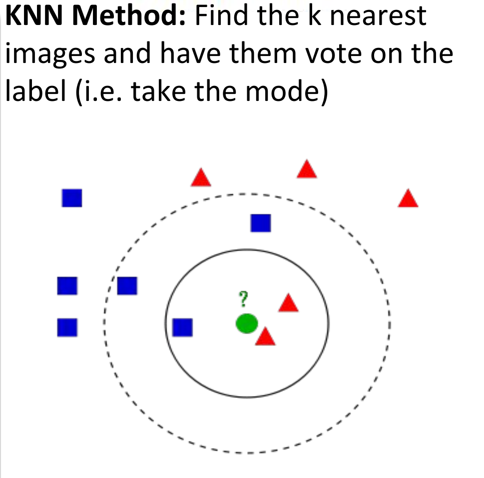
Regression:
- Continuous output y
- Quantitative approach
- Linear or Non-linear

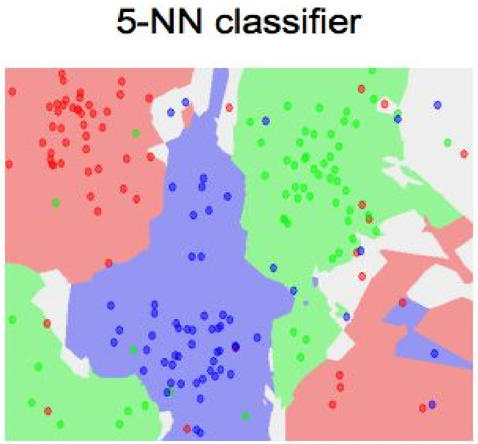


Classification:
- Discrete output y
- Qualitative approach
- Linear or Non-linear

Ex. KNN, K-means
Logitstic, SVM, ..



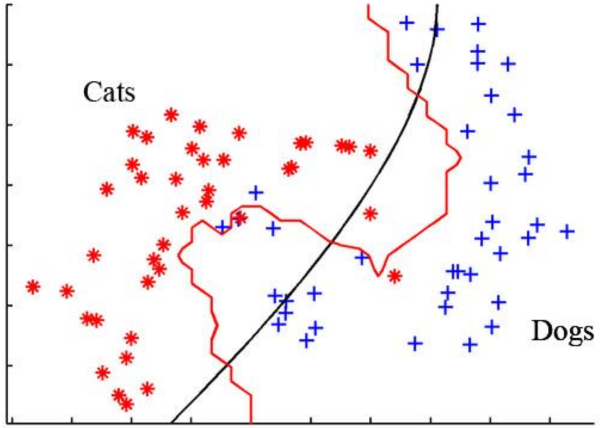


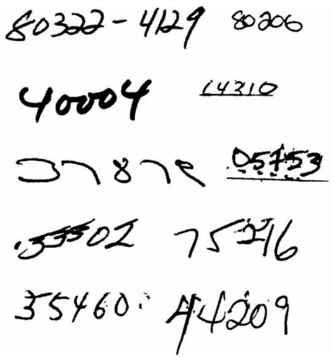


Examples of classification

Examples

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





Our Goal: To classify items

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

xi $\xrightarrow{\hspace{3cm}}$

Model

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

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

- xi is a vector (or even matrix) for each data element
- **Example:** xi = [12 15 22] = [height, weight, color]
- **For a picture:** x_i = [32 x 32 x 3]: array of numbers

Binary classification (cat vs dog):

y = 1 if picture is dog

Y = 0 if picture is cat

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

Multi-class classication:

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

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

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

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.

** Sometimes: y_i = [+1 , -1, .. -1]*
-1 or 1 instead of 0 or 1

Our Goal: To classify items.

We have this: (X,Y)

xi → Model: $h_\theta(x)$

Actual Results:

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

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

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 (Y_atual, vs Y_estimated)
3. Optimize across the parameter space (θ) to minimize the loss function (to some small threshold)

Linear Regression for Classification? (Not so good!)

