MACHINE LEARNING

Directed Research

ABSTRACT

Machine Learning is the science of getting computers to act without being explicitly programmed. The many advances in Machine Learning include autonomous vehicles, effective searches and speech recognition. This paper will discuss the topics (i): Supervised Learning (Linear Regression, Logistic Regression, support vector machines, neural networks) (ii): Unsupervised Learning (K Mean and Principle Component Analysis) (iii): Best practices in Machine Learning (bias/variance theory; innovation process in machine learning)

Alton J Render

CS 6953

TABLE OF CONTENTS

Alton J Render

Andrew Ng

Coursera

11-28-2016

Machine Learning: Directed Research

Supervised Learning, as it pertains to Machine Learning (ML), is the act of presenting data sets or training data to the agent and allowing it to infer, based upon the input, an expected output. Key components to be discussed are Linear Regression, Logistic Regression, Neural Networks, and Support Vector Machines.

Linear Regression is the act of predicting a real-valued output based on input value.

$$h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i,$$

Linear Regression Model

Let's suppose we are given a set of data to determine the best cities to open a new franchise. This data contains profits and populations in cities. Figure 1 below shows a plot of the data.

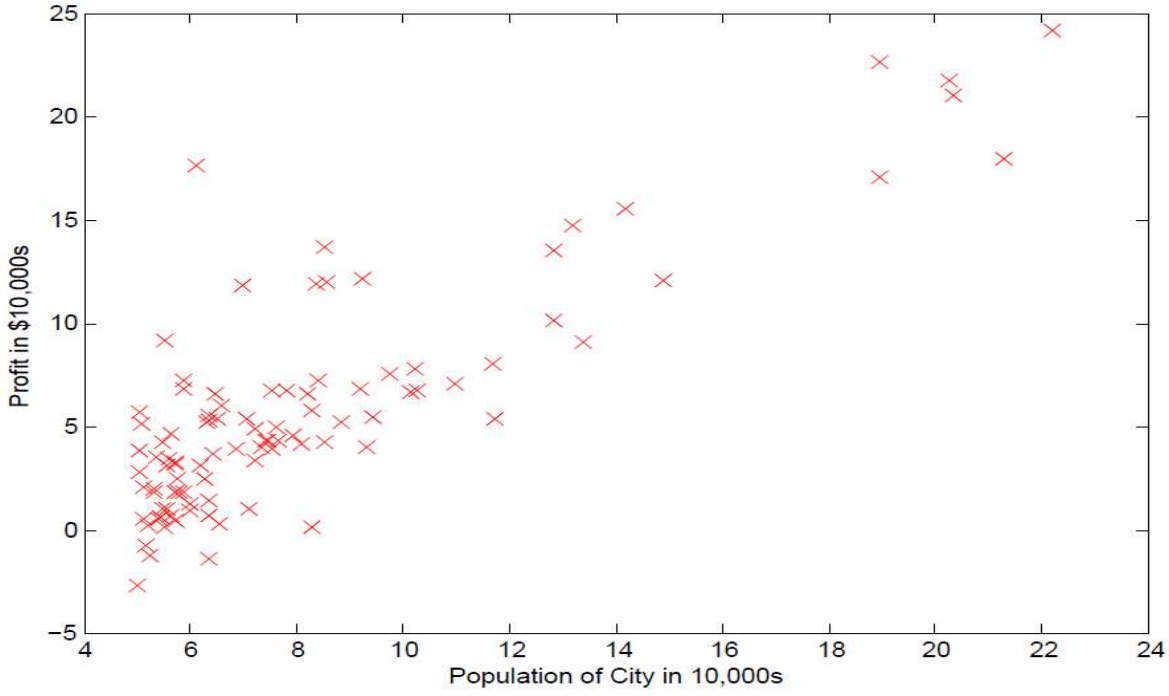


Figure 1: Sample training data

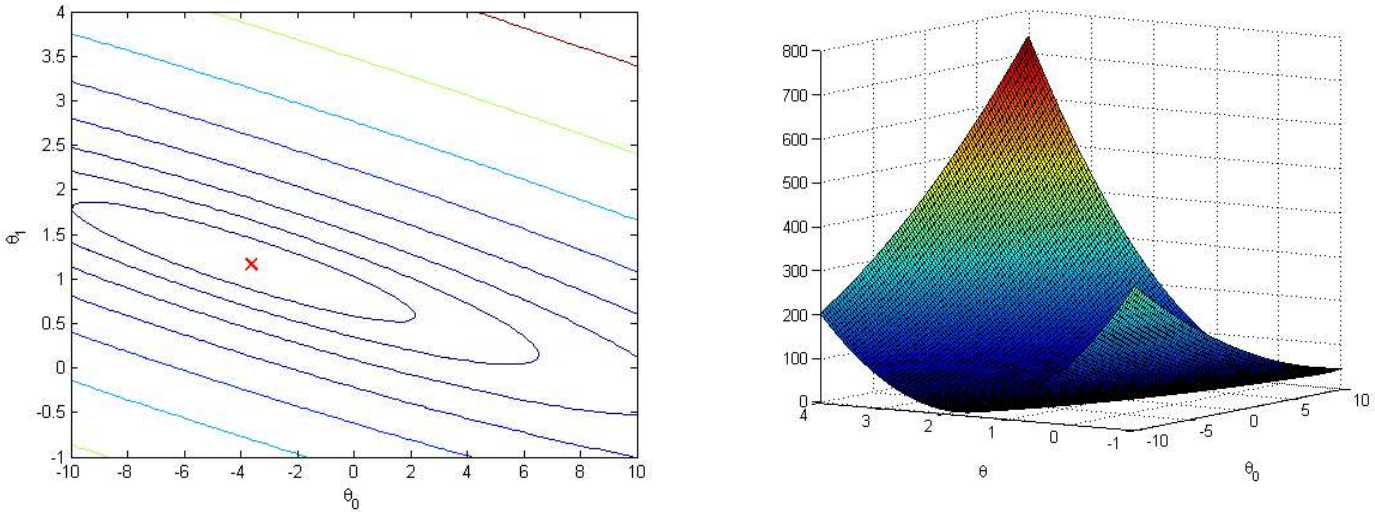
If we want to model a line on the set of these points linear regression is the key. The standard approach to solving this type of problem is to define an error function (also called a cost function) that measures how "good" a given line is. Our objective is to minimize the cost

