Machine Learning – Course summary notes and guidelines

Karthik A

April 3, 2018

Part I
Algorithms

1 Supervised Learning

1.1 Regression

Cost function:

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

Gradient descent: Simultaneous update: {
$\theta_j := \theta_j - \frac{\alpha}{m}\frac{\partial}{\partial\theta_j}J(\theta) - \alpha\frac{\lambda}{m}\theta_j$
}
Simultaneous update: {
$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y)x^{(i)}$
}

For small α, $J(\theta)$ should reduce with every iteration If polynomial regression, ensure feature scaling and mean normalization

Replace each value with $(x - \mu)/s$ where $s = (x_{max} - x_{min})$ or s=Std Deviation

1.2 Logistic Regression

Logistic and cost function

$$h_\theta(x) = \frac{1}{(1 + e^{-}\theta^T X)}$$

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

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

1.3 Support Vector Machines (SVM)

- if n is small (10k) and m is intermedeate (50k) use SVM. If n is small and m is large – use logistic regression. Neural network is good for all but difficult to train
- In SVM $C = 1/\lambda$
- Without kernel or linear regression SVM is logistic regression

1.4 Neural Network

Back propogation - neural network terminology for minimizing our cost funtion $\delta = a - y$, where a is the activation value
Layer L = Output layer
Layer 1 = Input layer
s_j = units in layer j
a_i^j = activation of unit i in layer j
$\Theta^{(j)}$ =matrix of weights controlling function mapping from layer j to layer j+1
$a^{j+1} = g(\Theta^T X)$
If s_j units in layer j, s_{j+1} units in layer $(j+1)$, then dimensions of Θ^j will be $s_{j+1} \times (s_j + 1)$

2 Unsupervised Learning

2.1 K Means and Dimensionality reduction

How to choose K in Kmeans

1. $J(\theta)$ vs K – see if it elbows out
2. No good elbows, use the downstream purpose. For example, shirt sizes

Dimensionality reduction

- Data compression
- Visualization

$$[USV] = svd(\Sigma)\ where, \Sigma = (1/m)\sum(X * X^T)$$

Determining number of dimensions k explaining variation of a target say 95%

$$Explained\ Variation = 1 - \sum_{i=1}^{k} s_{ii} / \sum_{i=1}^{n} s_{ii}$$

2.2 Anomaly detection

Common applications : Fraud detection, Manufacturing, monitoring machines in a large cluster

Differences between Anomaly detection and clustering

1. y=1 is very small , typically 0–20
2. y=0 is a large number
3. Different types of anamolies exist which are not known upfront

Some tips for anomaly detection

- Add a few failed samples to CV and test, do a F_1 score and evaluate the ϵ values
- When features are not gaussian distributed, use the $x^{(1/10)}$ or $x^{(1/2)}$ or $log(x)$ etc and make gaussian
- When encountering common problem that p(x) is similar to regular in muti dimensions try,

  –Introducing features such as
  $CPULoad/networktraffic$ or
  $(CPUload)^2/networktraffic$

  –Use multivariate gaussian - automatically captures the correlation between features but is computationally expensive (also m >n else Σ is not inversible)

3 Recommend-er systems and COllaborative FIltering

n_u = number of users (columns)(j) n_m = number of movies (i) $r(i,j) = 1$ if user j has rated movie i

X = hypothetical or real vector such as Genre - action, comedy each being a column x^1 or x^2 or x^n m^j = number of movies rated by user j

In COFI , X is also not known

- min $\theta = \sum_i \sum_n$ and get the double gradient over X and θ
- Initialize with small random values
- Feature scaling is not required as the rating scale is always same across uses but mean normalization helps faster convergence

Part II
Practical suggestions for machine learning

4 Model selection and refinement

4.1 Over–fitting problem

- Reduce the number of features
- Use the model selection algorithm described later
- Use regularization to reduce the magnitude of theta (λ high)

4.2 Model selection

60%–20%–20% is a broad split for the train, cross validation and test sets

1. Get the polynomial degree – Find $J(\theta)$ for each increasing polynomial feature. Test on CV to fix the polynomial features. Check on test
2. If underfit reduce λ, if overfit , increase λ - check the error on the CV set to fix the lambda value

3. To determine whether more or less data, review the learning curves i.e plot the J_{cv} and J_{train} against increasing number of examples

   —- if J_{cv} and J_{train} are converging with less samples and the value is high, then it is High Bias problem and adding more data to the problem will not help

   —- if J_{cv} and J_{train} are different, then adding more samples will help them converge - high variance problem

4. Error Analysis: General guideline, start with a simple algorithm and then increase the number of features. Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made. We can find some feaures that may be useful

4.3 Types of actions possible

1. Collect more data – High variance
2. Try smaller set of features – High Variance
3. Try additional features – High Bias
4. Try polynomial features – High Bias
5. Increasing λ – High Variance
6. Decreasing λ – High Bias

5 Pipeline Analysis

Example of pipeline for machine learning is given below: Image $\rightarrow$ text detection $\rightarrow$ Character segmentation $\rightarrow$ Character recognition

- Sliding windows - step size or stride
- 1D sliding window for char segmentation
- Get lots of data and artificial data

6 Rules for getting more data

- Make sure you have a low bias classifier before starting data collection eg. add more features or hidden units in a neural network until bias is low
- Estimate data collection effort
- Ceiling analysis - which part of the pipeline must I improve - Look at the accuracy in the test set by feeding the y_{actual} to the next part of the pipeline and see how it improves - delta accuracy should be used to determine where to put the effort

7 Learning with large data sets

- *It is not who has the best algorithm that wins but who has the most data. The problem with large data sets is that gradient descent is computationally expensive for each iteration*
- Plot the learning curve and if high bias and J is ok, no need to do on the full data set (m) as it will converge before then – adding more samples has no benefit
- If high variance use map reduce – i.e split the summation into separate machines

7.1 Stochastic gradient

- Stochastic gradient - Do gradient descent with m=1 after shuffling the data set
- Mini batch - do gradient descent in batches b <m
- Batch gradient descent - full m

Plot J over number of iterations and average over the batch sizes

1. Gets smoother with smaller α
2. Gets smoother with larger b
3. Clarifies with larger b if there is a lot of zig-zag in the chart
4. If J increases with b, use smaller α

5. interesting but not often used is reducing α with iterations = $\alpha = const/(const + iterations)$

Online learning is extreme example where the data is taken evaluated and dropped immediately

$P(y = 1|r; \theta)$ - predicted CTR (Click Through Rate)