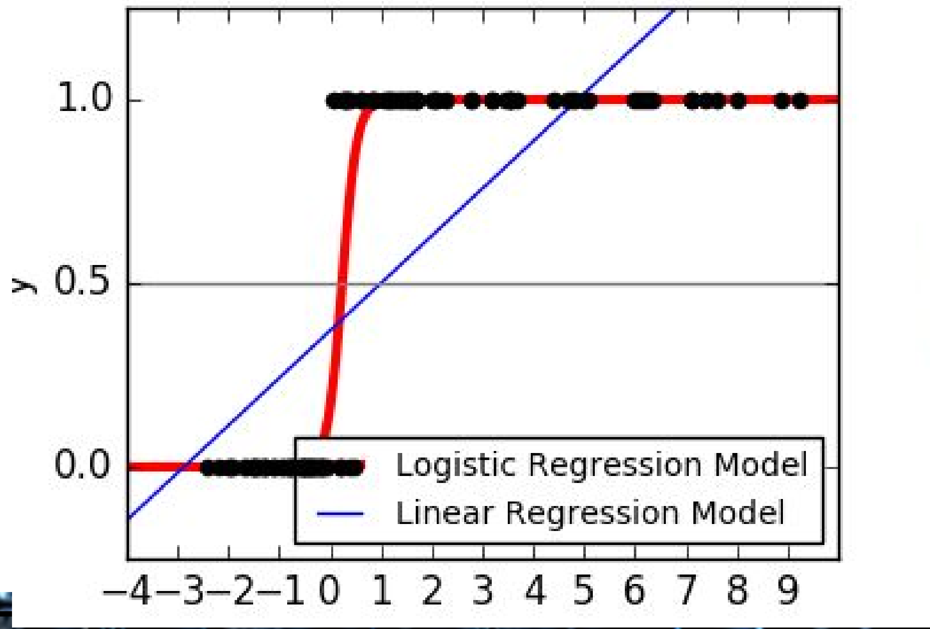
Why not choose a Linear model for classification?

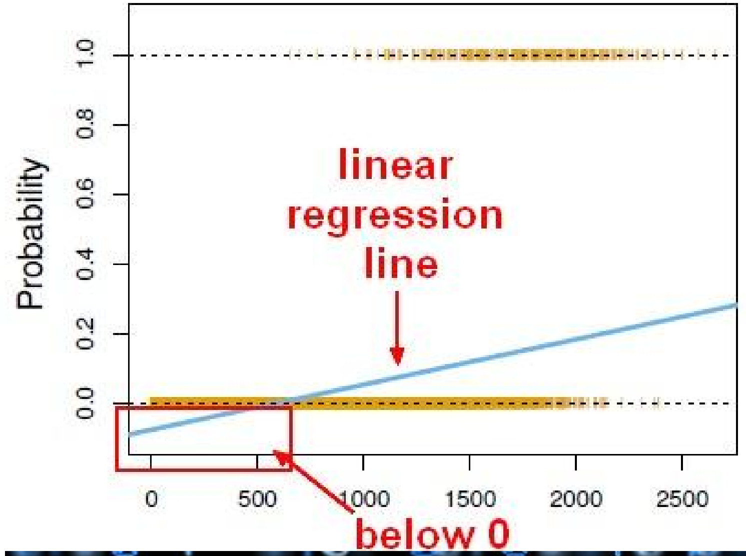
Because a line is not a good estimator for binary results (classification)