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

function $J(\theta)$ where the hypothesis $h_\theta(\mathrm{x})$ is given by the linear model below. By adjusting θ_j

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

we can minimize the cost $J(\theta)$. One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update. With each step of gradient descent, the parameters θ_j come closer to the optimal values that will help us achieve the lowest cost $J(\theta)$ to fit or model [Coursera 1]. Implementing linear regression with gradient descent in Octave gives us a good example of how the algorithms work. We can use a learning rate of 0.01 and have 1500 iterations to test convergence and performance of the algorithm. Running Gradient Descent on the collection of data yields the cost $J(\theta) = 32.073$. Upon further inspection, we see that θ_0 and θ_1 are -3.630 and 1.166 respectively. This can be seen in figure 2 on the next page. The purpose of these graphs are to show that $J(\theta)$ varies with the changes in θ_0 and θ_1. The minimum is the optimal point for θ_0 and θ_1, and each step of gradient descent moves us closer to this point [Coursera 1].



(a) Contour, showing minimum

(b) Surface

Figure 2: Cost function $J(\theta)$

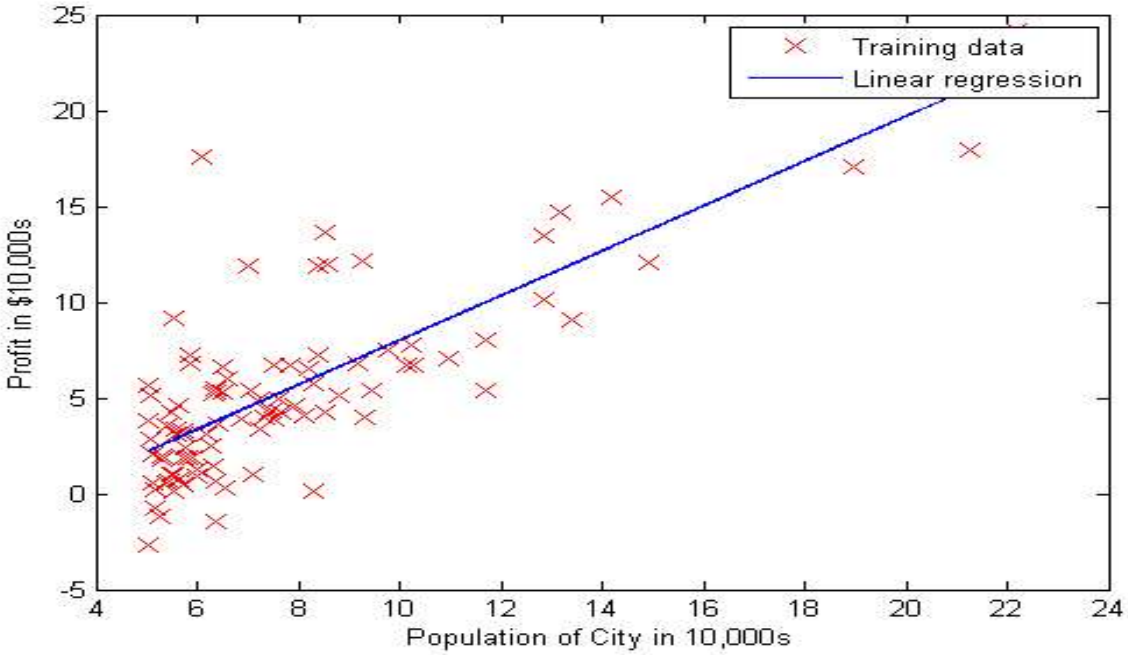


Figure 3: Training data with linear regression fit

Now that we've gained theta and the cost value, we can fit our model with a line to make predictions. Figure 3 shows the training data with linear regression fit. For our model, if we have

a population of 35,000, we can predict a profit of 4519.767. Similarly, if we have a population of 70, 000, we can predict a profit of 45342.450.

The figures and valued output show how linear regression can be used as a supervised learning method to help train an agent to make accurate predictions given simple sample data. The next section will focus on Logistic Regression.

Like all regression analyses, the logistic regression is also a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more metric (interval or ratio) independent variables [Stats 3]. In ML, common practices use logistic regression to handle classification problems. Consider, cases such as emails, transactions and tumors. Let's say we want to distinguish between the negative and positive classes; we want to separate spam from not spam emails; fraudulent form none fraudulent transactions; and malignant from benign tumors. Given a set of data which can be classified in terms of a binary 0 or 1, we, by using logistic regression, should be able to classify any new input after our system has learned. Let's take an example implemented via Octave. This example comes from the Coursera assignment 2 problem. We want to be able to accurately predict the probability of admission for a student based on the scores from two exams. Figure 5 on page 6 generated from Octave shows data displayed on a 2 – dimensional plot. The positive examples are indicated by the dark colored crosses while the negative examples by the yellow colored circles. The axis represents the two exam scores. Now that we have the data let's dive more into logistic regression. Consider we have our hypothesis $h_\theta(\mathrm{x})$, with a sigmoid function

$$h_\theta(x) = g(\theta^T x)$$

defined by $g(\mathrm{z})$ on page 6. The reason we choose a sigmoid function to model classification

problems solved with logistic regression is that we want to make sure the predicted value has a defined range. This will aid in differentiating between the positive and negative classes.

