0.1 Coursera

Week 1

Machine Learning algorithms : Supervised learning Unsupervised learning Reinforcement Learning Recommender systems

Supervised Learning based on right answers Regression problem: predict continuous valued output (price) Classification problem: output value of 0 or 1 To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X -¿ Y so that h(x) is a good predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:

Unsupervised Learning Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

We can derive this structure by clustering the data based on relationships among the variables in the data.

With unsupervised learning there is no feedback based on the prediction results. Not told what to do -¿ find some structure in data Clustering Algorithm not give the algorithm the right answers Cocktail party problem: take a data that seems dated together and separate them (two audio sources),

Hypothesis Function

Cost Function Cost Function measures the accuracy of our hypothesis function.

Gradient descent Algorithm Optimize parameters in hypothesis function to minimize the cost function Convex functions has only a global optima, no local optima (good for working with) "Batch" gradient descent: each step of gradient descent uses all the training examples

Learning rate Controls how big the step is in the optimization process

Week 2

Fix a notation thorughout the work m -¿ the number of training examples i n -¿ the number of features j
$x_j^{(i)}$
$i = 1, ..., m$
$j = 0, ..., n$ i are the lines and j the columns. Gradient descent for multiple variables

Feature scale mean normalize data to converge faster (do not apply to inserted x_0 that equals 1 for example). Adjust input values as shown in this formula.

$$x_i := \frac{x_i - \mu_i}{s_i} \tag{1}$$

where μ_i is the average of all values for feature (i) and s_i is the range of values $(max(x_i) - min(x_i))$ or the standard deviation of a given feature i. get all features between -1 and 1 range or 0 and 1

Gradient descent $\mathcal{O}(n^2)$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2}$$

works well when n is large

Vectorized implementation of Gradient descent $\mathcal{O}(n^2)$

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - Y) \tag{3}$$

Learning rate how to choose: look graph min J vs No. of iterations. It should decrease at every iteration.

Polynomial regression Create new features to fit a polynomial regression to a linear regression. E.g.

$$x_0 = (feature) \tag{4}$$

$$x_1 = (feature)^2 \tag{5}$$

Analytic solution (faster convergence) The value of $\theta = (X^T X)^{-1} X^T y$ (normal equation) gives the optimal value of θ that minimizes $J(\theta)$ and doesn't need feature scaling. Don't need to iterate nor choose α. Compute the inverse costs a lot $\mathcal{O}(n^3)$. So it is preferably to use if n is small over the iterative method. Noninvertibility: pinv -¿ pseudo inverse: calculates the value of the inverse even if it is non inversible.

Week 3

Classification problem: take a bunch of data and outputs discrete values multi class: several discrete outcomes binary class: 2 outputs (0,1)

Logistic regression (classification algorithm): output is always between 0 and 1. sigmoid function or logistic function: $g(z) = \frac{1}{1+e^{-z}}$. Hypothesis: $h(x) = g(\theta^T x)$ maps any real number to the (0,l) interval. $h(x)$ will give the probability that our output is 1.

Decision boundary: line where the hypothesis equals to 0.5. Separates 2 regions, greater than and less than 0.5 to the hypothesis function. The decision boundaries can be linear or not. The decision boundary is a property of the hypothesis function. The greater the order of the polynomial, the more complex is the shape.

Cost function: find a function that is convex to arrive in a global minimum.

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

No need to manually pick α in the 3 last algorithms. They are fasten than gradient descent. However, they are more complex

Multiclass classification One vs all (one vs rest)

Avoid overfitting: doesn't add too much features, otherwise the training data will not predict well values. The cost function is almost zero to the training example. To fight this problem: reduce the number of features (select which features to keep), regularization (keep all features but reduce the magnitude of parameters theta), Regularization works well when we have a lot of slightly useful features. Add a regularization term in the end of the cost function. A regularization parameter is set to make a trade off between how well the data is fitted and weakening the parameters theta.

In the end we have a regularized linear regression that works much better than the regular one. It even avoid that the normal equation is non-reversible. The calculus is the same, but we add an extra term to the cost function. To calculate the gradient descent, one must actualize the value of theta 0 and of others thetas, since the added sum takes from indexes 1 to n. The value of theta 0 is known as bias.

Regularized cost function:

$$J(\theta) = \frac{1}{2m} \sum_{n=1}^{m} (h(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{n=1}^{n} \theta_j^2 \tag{6}$$

Cost Function in logistic regression

...