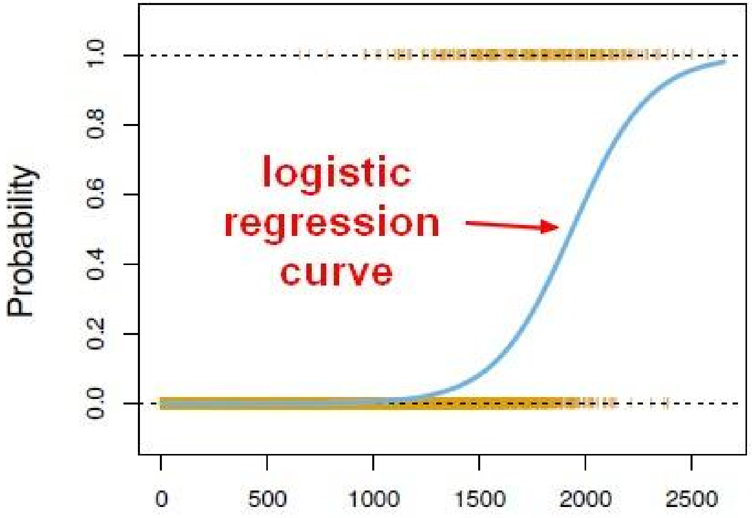
Linear model: $f(x,\theta) = h_\theta(x) = \theta x_i = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \dots$

Results in a ***different estimation of Yi for each sample point x_i***

Instead we use Logistic Regression!







Data X

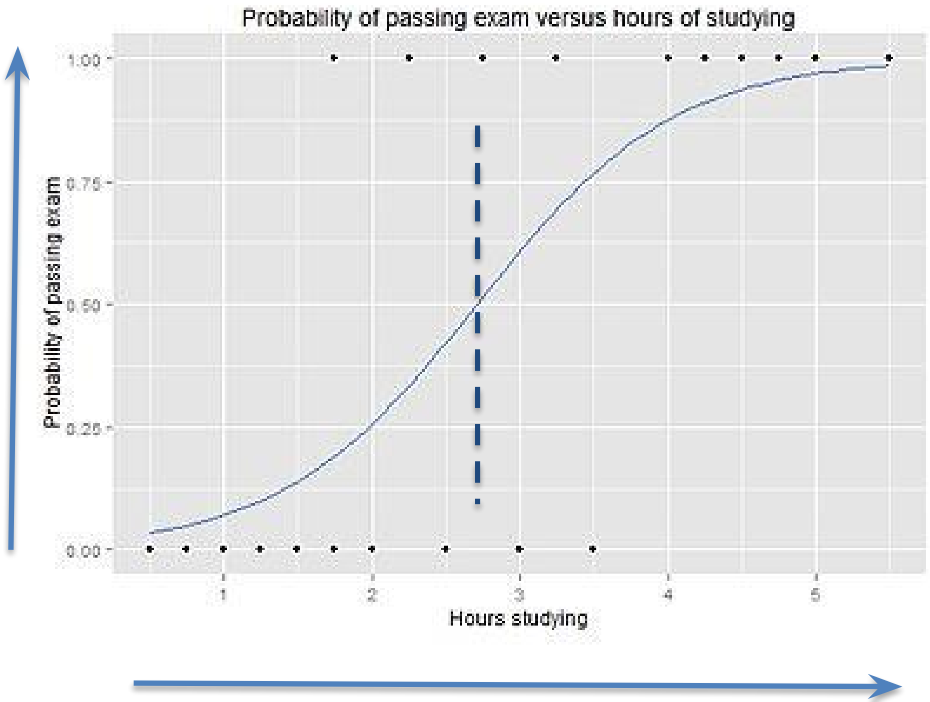
Logistic Regression

Example Classification with Logistic Regression

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:
Student studies x hours
We want to predict will the student pass?

We use this curve to predict the probability that the student would pass given x hours of study