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

Sigmoid function

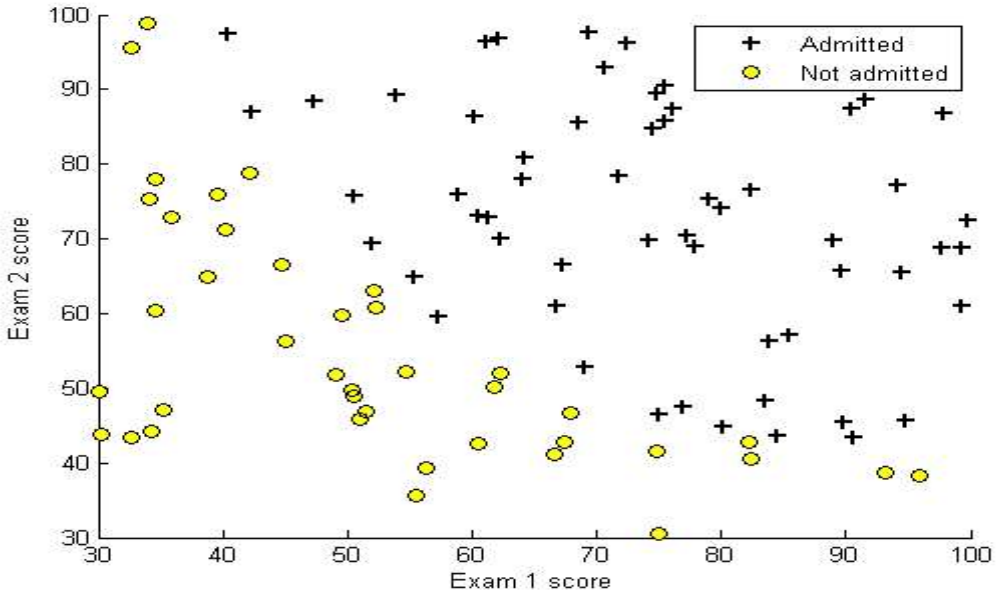


Figure 5: Scatter plot of training data

Similar to linear regression, logistic regression is also implemented with a cost function and gradient. The cost function in logistic regression is

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

and the gradient of the cost is a vector of the same length as θ where the j^{th} element

(for j = 0,1,…n) is defined as

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

While the gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regressions have different definitions of $h_\theta(x)$ [Coursera 1] Now let's solve the classification problem of the probability of admitting a student based on scores. The cost at initial θ (zeros) evaluates to 0.693 and the gradient here is -0.100, -12.001, and -11.263. Normally, after we have calculated the cost function and the gradient for it, we would take gradient decent steps. Instead, for this exercise, a built-in Octave function FMINUNC was used in order to help us find the best parameters θ for this logistic regression cost function, given our fixed data set consisting of (X and y values) [Coursera 1]. After the final pass of FMINUNC, the cost was found to be 0.2035 and the θ: -24.9329, 0.2044, and 01.99617. Based upon these findings, we can now separate the input space into two regions, one for each distinctive class (e.g. admitted, not admitted), by a linear boundary. The two dimensional representation of this boundary can be viewed below in figure 6 along with the training data

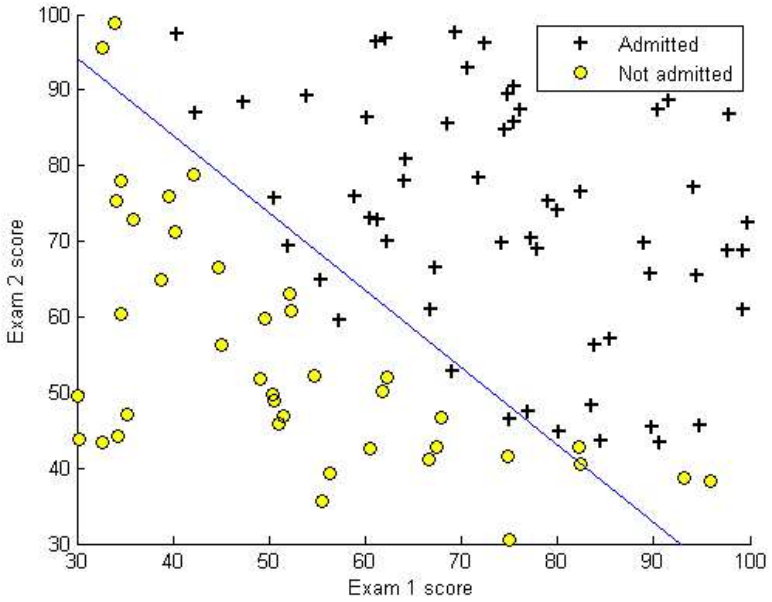


Figure 6: Linear boundary with logistical regression

With the model set and trained to the data, we are able to accurately predict a student admission probability. For a student with a score 45 on exam 1 and 85 on exam 2, we predict an admission probability of 0.7743. Based upon the model, we have measured the accuracy of prediction on this data to be 89 percent.

