CS229 Machine Learning

Lecture Notes

Created by Blake Cecil

1 Supervised learning

Notation $x^{(i)}$ is used to denote input variables, also called **features** and $y^{(i)}$ to denote output or **target** variables. A pairing $(x^{(i)}, y^{(i)})$ is called a **trainingexample** and the full data set we use for learning is a list of m training examples which we call a **trainingset**. We use $\mathcal{X}, \mathcal{Y}$ to denote the space of inputs and outputs respectively.

We can now describe the supervised learning problem more formally. Given a training set we aim to learn a function $h : \mathcal{X} \to \mathcal{Y}$ so that $h(x)$ is a good predictor of the corresponding value of y. We sometimes call this function h the **hypothesis**.

When the target variable we're trying to predict is continous we call the problem a **regeression** problem, if it is discrete we call it a **classification** problem.

1.1 Linear Regression

In order to perform supervised learning we must come up with a sensible representation of the hypothesis. Suppose we choose the following simple representation approximating the target as a linear combination of the $x^{(i)}$'s and some parameters θ

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots \theta_n x_n$$

The θ_i's are also called **weights**. We can introduce the following more compact notation

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

in other words h is simply the familiar dot product of two vectors. Now we reach the central question, how do we learn which values of θ are most appropriate? To answer this we need a way of quantifying how "good" a particular choice of theta is. We can do this by measuring how close each $h_\theta(x^{(i)})$ is to the corresponding $y^{(i)}$.

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

This is know as a **costfunction**. The particular equation given here gives rise to the **ordinary least squares** model of regression.

1.2 LMS Algorithm

We want to minimize this cost function and will do so by searching through the space of possible choices for θ. We employ a natural method from vector calculus to do so, **gradient descent**.

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

Where this rule is applied for all values of $j = 0..n$ where n is the size of the data set. We also introduce α to be a specially tuned constant we call the **learning rate**. Conceptually this algorithm takes the current point we are at and computes the direction of steepest decrease of J. All that remains is to derive the partial derivative of J. Instead we will handle the special case of one training example (x, y).

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial J(\theta)}{\partial \theta_j}\frac{1}{2}(h_\theta(x) - y)^2 \\ &= (h_\theta(x) - y)\frac{\partial J(\theta)}{\partial \theta_j}h_\theta(x) - y) \\ &= (h_\theta(x) - y)\frac{\partial J(\theta)}{\partial \theta_j}\sum_{i=0}^{n}\theta_i x_i - y \\ &= (h_\theta(x) - y)x_j\end{aligned}$$

So our update rule becomes

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

This rule is known as the **LMS** update rule or **Widrow – Hoff** learning rule. Its basic properties are nice in the sense that the magnitude of the update is proportional to the error, i.e if we aren't very far off we only take small steps, and if we are far off we take larger ones. Despite the rule being applicable only to a single training example we can extend it in the following manner

```
1  Repeat until convergence {
2       θ_j := θ_j + α Σ_{i=1}^{m} (y^(i) − h_θ(x^(i))) x_j^(i)  (∀j)
3  }
```

Which gives us the gradient descent of the original function J. Notice that this method requires looking at the entire training set on every step, giving it the name **batch gradient descent**. This has its payoffs however because the function is guaranteed to find the global minimum (this is because J is a convex quadratic function).

There is an alternative to batch gradient descent that avoids processing the entire set. This is called **stochastic gradient descent** or **incremental gradient descent**, and is given in the following algorithm

```
1  Loop {
2      for i=1 to m {
3          θ_j := θ_j + α(y^(i) − h_θ(x^(i))) x_j^(i)  (∀j)
4      }
5  }
```

SGD can often converge quicker than BGD, but it is possible for it to never converge to a minimum despite getting close enough for practical purposes. Generally when large training sets are used SGD is preferred.

1.3 The Normal Equations

It turns out that in the special case of least squares, we can actually explicitly solve for the optimal value of θ, but first we will need to introduce some notation to help with taking the matrix derivatives.

1.3.1 Matrix Derivatives

Let f be a function $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ mapping from matrices to real numbers, then we define the derivative of f with respect to A to be

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

As we can see the gradient is itself a $m \times n$ matrix. We now introduce the **trace** operator denoted "tr" and defined on square matrices as

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii}$$

We now state some useful identities about matrix derivatives without proof

$$\begin{aligned} \nabla_A \mathrm{tr}AB &= B^T \\ \nabla_{A^T} f(A) &= (\nabla_A f(A))^T \\ \nabla_A \mathrm{tr}ABA^TC &= CAB + C^TAB^T \\ \nabla_A |A| &= |A|(A^{-1})^T \end{aligned}$$

Where $B, C \in \mathbb{R}^{m\times n}$ are fixed matrices and $|A|$ is the determinant of A.