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

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

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

The logistic / sigmoid function is a better fit to predict binary outcomes

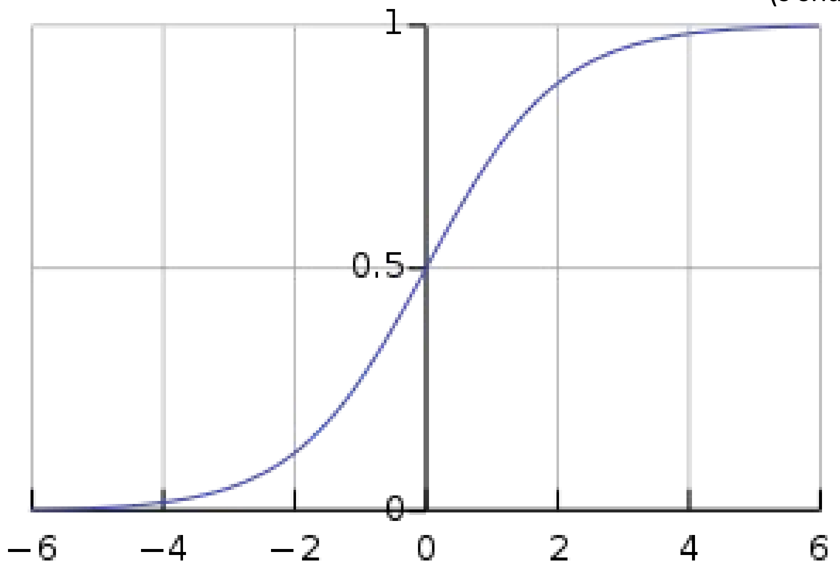
The sigmoid function:

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

Large t -> 1
Small t -> 0

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

(s shaped curve)



And if t has this form:
$t = \theta_0 + x_1\theta_1$ (a line)

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

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

- **Think of z(θx) as the probability of y = 1 given any x**
- Prob (y=1) = ½ when $e^{-(\theta\_0 + xi,1\ \theta\_1)} = 1$, ie $\theta_0 + x_1\theta_1 = 0$
- Choose parameters θ to get best fit
- Still need a **cost function J(θ)**, then solve for best θ

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 either y=0 or y=1 equals 50%

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

$$\Leftrightarrow$$

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

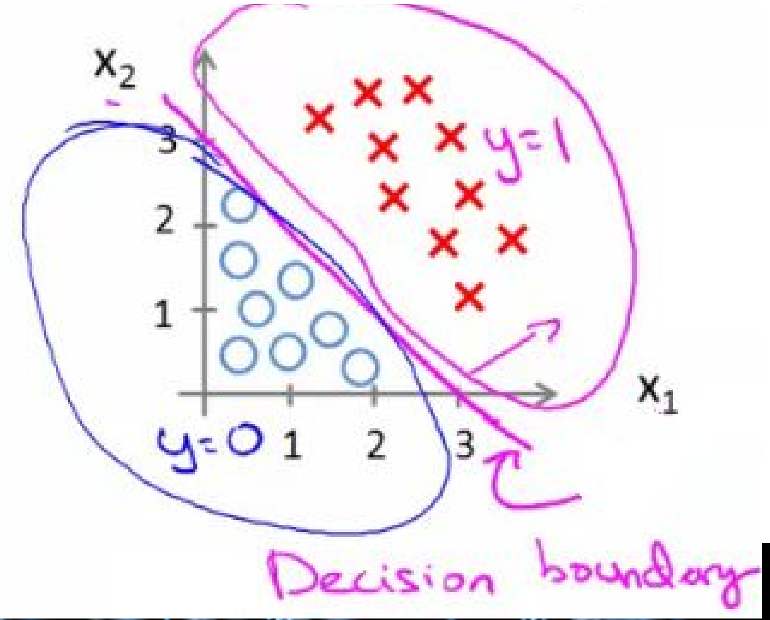
Example

$\theta_0 = -3 \quad \theta_1 = 1 \quad \theta_2 = 1$

Then $x_1 + x_2 - 3 \geq 0$

will predict y=1 and vice versa

(see example below)



Derivation of the logistic cost function

Intuition:

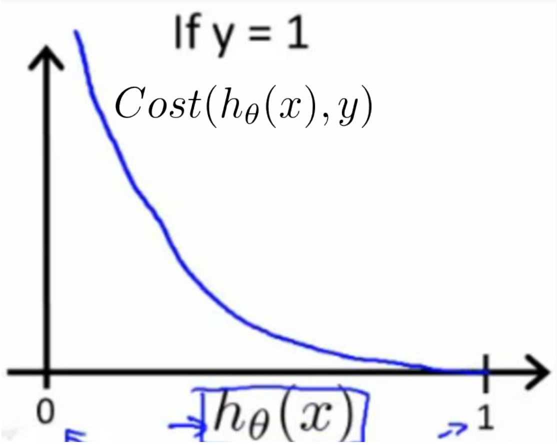
- How do we get the optimal decision boundary?
- Output y can only take on two values (0 or 1)
- We want a cost function that penalizes when our prediction $h_\theta(x)$ is wrong

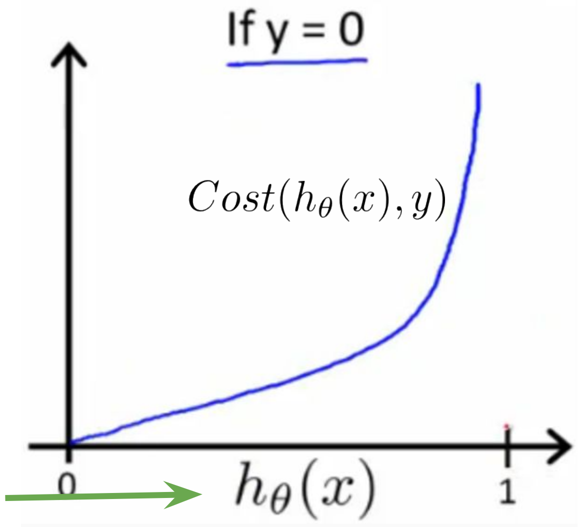
Our total overall cost is J(θ)

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

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

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





$$\frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

Logistic cost function

Cross Entropy =

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

Actual output

y

Estimated output

$h_\theta(x)$

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

We choose this cost function, because it can be derived from the Maximum Likelihood estimation of the parameters.

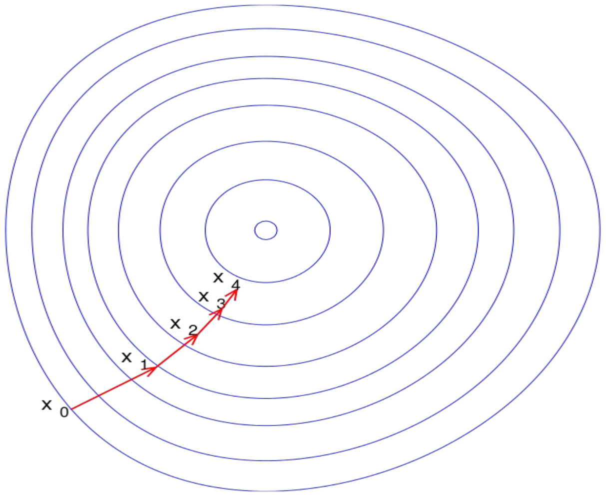
Gradient Descent & Logistic Regression

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

Try to find optimal θ (first initialize 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 {

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

}

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 = θx_i (in matrix form) = $\theta_1 x_{i,1} + \theta_2 x_{i,2} \dots$

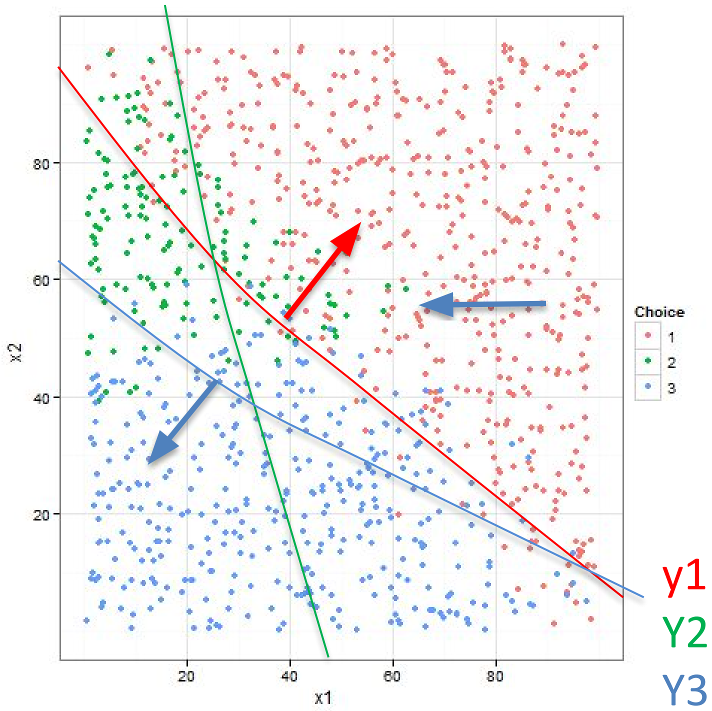
$$z(t) = \qquad z(\theta^T x) = h_\theta(x) =$$