The key difference between linear regression and logistic regression is to realize that logistic regression does not try to predict the value of a numeric variable given a set of inputs. Instead, the output is a probability that the given input point belongs to a certain class [Kumar 4]. Knowing the predicted probability of an event is also the basis for neural networks. We will discover in the next section how this is beneficial to the learning algorithm.

Logistic regression, as mentioned in the previous section, is considered a linear classifier and isn't able to perform complex hypothesis (e.g. pattern recognition, signal processing, anomaly detection). However, the ability to make such complex hypothesis is the advantage of a neural network. In addition, a neural network (NN) is considered a "connectionist" computational system; information is processed collectively in parallel throughout a network of nodes. Similar to the neurons within the human biology, NN have what are known as perceptrons. A perceptron consists of one or more inputs, a processor, and a single output [Shiffman 5]. It follows the "feed-forward" model; the network has links that extend in only one direction proceeding from input nodes extending toward output nodes. Weight values are

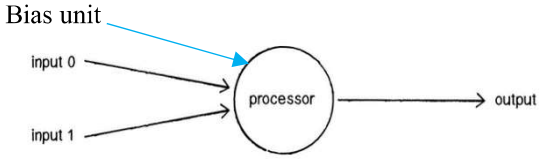


Figure 7: The perceptron

associated with each vector and node in the network, and these values constrain how input data

(e.g., satellite image values) are related to output data (e.g., land-cover classes). Weight values associated with individual nodes are also known as biases. Weight values are determined by the iterative flow of training data through the network (i.e., weight values are established during a training phase in which the network learns how to identify particular classes by their typical input data characteristics) [Leverington 6]. Once the network is trained, it can be applied toward

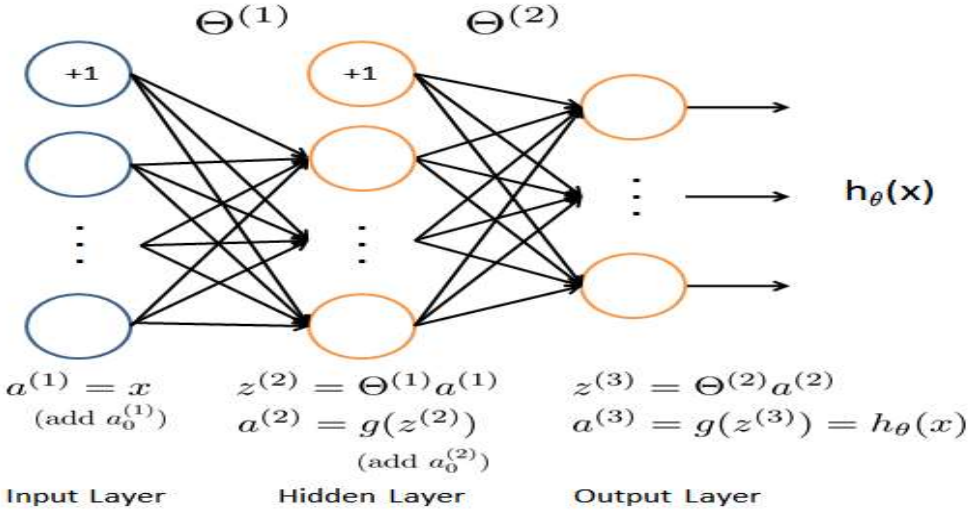


Figure 8: Neural network model

the classification of new data. Let's consider an example from the ML Coursera programming assignment. The neural network is shown in figure 8. It has 3 layers: an input layer, a hidden layer and an output layer. Here $g(z)$ is still denoted as the sigmoid function. The $a_i^{(j)}$ is known as the "activation" of unit *i* in layer *j*. The $\Theta^{(j)}$ denotes the matrix of weights controlling function mapping from layer *j* to layer *j + 1*. This model represents feed-forward or how forward propagation works to find the hypothesis. The input layer is set to x; the hidden layer is the vector of values calculated by the sigmoid function; and the output layer is logistical regression. However, unlike logistical regression, features are values calculated by the hidden layer. The data set used is a collection of 5000 training examples of handwritten digits. Each training

example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixels is “unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image [Coursera 1]. Implementing feedforward propagation and prediction in Octave yields a training set accuracy for this neural network of 97.520 percent. We can then apply what is known as backpropagation to learn the parameters for our network. The backpropagation algorithm will help find θ values which will assist in the minimization of $J(\theta)$ which can be denoted as $\min_{\theta} J(\theta)$. In order to implement gradient descent, we need to define a function that takes in input parameters θ and computes both $J(\theta)$ and the partial derivative terms $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$. It is safe to note that in our neural network, instead of having 1 logistical regression output units we could have several or *K* regression units. Having *K units* will yield a cost function:

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

The second term in the sequence is the regularization term similar to that of logistic regression. In addition, while computing the summations, we do not calculate the biases or sum over the terms when i *is equal to zero*. Coupled with the cost function is finding the partial derivate terms. The intuition of the backpropagation algorithm is that for each node, we will

compute the term $\partial_{ij}^{(l)}$ which represents the error of node j in layer l. In essence, it will capture the error of the activation $a_j^{(l)}$ of the node. In relation to our network in figure 8, we can demonstrate the backpropargation. In layer 3, the term $\partial_j^{(3)} = a_j^{(3)} - y_j$ can be used to describe the "error" of the layer. After we have found this delta we can then compute the delta terms for the earlier layers of our network in terms of $\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$. Where "**.***" is the element wise multiplication in Octave. Using the activation terms and the delta terms it is possible to compute the partial derivative term $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta^{(l+1)}$. Finally, we can compute what is known as the D term. It is exactly the partial derivative of the cost function with respect to each of the parameters and so, can be used for gradient descent or in one of the advanced authorization algorithms.

