**Linear Regression**

**Notation**  
*Feature: An independent variable which affects the predicted dependent variable (E.g. floor space & age of a house are features to consider when predicting the price of a house.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MATRIX X | 0 (space) | 1 (age) | j | … | n |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| i |  |  |  |  |  |
| … |  |  |  |  |  |
| m |  |  |  |  |  |

is the value of the jth feature of the ith training data point

is the ith training data point

is the vector containing the values of the jth feature for each training data point

**Hypothesis Function (h):**

* The function generated by the model to predict the output of a given input
* h can be an equation or a graph
* E.g. for a univariate linear regression, +
  + h predicts a value of y for all values of x
* E.g. for multivariate linear regression,
  + Where h(x) is based on [, each to are features and = 1 (for convenience of notation)
  + x = [ (vertical vector)
  + (vertical vector)

**Cost Function (J):**

* Function to minimize to ensure best fit of model
* Example (equation representation of cost function):
  + Univariate Linear Regression cost function:
  + General Linear Regression cost function:
  + Sum of Squared Errors cost function

**Gradient Descent Algorithm**

* Used to minimize cost function by selecting values of for j = [0, n]

*Mathematical Definition*

Repeat until reaches equilibrium (convergence)

is the learning rate, higher means that the algorithm takes larger steps

For all j = [0, n]; updated simultaneously (move all values of together [see below])

Examples of

* For multivariate linear regression:

Properties

* Converges to local minimum
* If is too small, the algorithm takes too long
* If is too large, the algorithm might not converge or even diverge
* As decreases as it approaches the local minimum, the gradient descent automatically takes smaller “steps”, thus there is no need to decrease over time.

*Python Implementation*

arr = [] #arr[j] contains

a = 1.0

while True:  
 temp = []

for j in range(n):

temp[j] =

#Use arr[j], not temp[j]

arr = temp.copy()

*Gradient Descent Speed-Ups*

* Feature scaling
  + Ensure that all features are on a similar scale/range (does not necessarily need to be the same.
  + Generally, move all features into a [-1, 1] or [0, 1] range
    - features in [-3, +3] or [-0.75, 0.75] (near [-1,1]) are generally also fine
  + Allows gradient descent to converge more quickly
* Mean normalization
  + Ensure that mean of is 0 (for j in [1, n])

To apply both feature scaling and mean normalization, a common technique is to apply:

On all feature vectors for j in range [1, n]

* Where and are the mean and range of respectively

*Debugging Gradient Descent*

* To ensure that gradient descent is working properly, note the value of J() at each iteration of gradient descent
* It would be good to plot for the graph of J() against the number of iterations (also helps to see if gradient descent has converged
* If J() increases with each iteration or fluctuates, it might be due to too large of an alpha value
  + It can be proven that with a small enough alpha, J() should decrease with each iteration
* Might need to binary search n for 3\* 10^n for alpha
  + [0.003, 0.003, 0.03 … 300, 3000]

Choosing features

* It can sometimes be advantageous by combining features together to get a more optimal model which takes less iteration to fit
* One can create new features from existing features to perform polynomial regression with linear regression machinery
  + E.g. the hypothesis for a univariate quadratic regression is h(x) =
  + To apply linear regression algorithms to this, one can use the multivariate linear regression model h(x) = while setting
  + This also works by combining multiple features together
    - One can create a new feature

**Normal Equation**

Where X is the matrix shown at the top and Y is the dependent variable in column matrix format.

* Used as an alternative non iterative approach to gradient descent
  + No need to find a good value of alpha
  + No need to perform iterations
  + BUT has O(n^3) complexity (due to inverse matrix function) so not good for x >= 10000

Note:

* If is non-invertible
  + There may be redundant / overlapping features
  + There may be too many features (n >= m)

**Classification Problems**

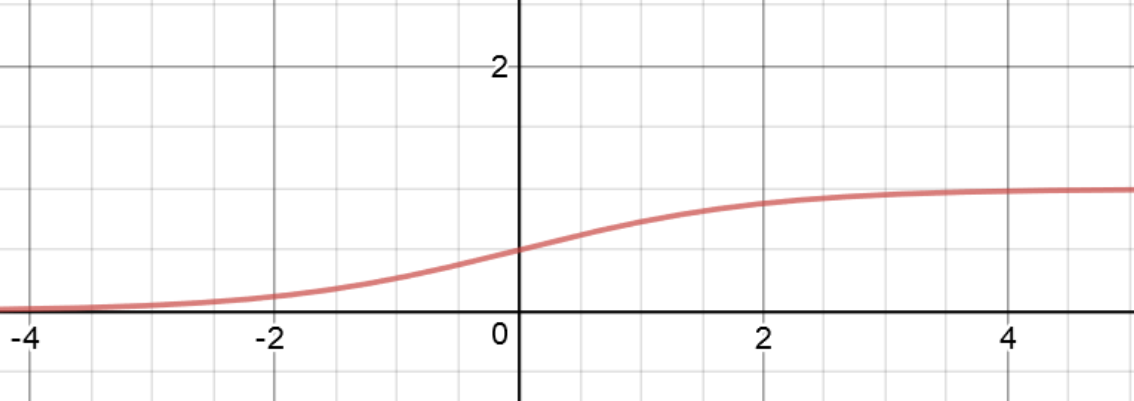
Classes

* Binary-class classification
  + Got rat shit / no rat shit
* Multi-class classification
  + What type of rat shit

