A Basic Introduction to Machine Learning

Thuso Ben Modise (thuso@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Professor Bruce Basset
Co-supervised by: Dr. Navin Sivanandum
African Institute for Mathematical Sciences, South Africa

22 May 2014
Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

This paper gives an overview of techniques, methods and protocols in machine learning. The focus is on supervised learning. The techniques considered include polynomial regression, discriminant analysis, support vector machines and neural networks. It assumes a modest background in linear algebra, calculus and probability theory.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Thuso Ben Modise, 22 May 2014

Contents

1. Introduction

Machine learning is a branch of computer science and statistics that deals with algorithms that are able to learn from data. An algorithm has a hypothesis that allows it to perform some task in a certain way. It is said to learn whenever the hypotheses changes in such a way that allows the algorithm to perform the task better (Mitchell, 1997).

Machine learning is useful for certain tasks that are quite difficult to program explicitly. It would be easier to feed the algorithm some examples and allow it to adjust its hypothesis such that it will be useful for dealing with future data. Some examples of such tasks are fraudulent credit card transaction detection, visual object recognition and email spam filters.

1.1 Categories

There are three main categories of machine learning methods or algorithms.

1. Supervised learning: The goal is to predict an output with a given set of input parameters. For example given a set of labelled images, say bicycles and elephants, the algorithm could learn the features from the images of the two classes and given a new image, be able to tell whether it is more like a bicycle or more like an elephant.
2. Unsupervised learning: The goal is to find a good representation of the input data. The algorithm could be given a set of images of bicycles and elephants this time without labels. It could then look for similarities in the set and group one set of images - bicycles - to one cluster and another set to another cluster. Unsupervised learning can be used to find an underlying structure in the data or prepare it for use in supervised or reinforcement learning.
3. Reinforcement learning: The goal is to choose an action or a sequence of actions that will maximize a future pay-off. An example of this is robot navigation. Reinforcement learning is especially difficult because feedback is not immediate.

In this essay we give some of the more common methods in supervised learning, and show how some of them may be used to build an early trend detection system for topics in high energy theoretical physics.

2. Supervised Learning

Supervised learning refers to a type of machine learning algorithm which tries to predict an output from a given input. The target output may also be quantitative - taking continuous real values - for which the task becomes a regression problem. The output may also be qualitative - taking distinct values - for which the task becomes a classification problem. In this chapter we will try to outline some of the more common methods that can be applied to these problems. For a detailed treatment of these topics please refer to Hastie et al. (2009).

2.1 Linear and Polynomial Regression

In this type of problem we try to teach the algorithm to predict a continuous real valued output from an input variable of n dimensions or features. These could be, for example, height and weight of an individual. We have m examples which have the input vector $\mathbf{x}$ and correct output y. We call these training examples. Our hypotheses, the function that represents the relationship between the input and output variables, is a linear function:

$$\begin{aligned} h_\theta(\mathbf{x}) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \theta^T \mathbf{x}, \end{aligned}$$

where $\theta, \mathbf{x} \in \mathbb{R}^n$, $\theta = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$ and $\mathbf{x} = (x_0, x_1, x_2, \ldots x_n)$ with $x_0 = 1$. The θ_is are called parameters. We want to fit these parameters such that the hypothesis is as close as possible to the observed output of the training examples. Our measure of closeness will be the squared error which is defined as

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(\mathbf{x}) - y)^2. \tag{2.1.1}$$

We could possibly find a better fit for the model by fitting a polynomial function composed of the original n features. If we take a second order polynomial, the hypotheses may look like

$$\begin{aligned} h_\theta(\mathbf{x}) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \theta_{n+1} x_1^2 + \theta_{n+2} x_1 x_2 + \cdots + \theta_{n'} x_n^2 \\ &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \theta_{n+1} x_{n+1} + \theta_{n+2} x_{n+2} + \cdots + \theta_{n'} x_{n'}. \end{aligned}$$

Where $x_{n+1} = x_1^2$, $x_{n+2} = x_1 x_2 \ldots$ and $n' = n(n+1)/2$. It is worth noting that the number of terms increases considerably as you take higher order polynomials. One may decide to leave out some higher order terms in the hypothesis. As in the simple linear regression hypothesis, the cost function is equation (2.1.1). We fit the curve by finding the parameters θ_i that minimize the cost function using a suitable method. One of these methods, gradient descent, is outlined below.

We can justify the use of the squared error metric by maximum likelihood estimation, though this may not be the only reason. We assume that the relationship between each input variable $\mathbf{x}^{(i)}$ and output variable $y^{(i)}$ can be modelled by the following equation

$$y^{(i)} = \theta^T \mathbf{x}^{(i)} + \epsilon^{(i)}. \tag{2.1.2}$$

Where $\epsilon^{(i)}$ is the error term used to capture any effects that might be missed in the regression model. We further assume that the each error term $\epsilon^{(i)}$ is independent of $\mathbf{x}$ and is independently identically distributed by the Normal distribution $\mathcal{N}(0, \sigma^2)$. So we have

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(\epsilon^{(i)})^2}{2\sigma^2}\right)$$
$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right).$$

We find the parameters θ that will make the observed data most likely, thus we want to maximize the likelihood function

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta).$$

It is more convenient to maximize the log likelihood $(log(L(\theta))$ because we can break each factor into terms and deal with each term separately. Since $\log(x)$ is a strictly increasing function, the θs found in both cases will be the same.

$$\begin{aligned}
\log(L(\theta)) &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} \log \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)
\end{aligned}$$

$$\log(L(\theta)) = m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2. \tag{2.1.3}$$

We want to maximize $\log(L(\theta))$ with respect to θ. From the expression above we can see that this is the same as minimizing the squared error $J(\theta)$ in equation (2.1.1) as they only differ by a constant. Depending on the method of optimization, it may be useful to compute the derivatives of $J(\theta)$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})x_j^{(i)} \tag{2.1.4}$$

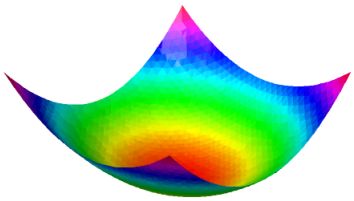
and the entries of the Hessian, the matrix of second order partial derivatives of J, are of the form

$$\frac{\partial^2 J(\theta)}{\partial \theta_j \partial \theta_k} = \sum_{i=1}^{m} x_j^{(i)} x_k^{(i)}. \tag{2.1.5}$$

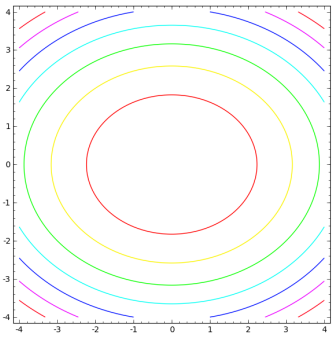
2.1.1 Gradient Descent. To find the parameters θ that minimize J we can use the gradient descent algorithm which is as follows:

Repeat until convergence:

$$\theta_j^{\text{new}} := \theta_j^{\text{old}} - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \quad \text{for all j,}$$



(a) Error surface of the least square cost function



(b) Contour plot of the least square cost function

where α is the learning rate. Because the cost function $J(\theta)$ is the square error term, the problem is convex and with the appropriate α, the algorithm should converge to the minimum. One technique that may aid in convergence is feature normalization.