It is apparent from the above discussion that a neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and, therefore, generalize. In practice, however, neural networks cannot provide the solution working by themselves alone; a complex problem of interest is decomposed into a number of relatively simple tasks, and neural networks are assigned a subset of the tasks (e.g. pattern recognition, associative memory, control) that match their inherent capabilities [Hajek 7]. These capabilities make neural networks a powerful modeling tool for supervised learning.

The finally supervised learning topic I will discuss is termed support vector machines (SVM). In this section, I will discuss the application of using SVMs with Gaussian kernels on

data sets that are not linearly separable. To find non-linear decision boundaries with the SVM, we need to first implement a Gaussian kernel. The Gaussian/RBF kernel function is given as:

$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$

Theoretically, the kernel function is mathematical "sugar" which allows the SVM to perform a two – dimensional classification on a set of originally one-dimensional data. In general, the kernel function is able to project data from a lower dimension space to a higher dimensional space. Consider the data given below in figure 9, it is impossible to correctly

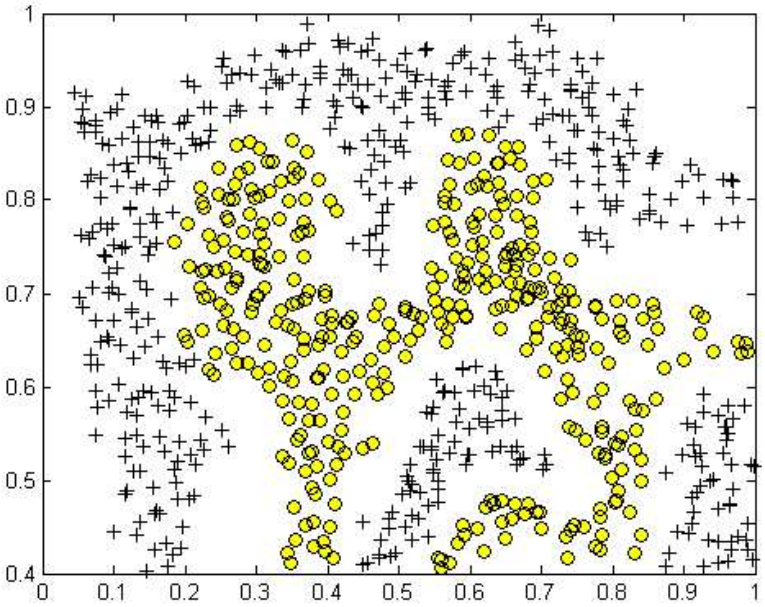
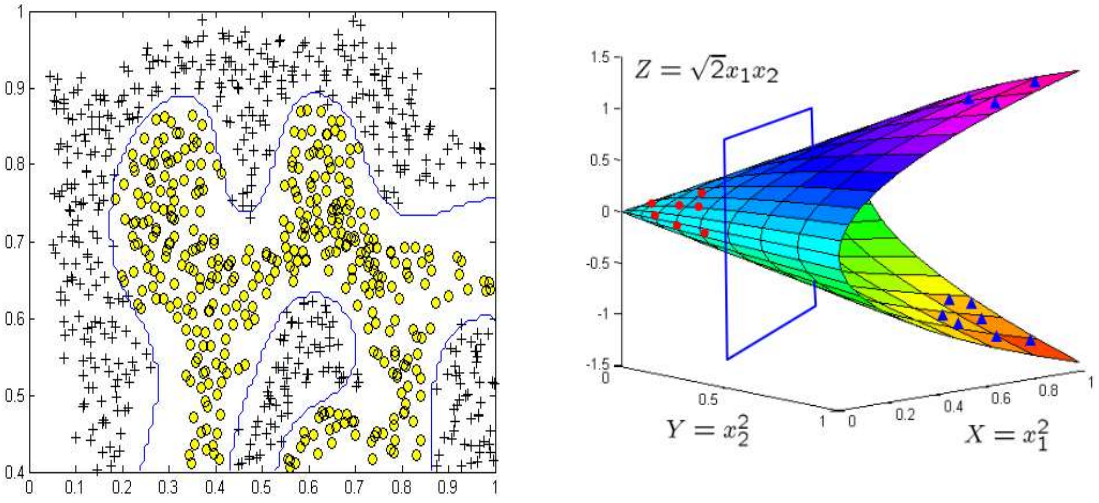


Figure 9: Example Dataset

separate the data using a linear boundary. If we, however, use SVMs with RBF, a decision boundary is created, similar to that of figure 10a, which is adequately able to separate most of the positive and negative examples along the contour of the dataset. Figure 10b shows the data being projected from the lower plane to a higher dimension. In this 3D image, it is easy to see that the

data is now linearly separable. This means that the problem is now solvable by the linear classifier. The Gaussian kernel was parameterized by the bandwidth parameter, $\sigma = 0.1$.



(a) 2-D Representation

(b) 3-D Representation

Figure 10: SVM (Gaussian kernel) Decision Boundary

The linear classifier used in this problem set will be the SVM. The high level idea behind the SVM is that it is an algorithm for maximizing a particular function with respect to a given collection data. Considering negative and positive examples, SVM creates a margin or "gap" between the two for classification. Formally, SVMs tries to maximize the marginal distance between training samples. More relevantly, for the optimization objective, it tries to find a setting of parameters where the norm of theta is small. In the end, the concept behind a support vector machine algorithm is to draw a decision boundary that divides the training data into positive and negative samples in the best optimal way, then classify new data by determining which side of the hyperplane they lie on.

