**Coursera Machine Learning Class**

**Taught by Professor Andrew Ng, Stanford**

**Week 1**

**1 - Introduction**

*What is Machine Learning?*

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, improves with experience E.”

Example: playing checkers.

* E = the experience of playing many games of checkers
* T = the task of playing checkers
* P = the probability that the program will win the next game

*Supervised Learning*

* In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
* Supervised learning problems are categorized into “regression” and “classification” problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

*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.

**2 - Linear Regression with One Variable**

*Model Representation*

* To describe the supervised learning problem, 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.
* When the target variable that we’re trying to predict is continuous, we call the learning problem a regression problem. When y can take on only a small number of discrete values, we call it a classification problem.

*Cost Function*

* The cost function takes an average difference of all the results of the hypothesis with inputs from x’s and the actual output y’s.
* This function is otherwise called the “Squared error function”, or “Mean squared error.” The mean is halved as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the ½ term.

*Gradient Descent*

* So we have our hypothesis function and we have a way of measuring how well it fits into the data. Gradient descent can be used to estimate the parameters in the hypothesis function.
* We take the derivative of cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter a, which is called the learning rate.
* Adjust our parameter a to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.
* Gradient descent can converge to a local minimum, even with the learning rate a fixed. As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease over time.

**Week 2**

**1 – Multivariate Linear Regression**

* Linear regression with multiple variables is also known as “multivariate linear regression”.
* We can speed up gradient descent by having each of our input values in roughly the same range. This is because alpha will descent quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.
* Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input variables by the range of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just 0.
* If alpha is too small: slow convergence, if alpha is too large: may not decrease on every iteration and thus may not converge.

**2 – Computing Parameters Analytically**

|  |  |
| --- | --- |
| **Gradient Descent** | **Normal Equation** |
| Need to choose alpha | No need to choose alpha |
| Needs many iterations | No need to iterate |
| O(kn^2) | O(n^3), need to calculate inverse of XTX |
| Works well when n is large | Slow if n is very large |

If XTX is noninvertible, the common cause might be having:

* Redundant features, where 2 features are very closely related
* Too many features. In this case, delete some features or use “regularization”

**Week 3**

**1 – Logistic Regression**

*Classification*

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn’t work well because classification is not usually a linear function.

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we will focus on the binary classification problem in which y can take on only 2 values, 0 and 1.

*Decision Boundary*

The decision boundary is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

*Advanced Optimization*

“Conjugate gradient”, “BFGS”, and “L-BFGS” are more sophisticated, faster ways to optimize theta that can be used instead of gradient descent.

*Multiclass Classification: One-vs-all*

We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction

Train a logistic regression classifier h(theta)(x) for each class to predict the probability that y = i. To make a prediction on a new x, pick the class that maximizes h(theta)(x).

**2 – Regularization**