Week 4 - Neural Networks

Neural network: more efficient way to learn complex hypothesis.

Model 1 representation:

- Dendrite: input wires
- Axon: output wire
- Cell body: do computations
- axon send info to another neuron

Hypothesis function = activation function

θ = parameters = weights

Layers: first layer: input layer, last layer: output layer, hidden layer: values you don't observe between the input and output.

Notation: Θ^j - matrix of weights controlling function mapping from layer j to layer $j+1$. The dimension is equal to $s_{j+1} \times (s_j + 1)$. The index $+1$ takes into account the bias added to the inputs and to the computed layer.; $a_i^{(j)}$ - activation of unit i in layer j;

Model Representation II Vectorized: forward propagation:

$$z^{(j)} = \Theta^{(j-1)} a^{j-1} \tag{7}$$

$$a^{(j)} = g(z^{(j)}) \tag{8}$$

Last layer:

$$z^{(j+1)} = \Theta^{(j)} a^{j} \tag{9}$$

$$a^{(j+1)} = g(z^{(j+1)}) = h_\Theta(x) \tag{10}$$

The away a NN is connected is called an architecture. The first layers computes task of a certain complexity that increases when we go deeper into the network. The last layer usually computes the most demanding task.

Multiclass Classification To classify data into multiple classes, we let our hypothesis function return a vector of values.

Task 3: Multi-class Classification and Neural Networks (matlab)

The training examples of a training set must be independent!

Week 5

Backpropagation. "Backpropagation" is neural-network terminology for minimizing our cost function

Generalized linear regression for neural networks

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) log(1-h_\Theta(x^{(i)}))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ij}^{(j)})^2 \quad (11)$$

where L = total number of layers in the network, sl = number of units (not counting bias unit) in layer l, K = number of output units/classes.

We denote $h_\Theta(x)_k$ as being a hypothesis that results in the k^{th} output.

We have added a few nested summations to account for our multiple output nodes.

The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit).

- the double sum simply adds up the logistic regression costs calculated for each cell in the output layer
- the triple sum simply adds up the squares of all the individual Θs in the entire network.
- the i in the triple sum does not refer to training example i

The back-propagation algorithm is used to minimize J. So we need to compute J and the partial derivatives of J. We compute the delta error for each layer (but the input layer because we set $a^{(1)} = x^{(i)}$), beginning in the last layer, where: $\delta_j^L = a_L^l - y_j$, $\delta_j^{l-1} = (\Theta^{l-1})^T\delta^{(l+1)}. * g'(z^{(l-1)})$, $g'(z^{(l-1)}) = a^{(l-1)}. * (1-a^{(l-1)})$. Step by step

1. Training set $\left\{(x^1, y^{(1)}), ..., (x^m, y^{(m)})\right\}$.
2. $\Delta_{ij}^l = 0$ for all l, i, j. (Useful to compute $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$)
3. $\delta_j^{l-1} = (\Theta^{l-1})^T\delta^{(l)}. * g'(z^{(l-1)})$

4. $g'(z^{(l-1)}) = a^{(l-1)}. * (1 - a^{(l-1)})$

1. For $i = 1$ to m

```
        Set $a^{(1)} = x^{(i)}$
        Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, ..., L$
        Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
        Compute $\delta^{(L-1)}, ..., \delta^{(2)}$ where $\delta_j^{l-1} = (\Theta^{(l-1)})^T \delta^{(l)}. * g'(z^{(l-1)})$.
          Compute $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$. Contribution of each
    training set to the gradient that is calculated with the general Theta
    for the NN.
```

2. Compute $D^{(l)} := \frac{1}{m}(\Delta^{(l)} + \lambda\Theta^{(l)})$ if $j \neq 0$.

3. Compute $D^{(l)} := \frac{1}{m}\Delta^{(l)}$ if $j = 0$ (bias term).

4. $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

So in general you do a forward propagation with a training example, than a back propagation with the same training example, than changes the training example and repeats.

Implement gradient checking for make sure that forward and backpropagation are working correctly. After you know they are working, comment the code.

The neural network used to check back-propagation is usually small, that's why it is important to code general functions.

Week 6

Improving the Learning Algorithm:

- Get more training examples
- Try smaller setof features
- Try getting additional features
- Try adding polynomial features
- Try decreasing λ
- Try increasing λ

Diagnostic: show what is working well and what is not in the algorithm. Show what is more fruitful to try to improve next.