In summary, supervised learning algorithms can be seen as a process that given sample data or particular input, the structure generates a desired output. Over time, in the case of all of the supervised learning techniques (e.g. linear regression, logistic regression, neural networks, support vector machines), the system learns how to model itself based upon regression and classification.

The disadvantage of using supervised learning is that for every training example, we have to provide the correct output, an in many cases, this is quite expensive [Garreta 8]. This leads us to another category of machine learning problems called unsupervised learning. In unsupervised learning, unlike supervised learning, data isn't labeled; there are no distinguishable positive or negative features. However, given a data set, the unsupervised learning algorithm may decide that the data lives within a cluster or similarly related plane. This leads us to discuss, in the next phase, the various techniques such as k means, principle component analysis.

Let's first start by discussing the K means algorithm. It is considered one of the simplest unsupervised learning algorithms for solving clustering problems. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (*k clusters*) [Tan 9]. Assume, in the problem, we are given a training set $x^{(1)}, \dots, x^{(m)}$, and want to group this data into a few clusters. Given feature vectors for each data point $x^{(i)} \in \mathbb{R}^n$ as usual; but no labels $y^{(i)}$. The goal is to predict k centroids and a label $c^{(i)}$ for each data point. The k-means clustering algorithm is:

1. Initialize cluster centroids $\mu_1\mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.

2. Repeat until convergence: {

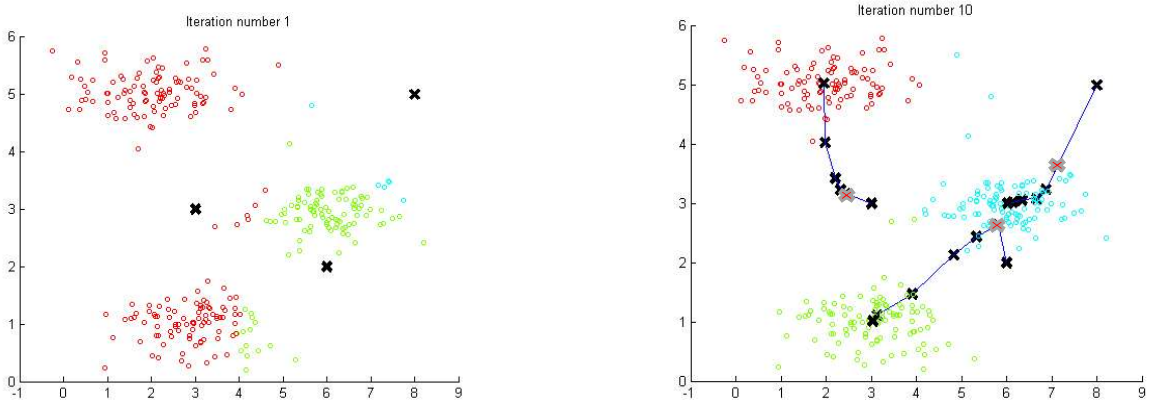
   For every *i*, set

$$c^{(i)} := \arg\min_j \left\| x^{(i)} - \mu_1 \right\|^2.$$

   For each *j*, set

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}}$$

   }



(a) Random Initialization (b) The expected output

Figure 11: K-Means

The algorithm is straight forward and simple. Let's look at an example implemented in Octave. Consider the 2-D representation in figure 11, 11a shows the initial step 1 of the algorithm. The closet centroids for the first 3 examples were 1, 3, and 2. Figure 11b shows the centroids computed after the initial finding of closet centroid for $c^1, c^2, and\ c^3$ are [2.428301, 3.157924], [5.813503, 2.633656], and [7.119387, 3.616684], respectively, indicated by the red

colored x. Figure 11b also shows the step 2 of the algorithm until convergence. It is shown that based upon the data set, it took 10 iterations for the k-means algorithm to find the optimal set of centroids in order to cluster the data.

Closely viewed, k-means is a prototype- based, partitioned clustering technique that attempts to find a user-specified number of *k clusters*, represented by centroids [Tan 9]. It is safe to note that there is no theoretical solution to find the optimal number of clusters for any given data set. Generally, multiple runs of the algorithm happens in order to find the optimum number of clusters. Due to this problem, overfitting can occur if k becomes too large.

K-Means is not the only unsupervised learning technique used to analysis clusters. In the next section, I will discuss the clustering technique principal component analysis (PCA).

To begin understanding PCA, let's begin with a 2-D dataset, figure 12, which has one direction of large variation and one of smaller variation.

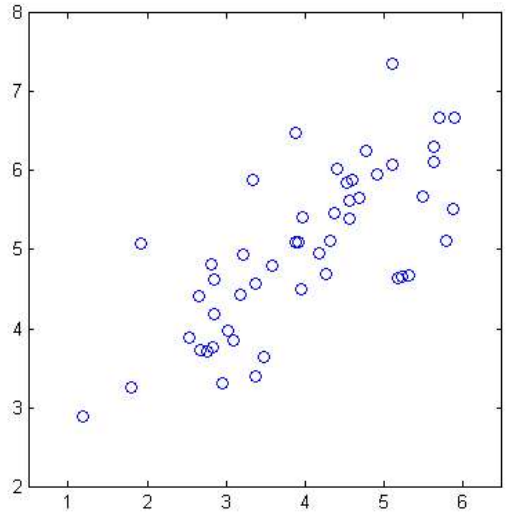


Figure 12: Example Dataset

Given this dataset, PCA tries to find a find a lower dimensional space to project data. Quite different from SVM with a Gaussian kernel. Intuitively, PCA tries to find a vector $U^{(i)}$ to

project our data. More formally, it reduces $x^{(i)} \in \mathbb{R}^2 \implies z^{(i)} \in \mathbb{R}$. Where z is the point on the line of our 1x1 matrix. Similarly, if we were to have a reduction from a three dimensional space to a two dimensional space, PCA reduces $x^{(i)} \in \mathbb{R}^3 \implies z^{(i)} \in \mathbb{R}^2$ where Z = $\begin{bmatrix} z1 \\ z2 \end{bmatrix}$