Alternatively, we could find the minimum of $J(\theta)$ explicitly. Let $\mathbf{X}$ be the data matrix containing the vectors of the training set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots \mathbf{x}^{(m)}\}$ as rows, and $\mathbf{y}$ be the vector with the corresponding output values $y^{(i)}$ for each $\mathbf{x}^{(i)}$

$$\mathbf{X} = \begin{pmatrix} -(\mathbf{x}^{(1)})^T - \\ -(\mathbf{x}^{(2)})^T - \\ \vdots \\ -(\mathbf{x}^{(m)})^T - \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{2.1.6}$$

We can write our cost function in matrix form as

$$\begin{aligned} J(\theta) &= \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \\ &= \frac{1}{2}(\mathbf{X}\theta - \mathbf{y}) \cdot (\mathbf{X}\theta - \mathbf{y}). \end{aligned}$$

Now we set the first derivative of J with respect to θ to 0 to find the minimum.

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) = 0 \\ \implies -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\theta &= 0 \\ \implies \theta &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

This solution is preferable so long as computing the inverse of $(\mathbf{X}^T\mathbf{X})$ is not too computationally taxing.

2.2 Logistic Regression

For a classification problem, we are trying to predict which category a certain case will belong to given an input vector of n features. We will start with the case where the output is one of two values. We encode the output y as $y \in \{0, 1\}$ and represent our hypothesis as

$$p(y = 1|x) = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}},$$

which makes use of the sigmoid function $g(z) = 1/(1 + e^{-z})$. This choice is suited to the problem because the function can only take a range of values within $[0, 1]$. Our decision rule is to predict 0 if $h_\theta < 0.5$ and to predict an output of 1 if $h_\theta \geq 0.5$. We derive the corresponding cost function using maximum likelihood.

If

$$\begin{aligned} Pr(y = 1|x; \theta) &= h_\theta(x) \\ Pr(y = 0|x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

then

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y} \tag{2.2.1}$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)}|x; \theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

and the log likelihood can be written

$$\begin{aligned} l(\theta) &= \sum_{i=1}^{m} \left(y^{(i)} \log p(y^{(i)}|x^{(i)}; \theta) + (1 - y^{(i)}) \log(1 - p(y^{(i)}|x; \theta)) \right) \\ &= \sum_{i=1}^{m} \left(y^{(i)} \log \frac{1}{1 + e^{-\theta^T x^{(i)}}} + (1 - y^{(i)}) \log \left(\frac{e^{-\theta^T x^{(i)}}}{1 + e^{-\theta^T x^{(i)}}} \right) \right) \\ l(\theta) &= \sum_{i=1}^{m} \left(y^{(i)} \theta^T x^{(i)} - log(1 + e^{\theta^T x^{(i)}}) \right) \\ J(\theta) &= -l(\theta) = -\sum_{i=1}^{m} \left(y^{(i)} \theta^T x^{(i)} - log(1 + e^{\theta^T x^{(i)}}) \right) \end{aligned}$$

Depending on the optimization method, we may find the derivatives useful. For the sigmoid function:

$$\begin{aligned} g(z) &= \frac{1}{1 + e^{-z}} \\ \frac{dg(z)}{dz} &= \frac{e^{-z}}{(1 + e^{-z})^2} \end{aligned}$$

For the log-likelihood

$$\frac{\partial l(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} x_j^{(i)}(y^{(i)} - h_\theta(x^{(i)})) \tag{2.2.2}$$

$$\frac{\partial^2 l(\theta)}{\partial \theta_k \partial \theta_j} = -\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)})) \tag{2.2.3}$$

We could use gradient descent to find θ which would maximise $l(\theta)$. Alternatively, we could use the Newton-Raphson method.

2.2.1 Newton-Raphson method. This is a generalization of Newtons method for multidimensional spaces. The Newtons method algorithm is used to find the zeros of a function $f(\theta)$.

Repeat until convergence:

$$\theta_j^{\text{new}} := \theta_j^{\text{old}} - \frac{f(\theta)}{f'(\theta)}.$$

In our case, we want to find the zeros of the first derivative of the log-likelihood. So $f(\theta) = l'(\theta) \implies f'(\theta) = l''(\theta)$ and our algorithm becomes

Repeat until convergence:

$$\theta_j^{\text{new}} := \theta_j^{\text{old}} - \frac{l'(\theta)}{l''(\theta)}.$$

The first derivative of l, $\nabla_\theta l(\theta)$ is the vector with entries defined by equation (2.2.2) and the second derivative, or Hessian,is a matrix with entries defined by equation (2.2.3). Using these, the Newton-Raphson method is defined as

Repeat until convergence:

$$\theta_j^{\text{new}} := \theta_j^{\text{old}} - H^{-1}\nabla_\theta l(\theta).$$

The model can be extended to the case when y can be in one of K classes with $K \geq 2$. The probability that y belongs to a particular class is

$$p(y = j|x;\theta) = \frac{\exp((\theta^j)^T x)}{1 + \sum_{k=1}^{K-1} \exp((\theta^k)^T x)} \quad \text{for } j = 1 \ldots K-1$$

$$p(y = K|x) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp((\theta^k)^T x)}$$

where θ^j is the parameter vector associated with the j-th class.

For $\mathbf{x}$, $h_\theta(\mathbf{x})$ will be a $K-1$ dimensional vector with the ith entry being $p(y = i|x;\theta)$. The log likelihood is given by

$$\begin{aligned} l(\theta) &= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\theta) \\ &= \sum_{i=1}^{m} \log \prod_{k:y^{(i)}=k} \left(\frac{e^{(\theta^k)^T x^{(i)}}}{\sum_{j=1}^{K} e^{(\theta^j)^T x^{(i)}}} \right), \end{aligned}$$

which we maximize by gradient descent or another suitable method.