Evaluating a hypothesis: split the dataset in trining set (80%), cross validation set (20%) and test set (20%). Each set must be randomly distributed. You can try several hypothesis models and generate several θ and test on the cross validation set to find the best degree d of the polynomial used to estimate θ. Then, estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$.

Missclassification error:

Classification Test error $= 1/m_{test} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}, y_{test}^{(i)})$

Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv})^2$

Usually a learning algorithm may have a variance problem or a bias problem.

If Jtest is high, you are facing a bias error (underfit) Jcv approx Jtest. If Jcv is high, you are facing a variance error (overfitting) Jcv >> Jtrain.

We can observe this phenomena with the help of the graph Cost ($J(\theta)$ vs d (polynomial degree))

Choose a good value for λ to solve these problems.

1. Try several values.
2. minimize $J(\theta)$
3. calculates Jcv
4. Take the smalles Jcv
5. Evaluate Jtest of the chosen set
6. plot the errors in function of λ

Overfittin the data = low λ

1. Create a list of lambdas (i.e. $\lambda \in 0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10.24$
2. Create a set of models with different degrees or any other variants.
3. Iterate through the λ and for each λ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the $J_{cv}(\Theta)$ without regularization or $\lambda = 0$.
5. Select the best combo that produces the lowest error on the cross validation set.

6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

Learning curves

Error vs training size. We can find out if the algorithm suffers from variance or bias. If the learning algorithm suffers from variance, adding training examples may help. Adding more layers doesnt help with high variance error.

Resume:

1. Getting more training examples: Fixes high variance
2. Trying smaller sets of features: Fixes high variance
3. Adding features: Fixes high bias
4. Adding polynomial features: Fixes high bias
5. Decreasing λ: Fixes high bias
6. Increasing λ: Fixes high variance.

A neural network with fewer parameters is prone to underfitting. It is also computationally cheaper.

A large neural network with more parameters is prone to overfitting. It is also computationally expensive. In this case you can use regularization (increase λ) to address the overfitting.

Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.

Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.

In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Skewed classes: events that are rare in a given set. Example: 0.5 % of people have cancer.

The metrics to measure skewed classes are called precision and recall.

Precision is defined as:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{12}$$

Recall is defined as:

$$precision = \frac{true\ positives}{true\ positives + false\ negatives} \tag{13}$$

We can change the threshold value for h for classifying y = 1 or y = 0. if our threshold value is ¿ 0.5, we are valuing confidence, so high **precision**, lower recall.

However, the opposite means the false negatives are important and should NOT happen. So, when in doubt, y = 1. You end up with higher recall and lower precision.

You want precision and recall to be both elevated.

How to choose the threshold? You can compute the F1 score $2\frac{PR}{P+R}$. There are others too.

Week 7

Large margin calssifiers

We can rewrite the cost function to optimize the calculations, choose functions that are very close to the optimal but are easier to calculate.

For example, the logistic regression cost with support vector machine (SVM) is known as: SVM hypothesis and is written as follows for the logistic regression problem:

$$min_\theta C\sum_{i=1}^{m}[y^{(i)}cost_1(\theta^Tx^{(i)})+(1-y^{(i)})cost_0(\theta^Tx^{(i)})]+\frac{1}{2}\sum_{j=1}^{n}\theta_j^2 \tag{14}$$

Cost1 and Cost0 are the linearization of the cost curve.

The parameter C plays with bias and variance. Large C means lower bias and high variance. While small C means higher bias and low variance.

So in the end you want to predict the value of y juist basing over the value of θ^Tx.

Kernels:

Suppose you have a non-linear decision boundary. Given x, compute new features based on the distance with given landmarks. The functions that calculate is called a similarity function (kernel function) and may be, for example, a gaussian kernel, where:

$$f_1 = exp(-\frac{||x-l^{(k)}||^2}{2\sigma^2}) \tag{15}$$

Landmark: $l^{(k)}$. Kernel Function: andmark: $f^{(k)}(x)$.

With them, we can define new features and make complex boundary curves. Instead of basing ourselves in the product of θ^Tx, we compute $\theta^Tf(x)$ to make our decisions.

The variance of the kernel plays over bias and variance.

Large σ^2: features f_i vary more smoothly, high bias, lower variance.

Small σ^2: features f_i vary less smoothly, lower bias, higher variance.

If SVM is overfittinf your data, it is good to DECREASE C and INCREASE σ^2.