**Binary-class classification**

Hypothesis

* Where g is the sigmoid/logistic function ; a graph which has asymptotes at y=0 and y=1 and returns a value in range (0, 1)

 Graph of sigmoid function

* is the probability that y = 1 (positive class)
* = 1 -

Properties of hypothesis

* Predict that y=1 when
* when
* when

Cost Function

* Cost function for each training example
* Overall cost function (to be minimized)



Gradient Descent for binary classification

* Algorithm looks identical to linear regression, except the hypothesis function is different

**Alternatives to gradient descent**

* Conjugate gradient
* BFGS
* L-BFGS

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| No need to pick alpha | Complex, hard to self-code |
| Generally converges faster than gradient descent |  |

**Scaling Binary-Class Classification to Multi-Class Classification: One vs Rest with logistic regression**

* For each class i
  + Apply logistic regression with i as the positive class and the rest as the negative class
  + 🡪 leaving us with a hypothesis which gives the probability that x is of class i

**Neural Networks**

Used for non-linear hypothesis with high number of features

* Using linear / logistic regression on such features with the “create new features by multiplying features together technique” results in an exponential number of features as n increase and is prone to over fitting.
* Including all the quadratic features of a 50x50 pixel image results in ~3 million features which is computationally very expensive

**Modeling a neural network as a graph**





*This figure includes the bias nodes (always 1) x0 and a20*

* Each neuron (orange) will take a set of inputs (x) and perform logistic regression on x to return a hypothesis
* Multiple neurons can be connected together in layers to create a more complex final hypothesis
* Notation
  + j is 1 indexed, i is 0 indexed as and are always 1
  + refers to the activation (return value) of the ith node/unit in layer j
    - Note that
  + refers to the matrix of parameter/weights in the jth layer where is the weight of the edge to node[j+1][a] from node[j][b]
  + refers to the number of layers in the network
  + refers to the number of nodes in layer j, excluding the bias node
  + refers to the output of the network
* Computation
* Visualization of

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | For each left node | | | |
| For each right node |  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + If layer j has nodes and layer j+1 has nodes (note including bias node), then is a x ( +1) matrix
    - This is because as and are always 1
  + Note that is a 3 dimensional array
* Vectorised computation
  + for all j starting from 0
  + ;
  + ;

**Examples of simple neural networks for logical functions**

AND

|  |  |  |
| --- | --- | --- |
| x1 | x2 |  |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



NOT

|  |  |
| --- | --- |
| x1 |  |
| 1 | 0 |
| 0 | 1 |



|  |  |  |
| --- | --- | --- |
| x1 | x2 |  |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

OR



|  |  |  |
| --- | --- | --- |
| x1 | x2 |  |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

NAND



XNOR

|  |  |  |
| --- | --- | --- |
| x1 | x2 |  |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |



**Multi-class Classification**



Output is now an k dimensional vector where k is the number of classes.

* e.g. to denote the input as being of class 3
* The y values of the training set are also k dimensional vectors

|  |  |  |
| --- | --- | --- |
| Type of classification | Binary class classification | Multi class classification (k classes, ) |
| Nodes in last layer |  |  |
| Output and training labels |  | (y is a k dimensional vector) |

**Cost Function**

Notation:

* where k is the number of classes (can be 1 if binary classification)
* is the ith output of the neural network

Similar sum to logistic regression but over all the output nodes

Regularisation for all weights excluding bias (j=0) weights

**Back Propagation (calculating )**

* Algorithm to compute derivative terms
* minimize cost function and find optimal value of
* represents the “error” of node j in layer l
* the derivative of the cost function of training example t

For the last layer (j = l)

* where is a k dimensional vector and y is the output vector of the training data point

For all previous layers with (1 < j < l) excluding input layer

* *.\* is the elementwise multiplication operation; 1 is the*

**Backward propagation algorithm for multiple training examples**

Set for all l, i, j (1 index)

For i=1 to m

Set

Perform forward propagation to compute for l = 2,3,4…L

Using , compute

Compute

for all values of *l*

if

if

*Notes:*

* *functions as an accumulator to add up values of (for each l) for each training example*
* *results in a matrix, contains the “error” values of each weight from layer l to layer l+1*
* *is the total (for all training examples) value of*
* *is thus the average value of over all training examples*