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

- 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
- Choose the class that had the highest probability against all others.



y1 boundary
Y2 boundary
Y3 boundary

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

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

...

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

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

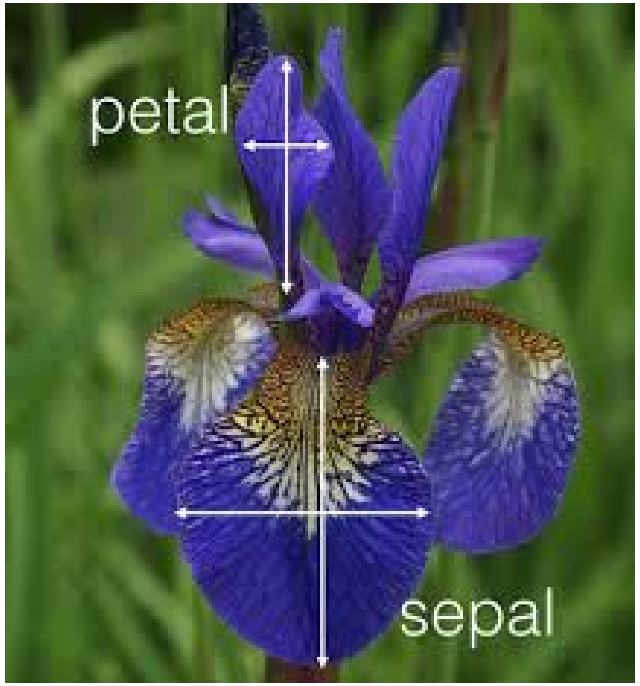
End of Section

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]
```

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….
…		

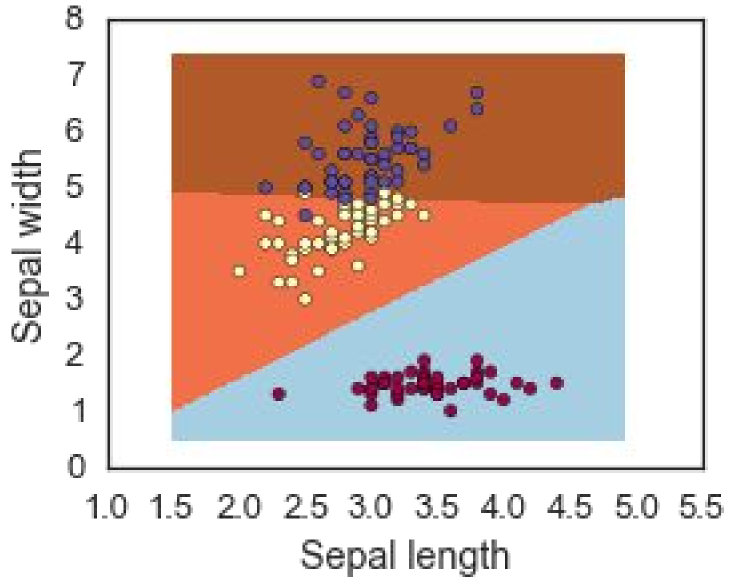
Plotting the Results

```
# 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()
```



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)
```

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

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

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 and predict from ScikitLearn

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.

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