2.3 Discriminant Analysis

An alternative way of modelling $p(y|x)$ is by discriminant analysis. For a multi-class classification problem, suppose we know $p(x|y=k)$ belongs to a probability distribution, and $p(y)=\phi$ is the prior distribution of y, then we use Bayes theorem to get

$$p(y=k|x)=\frac{p(x|y=k)p(y=k)}{\sum_{i=1}^{n}p(x|y=i)p(y=i)}$$

Suppose the class conditional density is a multivariate Gaussian distribution with mean μ_k and covariance matrix Σ_k, then

$$p(x|y=k)=\frac{1}{\sqrt{(2\pi)^n|\Sigma_k|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma_k^{-1}(x-\mu)\right).$$

In the special case that $\Sigma_k=\Sigma \quad \forall k$, it is linear discriminant analysis. To compare two classes $y=i$ and $y=j$, we look at the log-ratio

$$\begin{aligned}\log\frac{p(y=i|x)}{p(y=j|x)}&=\log\frac{p(x|y=i)}{p(x|y=j)}+\log\frac{p(y=i)}{p(y=j)}\\&=\log\frac{p(y=i)}{p(y=j)}-\frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)+\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\\&=\log\frac{p(y=i)}{p(y=j)}-\frac{1}{2}(\mu_i+\mu_j)^T\Sigma^{-1}(\mu_i-\mu_j)+x^T\Sigma^{-1}(\mu_i-\mu_j)\end{aligned}\tag{2.3.1}$$

which is linear in x. This implies that the decision boundary between i and j is linear. In general the exact values of the parameters $\{\phi_k,\mu_k,\Sigma\}$ are not known. They can be estimated by

$$\begin{aligned}\hat{\mu}_k&=\sum_{i:y^{(i)}=k}\frac{x^{(i)}}{N_k}\\\hat{\phi}_k&=\frac{N_k}{m},\quad \text{where } N_k \text{ is the number of class-k observations}\\\hat{\Sigma}&=\sum_{k=1}^{K}\sum_{i:y^{(i)}=k}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T/(m-K)\end{aligned}$$

The linear discriminant functions (for each class k) are

$$\delta_k(x)=x^T\Sigma^{-1}\mu_k-\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k+\log(p(y=k))\tag{2.3.2}$$

To make a prediction given an x we use the discriminant function which gives

$$\arg\max_k(\delta_k(x))=\arg\max_k(x^T\Sigma^{-1}\mu_k-\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k+\log(p(y=k)))$$

On the other hand, if $\exists k$ such that $\Sigma_k\neq\Sigma$, then we would need to estimate each Σ_k separately and quadratic terms in x remain in equation (2.3.1). In this case we use the quadratic discriminant function

$$\delta_k(x)=-\frac{1}{2}\log|\Sigma_k|-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)+\log\phi_k$$

and we can estimate Σ_k as

$$\hat{\Sigma}_k = \frac{1}{N_k} \sum_{i:y^{(i)}=k} (x^{(i)} - \hat{\mu_k})(x^{(i)} - \hat{\mu_k})^T$$

2.4 Support Vector Machines

Another technique used typically in classification problems is support vector machines. We will start with the idea of a separating hyperplane. If we have points on an n dimensional vector space, a separating hyperplane is a surface or subspace of $n-1$ dimensions that groups the points that belong to one class on one side of the hyperplane. Suppose you have a training data set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$ with $x \in \mathbb{R}^n$ and $y \in \{-1, 1\}$. Furthermore, suppose that the data is linearly separable, that is, it is possible to draw a hyper-surface, $w^T x + b = 0$, that separates the positive $(y = 1)$ from the negative $(y = -1)$ examples. This hyper-surface is called the decision boundary. The optimization problem can be stated as follows:

$$\begin{aligned} & \max_{w,b} M \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq M, \quad \text{for } i = 1 \dots m, \\ & ||w|| = 1 \end{aligned}$$

Figure 2.2 shows a separating hyperplane, along with the margin, in a two dimensional vector space. In this case the hyperplane is a 1-dimensional line. The minimum margin is drawn in the dashed lines.

M represents the margin, the distance from the decision boundary to $x^{(i)}$. The larger this distance, the more confident we are about the prediction. The problem as stated above is non-convex. It may be convenient to modify it to another form that is convex to make the optimization tractable. We restate the problem as follows:

$$\begin{aligned} & \max_{w,b} M \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq M||w||, \quad \text{for } i = 1 \dots m. \end{aligned}$$

For any solution w, b to the above problem we can scale the parameters arbitrarily without changing M. Let $||w|| = 1/M$. We will now be maximizing $\frac{1}{||w||}$ which gives the same answer as minimizing $\frac{1}{2}||w||^2$.

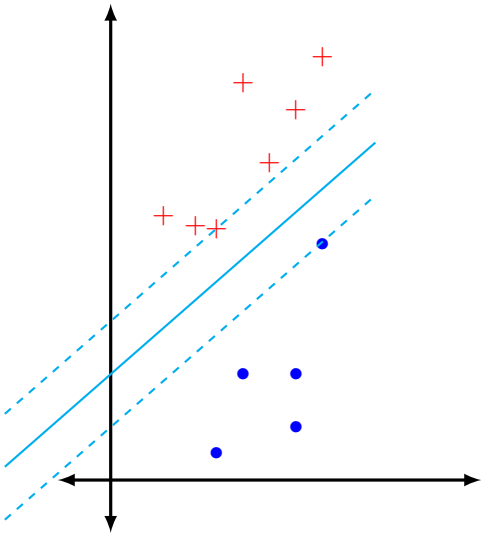
$$\begin{aligned} & \min_{w,b} \frac{1}{2}||w||^2 \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad \text{for } i = 1 \dots m. \end{aligned}$$

This is a convex optimization problem with inequality constraints. We use the method of Lagrangian multipliers to get the optimum.

2.4.1 Theorem. *Given the optimization problem*

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \textit{subject to} \quad g_i(\mathbf{x}) \leq 0 \quad \forall i,$$

Figure 2.2



the Lagrangian is defined as

$$\mathfrak{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_i \alpha_i g_i(\mathbf{x})$$

$\mathbf{x}^*$ *is a local minimum* $\implies \exists$ *a unique* $\alpha^* = \{\alpha_1^*, \alpha_2^*, \dots\}$ *such that*

$$\begin{aligned} \nabla_{\mathbf{x}} \mathfrak{L}(\mathbf{x}^*, \alpha^*) &= \mathbf{0} \\ \alpha_i^* &\geq 0 \quad \forall i \\ \alpha_i^* g_i(\mathbf{x}^*) &= 0 \quad \forall i \\ g_i(\mathbf{x}^*) &\leq 0 \quad \forall i \end{aligned}$$

The Hessian matrix of $\mathfrak{L}$ *with respect to* $\mathbf{x}$ *is positive semi definite.*

These are the Karush-Kuhn-Tucker conditions (Wikipedia).