**Under-fitting and Over-fitting**

**Under-fitting**

* The model does not fit the training set well
* E.g. Using a linear regression for a quadratic function

**Over-fitting**

* Where the model fits the training set extremely well, to the point that it starts to fail to make general observations and thus fails to generalize to new examples
* E.g. Using a very high order polynomial to fit a quadratic curve, the high variance of the model (many values for theta) mean that it could use a complex set of theta that works extremely well on the training set but doesn’t work well on other data
* Could be due to having too many features (leading to high variance)



Dealing with over-fitting

* Reduce the number of features
  + Manually removing redundant / unimportant features
  + Using a model selecting algorithm (automatically decides which features to keep)
* Regularization
  + Keeps all the features but reduces the magnitude of 🡪 results in a simpler hypothesis
  + Modify cost function
    - For linear regression
    - For logistic regression
    - is the regularization parameter to control the extent of regularization
    - If is too large, under-fitting can occur
  + Gradient descent algorithms for regularized regression
    - * For j in range [1, n]
      * Don’t regularize
    - Where is different for linear compared to logistic regression
  + Normal equation for regularized linear regression
    - Where L is the (n+1)(n+1) identity matrix with L[0][0] = 0

**Advice on dealing with machine learning**

**Debugging a learning algorithm**

E.g. Debugging linear regression (test accuracy very low)

- Get more training examples 🡪 Fixes high variance

- Try smaller sets of features 🡪 Fixes high variance

- Try additional features 🡪 fixes high bias

- Try adding polynomial features 🡪 fixes high bias

- Try decreasing 🡪 fixes high bias

- Try increasing 🡪 fixes high variance

**Evaluating Hypothesis Function**

Used to check if the hypothesis is overfitting or under fitting

* Split the dataset into 70% training data and 30% test data (randomly sorted)
* Train on the training set
* Evaluate test-set error:
  + Cost Function ; find the value of the cost function on the test set
  + Misclassification error:

**Model Selection**

* Split the data into
  + Training Set (60%)
  + Cross Validation Set (20%)
  + Test Set (20%)
* Train various models and choose the one with the lowest cross validation set error
  + Evaluate the selected model by analyzing its

**Bias (under-fitting) vs variance (over-fitting) in model selection**



* If both and are high, the model might be under fitting
* If is low but is high, the model might be over fitting

**Choosing the regularization parameter**

* Train various models with different values of
* Select the model with the lowest value of
* Evaluated the selected model using



**Learning curves**



* Note that is calculated with the partial training set while is calculated with the full cross validation set
* If a learning algorithm is suffering from high bias, getting more training data will not help much
* If a learning algorithm is suffering from high variance, getting more training data is likely to help

**Choosing Network Architecture**

* Smaller neural networks
  + More prone to under fitting
  + Computationally cheaper
* Larger neural networks
  + More prone to overfitting but can use regularization to deal with it (more effective than using a smaller neural network)
  + Computationally more expensive

**Recommended Steps:**

* Build the system fast and dirty
* Plot learning curves
* Manual error analysis
  + Classify misclassified training / cross validation data
  + Numerical error evaluation to check if optimizations work

**Skewed Classes**

* The proportion of one class compared to another in the training set is very high
* Precision/ Recall error metric for skewed classes

|  |  |  |
| --- | --- | --- |
| Predicted class / actual class | 1 | 0 |
| 1 | True Positive | False Positive |
| 0 | False Negative | True Negative |

Precision: Of all the examples predicted to be positive, what fraction is actually positive

Precision: Of all the examples actually positive, what fraction did we correctly dectect

**Support Vector Machine (SVM)**

Hypothesis:



Cost Function:

Where C is a parameter (replaces ) to be adjusted.

* Large C: low bias, high variance
* Small C: high bias, low variance

The svm is a *large margin classifier*, meaning that it chooses a decision boundary equidistant from both sides of classes.



**SVM with Kernels**

**Generates nonlinear decision boundaries**

Modified hypothesis

Given x, choose some landmarks and compute new features :

1. for each training example

Note that in this case, m=n

*Note: This is using a Gaussian kernel*

* Increasing makes the Gaussian curve more gentle, and vice versa
  + Larger : higher bias, lower variance
  + Smaller : lower bias, higher variance



If and ; points here l1 and l2 would be classified as positive.

Types of kernels

1. Gaussian kernel
2. Linear kernel (no kernel)
3. Polynomial kernel:
4. String Kernel
5. Chi-squared kernel
6. Histogram kernel

When to use logistic regression or SVM

* If n is large
  + Logistic regression, or SVM without a kernel
* If n is small, m is intermediate
  + Use SVM, Gaussian kernel
* Is n is small, m is large
  + Create/add more features, then use linear kernel or logistic regression

**Unsupervised Learning: Clustering**

This is the problem of grouping the dataset into separate clusters where data points in each cluster are similar