Let's take a look at the PCA algorithm. Before we begin the computational reductions we must first preprocess our data with feature scaling or normalization. The preprocessing function is of the form $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

After we have found the mean of each feature, we can then replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$ .We can assume, for simplicity, that features are of the same scale and there is no need for feature scaling. Now that we have preprocessed the data, in order to reduce from the higer dimension to the lower dimension 2 computational steps are required. Firstly, compute the "covariance matrix" it is in the form of $(sigma)\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$

Computed in Octave, the vectorized implementation of (sigma)Σ is $= \frac{1}{m}X'X;$. The next step is to compute the "eigenvectors" of matrix Σ . In octave, the command $[U, S, V] = svd(Sigma);$, is used to decompose the sigma. Where "Sigma" is an nxn matrix. Upon completion, the $svd(Sigma)$ will output three matrices U, S, and V. We only need the U matrix out of the function for reduction. U is in the form of U = $\begin{bmatrix} | & | & | \\ U^{(1)} & U^{(2)} & \dots U^{(m)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{nxn}$

Figure 13 shows the principal component after the Octave command has been executed.

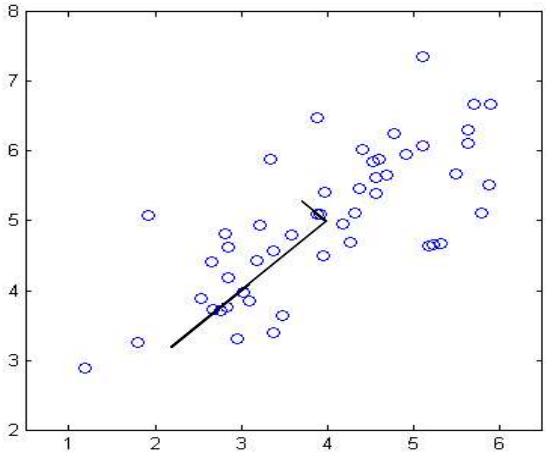


Figure 13: Computed eigenvectors of dataset

If we want to reduce down to a k-dimension / lower dimension we take the first k vectors of U. Which leaves us a vector from $U^{(1)}, U^{(1)}, .., U^{(k)}$. Giving us the k vector by which we want to project our data $x^{(i)} \in \mathbb{R}^n \implies z^{(i)} \in \mathbb{R}^k$.

After projecting the data onto the lower dimensional space, we can approximately recover the data by projecting them back on the original high dimensional space. Figure 14 shows the recovery projection approximation.

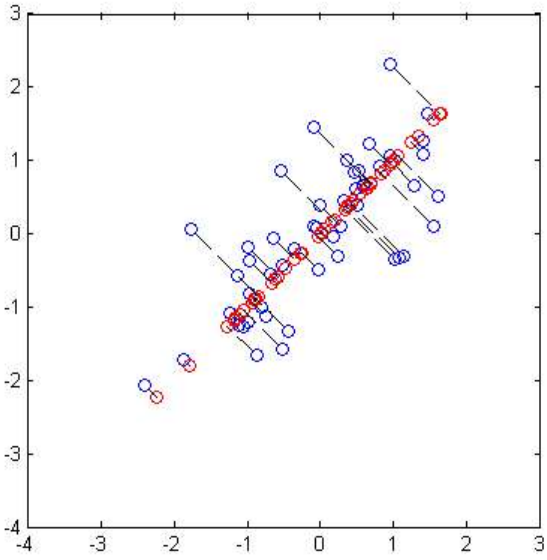


Figure 14: Normalized and projected data after PCA

In summary of principal component analysis, it is a statistical procedure that is useful to measure data in terms of its principal components rather than on a normal x-y axis. It is a power tool used in unsupervised learning when datasets are not distinguishable by labels.

In this paper, I have discussed some of the highly recognized techniques in machine learning. However, I didn't discuss what issues to look out for if an algorithm is not performing as well as expected. Usually, the root causes of poor performance is related to high bias, high variance or both. Depending on the level of bias/variance the ML algorithm can suffer from underfitting or overfitting. In the next section, I will begin to describe how to diagnose bias/variance issues.