For our problem, the Lagrange function is

$$\mathfrak{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i (y^{(i)}(w^T x^{(i)} + b) - 1). \tag{2.4.1}$$

We find the partial derivatives and equate to zero to get,

$$\mathfrak{L} = \frac{1}{2} w \cdot w - \sum_{i=1}^{m} [\alpha_i y^{(i)} w \cdot x^{(i)} + \alpha_i y^{(i)} b - \alpha_i]$$

$$\frac{\partial \mathfrak{L}}{\partial w} = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \implies w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{2.4.2}$$

$$\frac{\partial \mathfrak{L}}{\partial b} = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{2.4.3}$$

We substitute for w back in the Lagrangian function to get

$$\mathfrak{L} = \frac{1}{2}\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)} \cdot \sum_{j=1}^{m}\alpha_j y^{(j)}x^{(j)} - \sum_{i=1}^{m}\left[\alpha_i y^{(i)}(\sum_{j=1}^{m}\alpha_j y^{(j)}x^{(j)}) \cdot x^{(i)}\right] - b\sum_{i=1}^{m}\alpha_i y^{(i)} + \sum_{i=1}^{m}\alpha_i$$

The first two terms in the equation above differ by a factor and the third term is 0 by equation (2.4.3). Thus we have

$$\mathfrak{L} = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(j)})^T x^{(i)}$$

The solution to this problem must meet the Karush-Kuhn-Tucker criterion for the optimum solution. For this problem, these are (2.4.2), (2.4.3) as well as

$$\begin{aligned} \alpha_i &\geq 0 \quad \forall i \\ \alpha_i\left(y^{(i)}(w^T x^{(i)} + b) - 1\right) &= 0 \quad \forall i \end{aligned} \tag{2.4.4}$$

From here we use a quadratic programming technique to find the parameters $\{w, b, \alpha_i\}$. Equation (2.4.4) implies that for any i either $\alpha_i = 0$ or $y^{(i)}(w^T x^{(i)} + b) = 1$. In the second case, $x(i)$ just meets the minimum margin. Otherwise $y^{(i)}(w^T x^{(i)} + b) > 1$ and $x(i)$ is a distance greater than the minimum margin from the decision boundary.

We now give the modified optimization problem for the case where the data is not linearly separable. In this case we are looking for a linear hyper-surface that mostly puts one class of data in one partition with a minimum margin between the hyper-surface and the data points. The difference between this case and the linearly separable case is that we allow some points to be inside of the margin or on the wrong side of the hyperplane. The problem is as follows:

$$\begin{aligned} &\min_{w,b}\frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ &\quad \text{subject to} \quad \xi_i \geq 0, y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \quad \forall i. \end{aligned}$$

ξ_i is the slack parameter for x_i. If $\xi_i = 0$, then x_i is on the right side of the margin. If $0 < \xi_i < 1$, then x_i is classified correctly but is at a distance less than the margin. If $\xi_i > 1$ then x_i falls on the wrong side of the hyperplane. C is the tuning parameter that penalizes the use of slack variables. A large C allows very little violations of the margin. This would work if the data is separable.

We want to see the characteristics of the optimum like we did for the linearly separable case. The Lagrangian function is

$$\mathfrak{L}(w, b, \xi_i) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y^{(i)}(w^T x^{(i)} + b) - (1 - \xi_i)] - \sum_{i=1}^{m}\beta_i\xi_i.$$

We differentiate and set the partial derivatives to zero to get

$$\mathfrak{L} = \frac{1}{2} w \cdot w + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} [\alpha_i y^{(i)} w \cdot x^{(i)} + \alpha_i y^{(i)} b - \alpha_i + \alpha_i \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i$$

$$\frac{\partial \mathfrak{L}}{\partial w} = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \quad \Longrightarrow \quad w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{2.4.5}$$

$$\frac{\partial \mathfrak{L}}{\partial b} = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{2.4.6}$$

$$\frac{\partial \mathfrak{L}}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \Longrightarrow \quad \beta_i = C - \alpha_i \quad \forall i. \tag{2.4.7}$$

Substituting these back into the original Lagrangian function

$$\begin{aligned} \mathfrak{L} = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{m} \alpha_i y^{(i)} \sum_{j=1}^{m} [\alpha_j y^{(j)} x^{(j)}] \cdot x^{(i)} - b \sum_{i=1}^{m} \alpha_i y^{(i)} \\ + \sum_{i=1}^{m} \alpha_i (1 - \xi_i) - \sum_{i=1}^{m} (C - \alpha_i) \xi_i \\ = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)}. \end{aligned}$$

We want to maximize $\mathfrak{L}$ subject to $0 \geq \alpha_i \geq C$ and $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$. The solution must meet the Karush-Kuhn-Tucker conditions, which includes equations (2.4.6) (2.4.5) (2.4.7) as well as

$$\alpha_i [y^{(i)} (w^T x^{(i)} + b) - (1 - \xi_i)] = 0 \tag{2.4.8}$$

$$y^{(i)} (w^T x^{(i)} + b) - (1 - \xi_i) \geq 0 \tag{2.4.9}$$

$$\beta_i \xi_i = 0 \tag{2.4.10}$$

From equations (2.4.7), (2.4.8), (2.4.9) and (2.4.10), we can deduce that

$$\begin{aligned} \alpha_i = 0 &\implies y^{(i)} (w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C &\implies y^{(i)} (w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C &\implies y^{(i)} (w^T x^{(i)} + b) = 1 \end{aligned}$$

To extend this classifier to create non-linear decision boundaries, we can enlarge the feature space by adding polynomial terms or any other type of transformation φ on the given features $x^{(i)}$ from $\mathcal{R}^n$ to a higher dimensional space. In this case the Lagrange function to be optimized is

$$\mathfrak{L} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} (\varphi(x^{(i)}))^T \cdot \varphi(x^{(j)}).$$

The support vector machine is an extension of the support vector classifier that uses transformations on the given features using what are called kernels.

A kernel K is a function that maps data from the vector space of the input data to a higher dimensional inner product space. It can be defined explicitly as

$$K(x^{(i)}, x^{(j)}) = \left\langle \varphi(x^{(i)}), \varphi(x^{(j)}) \right\rangle = (\varphi(x^{(i)}))^T \cdot \varphi(x^{(j)}).$$

The kernel allows us to compute the inner product on the right hand side without knowing the transformation φ explicitly. All that is required is that φ exists. This makes the computation much less demanding. Some examples of kernels are

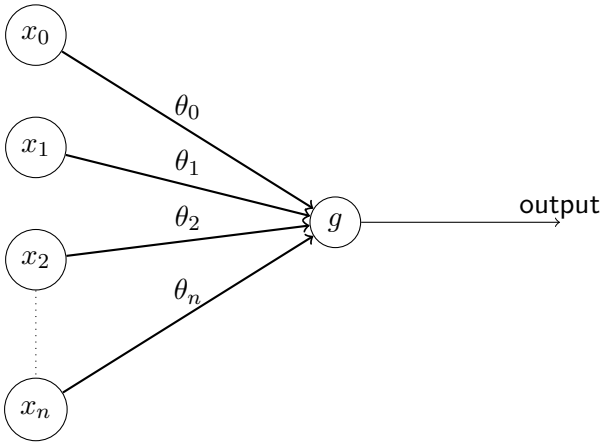
$$K(x^1, x^2) = ((x^1)^T \cdot x^2 + c)^d \tag{2.4.11}$$

$$K(x^1, x^2) = \exp\left(\frac{-||x^1 - x^2||^2}{2\sigma^2}\right) \tag{2.4.12}$$

2.5 Artificial Neural Networks

Artificial neural networks (ANN) are a set of non-linear statistical models often used in supervised learning problems. They are characterized as being able to be represented by an arrangement of computational units called neurons. In logistic regression we had our hypothesis function as $h_\theta(x) = 1/1 + e^{-\theta^T x}$. This could constitute a single neuron in an ANN as in Figure 2.3.

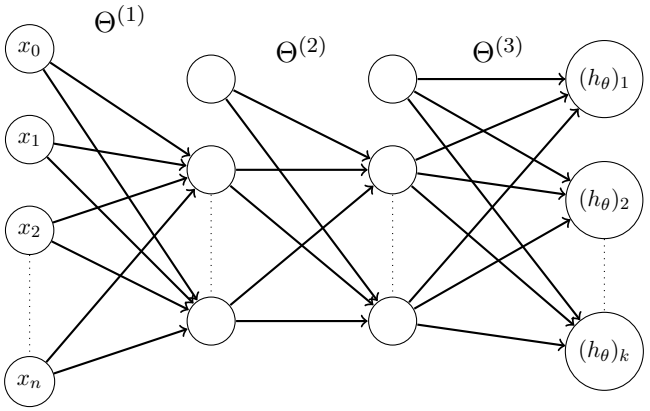
Figure 2.3: A single neuron



The x_is are the features of the input vector $\mathbf{x}$, the θ_is are the weights or parameters and g is the activation function which in this case is the sigmoid function. In a feed forward neural network, we have an arrangement of a number of units in succession. Each series is called a layer. The first layer is the input vector. It feeds in the second layer, which computes the output that feeds into the next layer and so on until the final layer computes the output.

Between each layer is a matrix of weights or parameters. The layers in between the input and output layers are called hidden layers. Figure 2.4 shows a feed forward neural network with 2 hidden layers and k outputs. Given an input of $\mathbf{x} = \{x_1, x_2, \ldots x_n\}$, the algorithm would calculate the output as follows:

Figure 2.4: A Neural Network with 2 hidden layers



$$\alpha_0 = 1$$
$$\alpha_i = g(\sum_{p=0}^{n} \Theta_{i,p}^{(1)} x_p) \quad \text{for } i = 1 \dots a$$
$$\beta_0 = 1$$
$$\beta_i = g(\sum_{p=0}^{a} \Theta_{i,p}^{(2)} \alpha_p) \quad \text{for } i = 1 \dots b$$
$$h_\theta(\mathbf{x})_i = g(\sum_{p=0}^{b} \Theta_{i,p}^{(3)} \beta_p) \quad \text{for } i = 1 \dots K$$

$\alpha = \{\alpha_0 \alpha_1, \alpha_2, \dots \alpha_a\}$ is the output of the second layer, $\beta = \{\beta_0 \beta_1, \beta_2, \dots \beta_b\}$ is the output of the third layer and $h_\theta(\mathbf{x}) = \{(h_\theta(\mathbf{x}))_1, (h_\theta(\mathbf{x}))_2, \dots (h_\theta(\mathbf{x}))_K\}$ is the final output.

Given this model, we now want to fit the parameters $\theta = \{\Theta^1, \dots\}$ such that the model is accurate given a training data set $X = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$. The cost function may be the least squared error term.

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left(y_k^{(i)} - (h_\theta(\mathbf{x}^{(i)}))_k \right)^2$$

Take

$$J_q = (\sum_{k=1}^{K} y_k^{(i)} - (h_\theta(\mathbf{x}^{(q)}))_k)^2$$

To minimize the cost, we use gradient descent which, in this situation is referred to as back-propagation.

The parameters we want to fit are

$$\Theta^{(1)} = \begin{pmatrix} \Theta^{(1)}_{1,0} & \dots & \Theta^{(1)}_{1,n} \\ \Theta^{(1)}_{2,0} & \dots & \Theta^{(1)}_{2,n} \\ \vdots & & \\ \Theta^{(1)}_{a,0} & \dots & \Theta^{(1)}_{a,n} \end{pmatrix} \quad \Theta^{(2)} = \begin{pmatrix} \Theta^{(2)}_{1,0} & \dots & \Theta^{(2)}_{1,a} \\ \Theta^{(2)}_{2,0} & \dots & \Theta^{(2)}_{2,a} \\ \vdots & & \\ \Theta^{(2)}_{b,0} & \dots & \Theta^{(2)}_{b,a} \end{pmatrix} \quad \Theta^{(3)} = \begin{pmatrix} \Theta^{(3)}_{1,0} & \dots & \Theta^{(3)}_{1,b} \\ \Theta^{(3)}_{2,0} & \dots & \Theta^{(3)}_{2,b} \\ \vdots & & \\ \Theta^{(3)}_{K,0} & \dots & \Theta^{(3)}_{K,b} \end{pmatrix}$$

We start from the final output of one training example and perform a backward pass to compute the partial derivatives of the parameters.

$$\frac{\partial J_q}{\partial \Theta^{(3)}_{i,j}} = -2(y_i - h_\theta(\mathbf{x})^q_i) g'(\sum_{p=0}^{b} \Theta^{(3)}_{i,p}\beta_p)\beta_j$$

$$\frac{\partial J_q}{\partial \Theta^{(2)}_{i,j}} = -2(y_i - h_\theta(\mathbf{x})^q_i) g'(\sum_{p=0}^{b} \Theta^{(3)}_{i,p}\beta_p) g'(\sum_{p=0}^{a} \Theta^{(2)}_{j,p}\alpha_p)\alpha_j$$

$$\frac{\partial J_q}{\partial \Theta^{(1)}_{i,j}} = -2(y_i - h_\theta(\mathbf{x})^q_i) g'(\sum_{p=0}^{b} \Theta^{(3)}_{i,p}\beta_p) g'(\sum_{p=0}^{a} \Theta^{(2)}_{j,p}\alpha_p) g'(\sum_{p=0}^{n} \Theta^{(1)}_{j,p}x_p) x^q_j$$