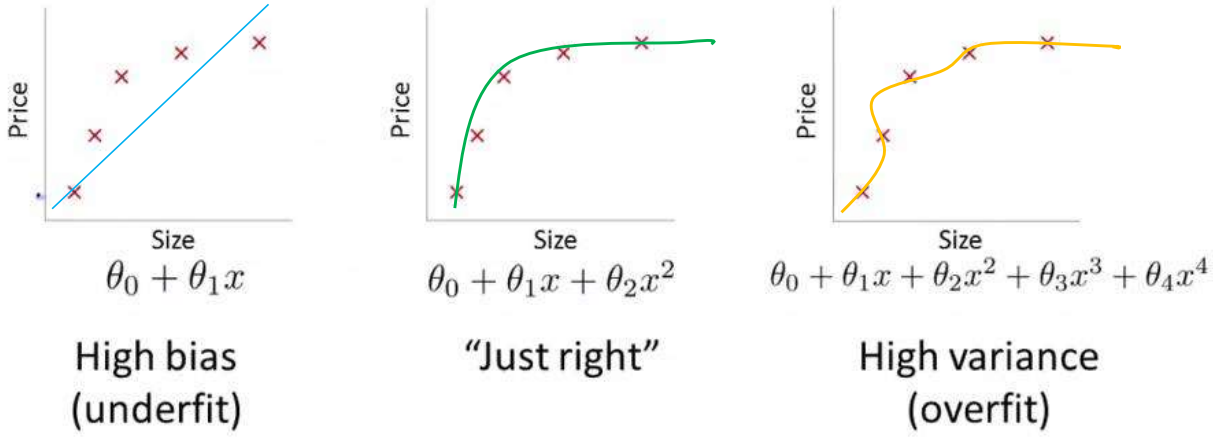


Figure 15: Bias/Variance

Looking at the figures, we can see that there is a relation between the order of polynomial and the fitting of data. Concretely, if the polynomial is low order, then the hypothesis may "underfit" the data. However, if the polynomial is of a higher order, then the hypothesis may "overfit" the data. Knowing which type of problem we have is critical when trying to improve the performance of the ML algorithm. Let's assume we created a ML algorithm from scratch.

How will we detect if the problem is based on bias or variance? Looking at the figure 15, we have what are known as the training error $J_{train}(\theta)$ and the cross validation error $J_{CV}(\theta)$. Where $J_{train}(\theta)$ is defined as $\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x)^{(i)}-(y)^{(i)}\right)^2$ and $J_{CV}(\theta)$ is defined as

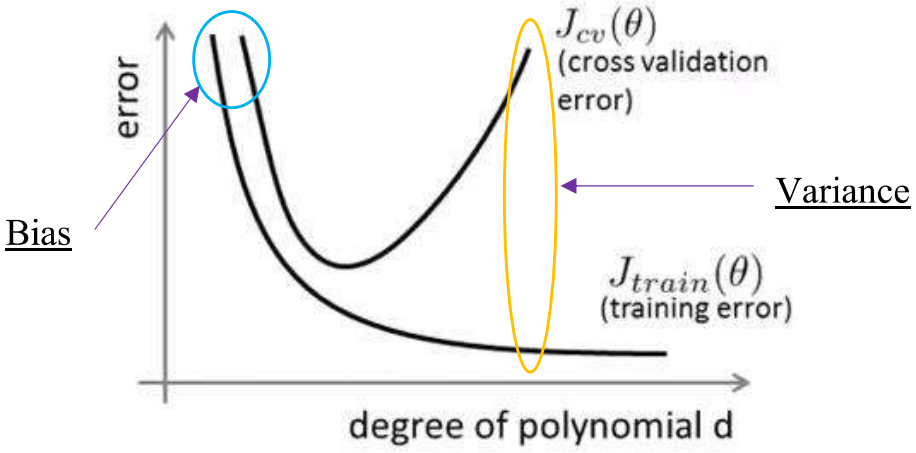


Figure 15: Bias/Variance

$$\frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right)-y_{cv}^{(i)}\right)^2$$

The blue oval indicates a high bias problem; the degree of polynomial is low, the cross validation and training error are both high. Conversely, the orange oval corresponds to a high variance; there is a high order of polynomial, high cross validation error and low training error.

In summary, by diagnosing our learning algorithm with a high Bias/Variance or both, we may be able to understand the machine learning algorithm better and improve the performance in the future.

In closing, machine learning is a widely used technique and is becoming increasingly popular. This paper has defined and describe several techniques for learning. Such as unsupervised (e.g., linear regression, neural networks) and supervised (e.g. k-means, principal component analysis) techniques. When trying to evaluate the correct approach to solving a learning problem it is essential that the right algorithm be applied. Once an algorithm has been applied, there are error checking techniques that could help test the performance and lead to a higher rate for predicting data. It was my intent in this paper, to describe, in the best way possible, the Machine Learning techniques as I understood them.

Works Cited

1. Ng, Andrew. "*Machine Learning"* Stanford University, 2012. Online Coursera.

2. Nedrich, Matt. "An Introduction to Gradient Descent and Linear Regression" : Atomic Object (June 24, 2014).

3. Lundman, Susan. "What is Logistic Regression?" *Statistics Solutions: Advancement Through Clarity,* http://www.statisticssolutions.com/what-is-logistic-regression/.*.

4. Manoj, Kumar. "Logistic Regression" *Machine Learning: Sachin Joglekar's Blog,* https://codesachin.wordpress.com/2015/08/16/logistic-regression-for-dummies/ .

5. Shiffman, Daniel. "Neural Networks" *The Nature of Code, 2012.*

6. Leverington, David. *"A Basic Introduction to Feedforward Backpropagation Neural Networks"* Texas Tech University, 2009.

7. Hajek, M., "Neural Networks", http://www.cs.ukzn.ac.za/notes/NeuralNetworks2005.pdf , 2005.

8. Garreta Raul, "A Gentle Guide to Machine Learning" MonkeyLearn, 2015, https://blog.monkeylearn.com/a-gentle-guide-to-machine-learning/.

9. Tan, Pang-Ning, "K-Means Clustering", *Introduction to Data Mining, 2006, pp. 488-568.*

10. Ding, Chris, and He, Xiofeng. "K-means Clustering via Principal Component Analysis"