We then repeat this over all training examples $\mathbf{x}^{(i)}$ and sum the partial derivatives. Then we can perform a single gradient descent update

$$\Theta^{(1)\text{new}}_{i,j} := \Theta^{(1)\text{old}}_{i,j} - \gamma \sum_{i=q}^{m} \frac{\partial J_q}{\partial \Theta^{(1)\text{old}}_{i,j}}$$

$$\Theta^{(2)\text{new}}_{i,j} := \Theta^{(2)\text{old}}_{i,j} - \gamma \sum_{i=q}^{m} \frac{\partial J_q}{\partial \Theta^{(2)\text{old}}_{i,j}}$$

$$\Theta^{(3)\text{new}}_{i,j} := \Theta^{(3)\text{old}}_{i,j} - \gamma \sum_{i=q}^{m} \frac{\partial J_q}{\partial \Theta^{(3)\text{old}}_{i,j}}$$

where γ is the learning rate. We iterate through this entire process a given number of times or until convergence.

In previous algorithms, the initial values of the parameters did not require much consideration; we could start with all the parameter values being 0 and the training algorithm would give something reasonable. In back-propagation, initializing the parameters at the same value will lead to the gradient being the same in all instances and the model will perform similar one which has no hidden layers. In practice we normally set them to small random values. This is called symmetry breaking.

3. Model Selection, Assessment and Regularization

3.1 Bias and Variance

In supervised learning, a good hypothesis is on that is able to generalize well from the training data to unseen data. A good model would be one that minimizes prediction error which is the difference between the actual value of a given example of the and the prediction from the hypothesis. Mathematically the expected prediction error is defined as

$$EPE(x) = E[(y - h_\theta(x))^2 | X = x],$$

which we can further decompose decompose. Suppose y is determined by a function $\psi(x) + \epsilon$ where ϵ is random noise with $E[\epsilon] = 0$, then

$$\begin{aligned} EPE(x) &= E[(y - \psi(x) + \psi(x) - h_\theta(x))^2] \\ &= E[(y - \psi(x))^2] + E[(\psi(x) - h_\theta(x))^2] + 2E[(\psi(x) - h_\theta(x))(y - \psi(x))] \\ &= E[\epsilon^2] + E[(\psi(x) - h_\theta(x))^2] + 2\left(E[\psi(x)y] - E[h_\theta(x)y] - E[\psi(x)^2] + E[h_\theta(x)\psi(x)]\right). \end{aligned}$$

Now the third term $2\left(E[\psi(x)y] - E[h_\theta(x)y] - E[\psi(x)^2] + E[h_\theta(x)\psi(x)]\right)$ goes to zero because

$$E[\psi(x)y] = \psi(x)^2 \quad \text{since } E[y] = \psi(x)$$
$$E[h_\theta y] = E[h_\theta(x)\psi(x) + h_\theta(x)\epsilon] = E[h_\theta(x)\psi(x)] + 0.$$

So we have

$$EPE(x) = E[\epsilon^2] + E[(\psi(x) - h_\theta(x))^2].$$

The last term can be written as

$$E[(\psi(x) - h_\theta(x))^2] = E[(\psi(x) - E[h_\theta(x)])^2] + E[([h_\theta(x)] - h_\theta(x))^2] - 2E[(\psi(x) - E[h_\theta(x)])(E[h_\theta(x)] - h_\theta(x))]$$

and the last term sums to zero

$$\begin{aligned} EPE(x) &= E[\epsilon^2] + E[(E[h_\theta(x)] - h_\theta(x))^2] + E[(\psi(x) - E[h_\theta(x)])^2] \\ &= Var(\epsilon) + Var(h_\theta(x)) + [Bias(h_\theta(x))]^2. \end{aligned} \tag{3.1.1}$$

where the bias term refers to the amount by which the hypothesis differs from the function $\psi(x)$, or in other words some part of the relationship between x and y that was not captured by h_θ. The first term in (3.1.1) is the variance of the random noise ϵ which is not affected by the choice of h_θ. Furthermore, since the second two terms are non-negative, we say that $Var(\epsilon)$ is the irreducible error since the expected prediction error can not be less than it. The variance term reflects how sensitive the model is to the training data. In practice the prediction error is estimated by evaluating the accuracy of the hypothesis on an independent data set.

The objective now is to choose a model that minimizes the bias and variance and so will be able to generalize well. In linear regression and other models described in Chapter 2, we can enlarge the feature space by adding polynomial terms constructed from the original features. This makes the model more complex and allows it to better fit the training data. This however makes the model difficult to interpret and could result in fitting noise in the model and make it fail to generalize well to unseen data. This is called over-fitting.

Thus adding complexity generally decreases the bias but increases variance.

3.2 Cross Validation

To find the right balance between variance and bias we need to test the hypothesis on a separate set of data not used for training. Usually we split the available training data $S = \{(x^{(i)}, y^{(i)})\}$ into a training set S_{train} and a cross validation set S_{cv}. A split of $70\% : 30\%$ is typically used. We then train different models $\mathcal{M}_i$ on the training set S_{train} to obtain hypothesis h_i. Then we calculate the prediction error $\epsilon(h_i)$ for each hypothesis on the validation set S_{cv} and pick the hypothesis that gives the least error.

3.2.1 K-fold Cross Validation. This is an alternative and arguably more robust way of performing cross validation. It is especially useful when training data is not abundant. We divided the training set in to K partitions. For the k-th partition, find a good subset of features that shows strong correlation with the output. We train the model on the K-1 partitions, leaving out the k-th part. We then evaluate the model by calculating the prediction error on the k-th partition. We iterate through this process for all K partitions and take the average of the prediction errors.

Finally it should be noted that cross validation is used for fixing the tuning parameter or selecting the subset. The prediction error of the model is calculated using a test data set which was not used for training or cross-validation.

3.3 Subset Selection

We may be able to improve the prediction accuracy by using a subset of features available rather than all. This would be increasing the bias for a hopefully more than proportional drop in variance. There are several ways to do this.

3.3.1 Best Subset. Suppose there are p features available for developing a model. For $k = 1, 2, \ldots, p$, fit all $\binom{p}{k}$ models that have k features. Pick the best of all the $\binom{p}{k}$ model for each k as model $\mathcal{M}_k$. Then evaluate each model $\mathcal{M}_k$ on the cross validation set S_{cv} and pick the model that performs the best. This method becomes computationally expensive for large p.

3.3.2 Forward selection. Start with a null model $\mathcal{M}_0$ that uses no features. For $k = 0, 1, \ldots, p-1$, pick the feature that when added to the current model $\mathcal{M}_k$ gives the best performance on S_{train} and call it $\mathcal{M}_{k+1}$. Evaluate each $\mathcal{M}_k$ on S_{cv} and pick the one that performs the best.

3.3.3 Backward selection. This method is similar to forward selection but goes in reverse. Start with a full model $\mathcal{M}_p$ that uses all of the features. For $k = p, \ldots, 2, 1$, pick the feature that, when removed to the current model $\mathcal{M}_k$ gives the best performance on S_{train} and call it $\mathcal{M}_{k-1}$. Evaluate each $\mathcal{M}_k$ on S_{cv} and pick the one that performs the best.

3.4 F-score

In some instances, the prediction error alone may not be satisfactory as an evaluation metric. We may have a classification problem where most of the data has negative examples, such as malignant cancer classification. A model that predicts no malignant cancer would have a high accuracy even if it misclassified all of the positive cancer cases. We may want to see the trade off between false positives and true negatives that the model misclassified. One metric which summarizes the misclassification

error is the F-score which is defined as

$$F = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where,

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$
$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

An F-score of 1 would indicate that the model has no misclassification.

3.5 Regularization

Subset selection uses some subset of the available features. Adding or dropping a feature could result in the model exhibiting a high variance. An alternative to deciding whether to use or discard a feature is to put a constraint on the coefficients of the features in the model. Two methods are described below.

3.5.1 Ridge regression. Ridge regression penalizes large values of fitted parameters. This prevents large derivatives of those parameters and result in a smoother regression line or classifier. In polynomial regression our regularized cost function will be

$$J(\theta) = \sum_{i=0}^{m}(y^{(i)} - \theta^T x^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2,$$

where λ is the complexity parameter. By convention, θ_0 is not part of the regularization term. A large λ heavily penalizes large derivatives in the regression curve and forces it to be close to a linear hyper-surface. A small λ allows greater complexity in the hypothesis function. We can find the appropriate λ through cross validation. We can minimize this cost function using gradient descent or other methods as before. The gradients may be useful:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)} + 2\lambda \sum_{j=1}^{n} \theta_j$$
$$\frac{\partial^2 J(\theta)}{\partial \theta_j \partial \theta_k} = \sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} + 2\lambda.$$

We can compute the closed form solution to the minimization of the regularized cost function. $\mathbf{X}$ is the data matrix as defined in equation (2.1.6) and similarly $\mathbf{y}$ is the corresponding output vector.

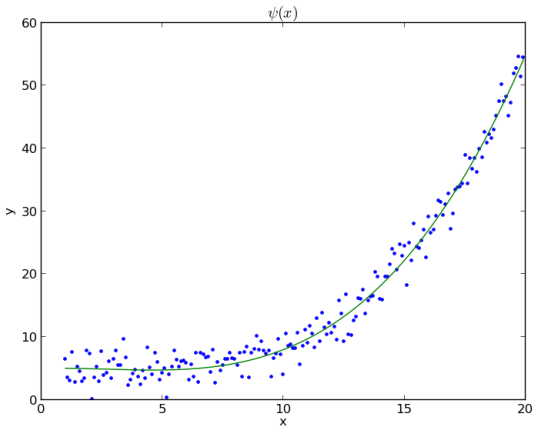
$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \frac{\lambda}{2}\theta^T\theta$$

Setting the first derivative to 0 we get

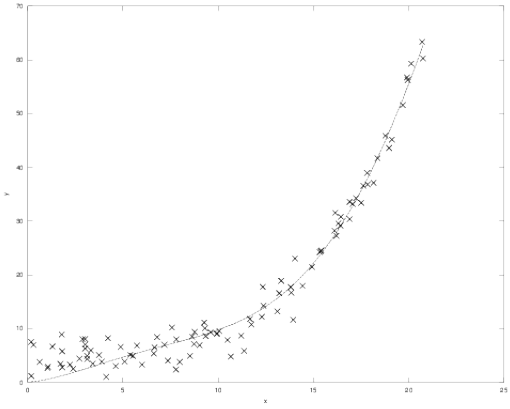
$$\frac{\partial J(\theta)}{\partial \theta} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) + \lambda\theta = 0$$
$$\implies \theta = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{y}$$

Figure 3.1

(a) This figure shows the training data with the true deterministic function fitted trough



(b) This figure shows the fit of the hypothesis on the data



3.5.2 Lasso. This is similar to ridge regression except that the parameters are penalized using the sum of absolute values instead of the sum of squares.

$$J(\theta) = \sum_{i=0}^{m} (y^{(i)} - \theta^T x^{(i)})^2 + \lambda \sum_{j=1}^{n} |\theta_j|.$$

Again, θ_0 is left out of the regularization term.

3.5.3 Example. Figure (a) in 3.1 shows simulated data using a 3rd degree polynomial (in green) with some random noise. We then tried to fit the generated data with polynomial regression. The result is shown in the second figure.

3.6 Principal Component Analysis

Occasionally one may be given a dataset where the training examples have large numbers of features. Some features may be redundant or highly correlated with other features. Principal component analysis allows us to reduce the dimension of the input data while keeping most of the variation. This may make computation easier and make it possible to visualize the data.

The idea is to find a lower dimensional subspace composed of vectors $\{v^{(1)}, v^{(2)}, \dots v^{(l)}\}$, with $l < n$, on which to project the data while minimizing the sum or orthogonal distance from the input data to the subspace. These vectors are called principal components. Given the data set as a matrix $\mathbf{X}$ as defined in equation (2.1.6), we normalize each feature so that all the features (columns of $\mathbf{X}$) are centred at 0 and have the same scale. Then we compute the covariance matrix as

$$\Sigma = \frac{1}{m}\mathbf{X}\mathbf{X}^T.$$

This is an $n \times n$ positive semi-definite matrix. The eigenvectors of Σ give the principal components. The first principal component, which is the vector showing the direction of largest variation in the data, is given by the eigenvector corresponding to the largest eigenvalue. We can rank the order of principal components by the corresponding eigenvalues. In this order we pick the first l eigenvectors to be the vectors $\{v^{(1)}, v^{(2)}, \dots v^{(l)}\}$.

To get the projection of the data onto the new subspace we compute

$$\mathbf{p} = \mathbf{V}\mathbf{X}$$

where,

$$\mathbf{V} = \begin{pmatrix} -(v^{(1)})^T - \\ -(v^{(2)})^T - \\ \vdots \\ -(v^{(l)})^T - \end{pmatrix}$$

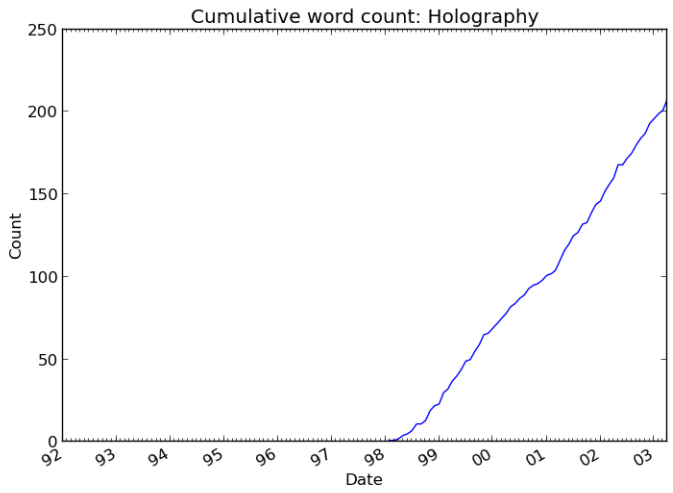
is the matrix composed of the principal components as row vectors.

4. Towards a Trend Detection System

ArXiv.org is a repository of pre-published papers in physics, astrology, mathematics, and other quantitative sciences. We collected data from the 2003 KDD cup of abstracts in high energy theoretical physics from January 1992 to April 2003 (KDD2003). The aim was to investigate possible methods of detecting trends in topics submitted for pre-publication. We would like to build a system that detects when a particular topic in theoretical physics is about to be popular or die out. This prediction will depend on the time series counts of topics of papers submitted through arXiv.

A large fraction of the time was spent sifting through the abstract text files for possible features to use. We built a dictionary based on all the individual words appearing in the title and content of the abstract. To make the data more manageable and to remove non-words (typically tex commands), we cut words with a cumulative count of less than 5. The final word count in the dictionary was at 19840. Figure 4.1 shows the cumulative count of one topic that went from making no appearances in papers submitted to having a sharp climb.

Figure 4.1



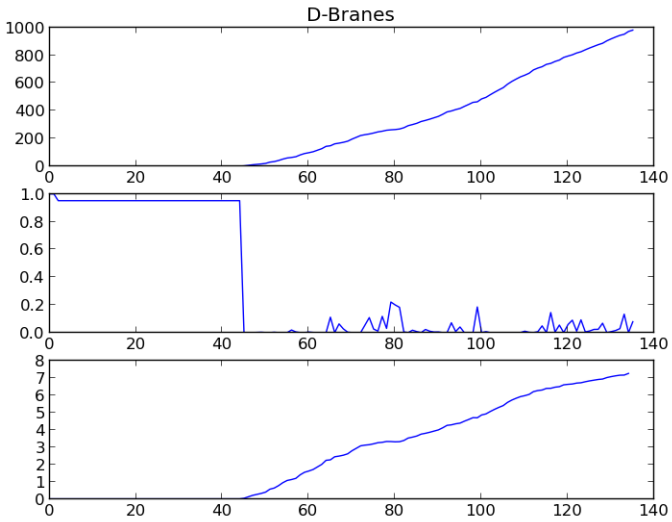
We decided to model the count of papers appearing of each word as a poisson process.

$$P[(N(t+\tau) - N(t)] = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!},$$

where $N(t)$ is the cumulative count at time (month) t, τ is the time period (number of months) in question, and k is the count for which we want to calculate the probability of observing. The parameter λ is estimated as the average of the cumulative count at time t. The decision rule as to classify a sudden change in trend would be a drop in the probability of seeing the observed count in that month.

Figure 4.2 shows the summary for the word D-branes. The top panel shows the cumulative word count, the middle panel gives the probability values and the bottom panel shows the estimated λ at each point. All variables are plotted against time in terms of number of months since January 1992.

Figure 4.2



To model this using machine learning we could use the 12 months historical count and the count for the month in question as the input parameters to a support vector machine, or any other classification algorithm. The output would be a classification indicating whether there is a significant change in trend or not. Training data would have to be created from the built dictionary. Positive examples would be words with known sudden historical changes such as the word shown in Figure 4.1. Negative examples would be common words that would typically appear in a body of text, or topics that have not had sudden changes. Labelling will be done manually by observation.

5. Discussion

In machine learning in the broader view of data science most of the time is spent on. data preparation rather than actual machine learning. This involves cleaning, normalizing and building or selecting appropriate features. In this essay we gave little treatment to this issue jumped straight into the machine learning algorithm as if data is readily available and suitable for use in learning. This was, however, one of the contributing factors that lead to the time shortage and resulted in the failure to implement a learning algorithm to the task.

5.1 Conclusion

We have discussed at length various machine learning procedures for classification and regression problems. From linear regression to artificial neural networks and support vector machines among others. We have also seen some assessment methods and testing protocols for implementing different algorithms. These methods have enjoyed much success in various real world applications. It is a field that has attracted a lot of attention due to its ability to help in finding meaning behind data.

5.2 Future Work

Due to time constraints we did not build the proposed machine learning algorithm. It would be interesting to see if the implementation would be successful.

In April 2014 there was a conference on machine learning held at the African Institute for Mathematical Sciences. Various problems were selected from previous competitions from kaggle.com. Of those problems was one in particular regarding making predictions of which devices generated sequences of accelerometer data. The group working on this problem relied heavily on random forest algorithms which were not covered in this paper. For future work we could look into these methods and see how they could be fully exploited towards building an algorithm for this task.

Acknowledgements

I would like to thank my supervisors, Bruce Basset and Navin Sivanandum, for providing the opportunity to explore the subject of this paper, and for guidance and support. I would also like to thank my tutor Evans Doe Ocansey for the help he provided in production of this essay. To my friends, thank you for the support and encouragement. I would especially like to thank Danny Parsons for being a sounding board in the final hours of the preparation. To the AIMS family I would like to say thank you for making this opportunity possible.

References

T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Springer, 2013.

KDD2003. http://www.cs.cornell.edu/projects/kddcup/datasets.html.

T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

A. Ng. Machine learning. University Lecture, 2003.

A. Ng. Coursera: Machine learning. Coursera: https://www.coursera.org/course/ml, 2014.

N. J. Nilsson. Introduction to machine learning: An early draft of a proposed textbook., 1998. URL http://robotics.stanford.edu/people/nilsson/mlbook.html.

Wikipedia. Karush-kuhn-tucker conditions. Wikipedia, the Free Encyclopedia, http://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions, Accessed May 2014.