1.3.2 Deriving the Normal Equations

We can now derive the normal equations by first reformulating the least squares problem in terms of matrix equations. Given the training set we define X to be the $bfdesign\ matrix$ which is an $m \times n + 1$ matrix containing the training examples' inputs as its rows

$$X \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \cdots \\ -(x^{(m)})^T- \end{bmatrix}$$

Let $\vec{y}$ be the m dimensional vector containing all the target values from the training set

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(m)} \end{bmatrix}$$

using the fact that $h_theta(x^{(i)}) = (x^{(i)})^T\theta$ we have that

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ (x^{(2)})^T\theta \\ \cdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} x^{(i)} \\ \cdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \cdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix} \end{aligned}$$

Finally we have that

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)} - \vec{y}^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

Finally minimizing J yields

$$\nabla_\theta J(\theta) = X^T X\theta - X^T\vec{y}$$

setting equal to 0 and solving for theta we obtain the normal equations

$$\theta = (X^T X)^{-1}X^T\vec{y}$$

2 Locally Weighted Linear Regression

In locally weighted linear regression we aim to fit data that does not follow a linear relationship by trying to fit neighborhoods of the data instead of the whole set. The main change is that we now seek to minimize

$$\sum_{i=1}^{m} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$$

here the $w^{(i)}$'s are non negative **weights**, roughly speaking the larger the $w^{(i)}$'s the more we will try to fit that data point and the smaller the weight the less we try. A typical choice for the

weights is

$$w^{(i)} = \exp(-\frac{(x^{(i)} - x)^2}{2\tau^2})$$

The weighting function depends on x the particular point were trying to evaluate. This will bias our attention towards training examples close to x. τ helps tune the radius around x that we are interested in and is called the **bandwidth** parameter. LWLR is the first example of a **non parametric** algorithm. This is a reference to the fact that our predictions are parameterized by the data set. In linear regression once the θ's were determined we could store them and no longer needed the testing data to make future predictions. With LWLR we need to keep the training set around, this means that the amount of information need to construct a hypothesis grows linearly with the size of the data set.

3 Classification and Logistic Regression

We now examine a new type of problem that leads naturally to logistic regression. Instead of the $y^{(i)}$'s taking on values in the reals, we consider only discrete values. Here we will examine the special case of only binary values i.e the **binary classification problem**. Often 0 is called the **negative** class and 1 is called the **positive** class, given a particular $x^{(i)}, y^{(i)}$ is called the **label** for that example.

This new problem requires a more appropriate hypothesis function called the **logistic function** or the **sigmoid function**

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

This function has the nice property of squishing its inputs into the range $[0, 1]$. Before proceeding we first obtain the derivative of the sigmoid function as follows

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

In order to best fit θ we derive the maximum likelihood estimator using the following assumptions

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

This can be represented more compactly as

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

Assuming independence of the training examples we can calculate the likelihood of the parameters as follows

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

It is easier to maximize the log likelihood so we define the new function

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log 1 - h(x^{(i)}) \end{aligned}$$

Now since we are maximizing the likelihood we will perform gradient ascent with the following update $\theta := \theta + \alpha\nabla_\theta l(\theta)$. We will start by deriving the update rule for just one training example (x, y)

$$\begin{aligned} \frac{\partial}{\partial\theta_j} &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})\frac{\partial}{\partial\theta_j} g(\theta^T x) \\ &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})g(\theta^T x)(1-g(\theta^T x))\frac{\partial}{\partial\theta_j} g(\theta^T x) \\ &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x))x_j \\ &= (y - h_\theta(x))x_j \end{aligned}$$

So we have the following stochastic gradient ascent rule

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Note that this is only aesthetically similar to the rule for linear regression as the underlying hypothesis function is non linear.

3.1 Another Maximization Algorithm

There is a method besides gradient optimization that helps us solve the maximization problem, its key feature is that it enjoys much faster convergence. Consider Newton's method for finding the root of a function $f : \mathbb{R} \to \mathbb{R}$. We seek to find $\theta \in \mathbb{R}$ such that $f(\theta) = 0$. Using newtons method we perform the following update

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

An intuitive way to interpret this algorithm is that it approximates f near a point by using the line tangent to that point. It is then easy to determine where the tangent line has a root, and we take a step in that direction. While this process lets us find a root, we can also use it to find optimum by starting the process with the derivative of the function we are interested in. In our case we substitute $f(\theta) = l'(\theta)$

$$\theta = \theta - \frac{l'(\theta)}{l''(\theta)}$$

In the case we are interested in θ is vector valued so we require the following generalization

$$\theta := \theta - H^{-1}\nabla_\theta l(\theta)$$

Where H is the so called **Hessian** matrix whose entries are given by

$$H_{ij} = \frac{\partial^2 l(\theta)}{\partial\theta_i \partial\theta_j}$$

Newtons method can be useful because of its faster convergence, however the calculating and inverting the Hessian can be a very expensive operation, so one needs to consider whether n is small enough for Newtons method to be considerably faster. When Newtons method is applied to maximize $l(\theta)$ the method is sometimes called **Fisher scoring**.

4 Generalized Linear Models

So far we have seen two distinct flavors of learning, classification and regression. More formally the assumptions about the distributions of our data are what drive the need for different methods, and in this section we will establish this as a mathematical fact, rather than an intuition. This will be done by investigating a family of models called Generalized Linear Models (GLMs).

4.1 The Exponential Family

First we will work up to GLMs by defining the exponential family as follows

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$$

Here we call η the **natural, canonical parameter** of the distribution, $T(y)$ is the **sufficient statistic** and $a(\eta)$ is the **log partition function**. The quantity $e^{-a(\eta)}$ ensures that the distribution sums to 1.

For any fixed choice of T, a and b define an entire family of distributions parameterized by η.

We can now show that certain distributions are part of this family starting with the Bernoulli distribution. Given the mean ϕ we can specify a distribution over $y \in \{0, 1\}$ so that

$$\begin{aligned} p(y = 1; \phi) &= \phi \\ p(y = 0; \phi) &= 1 - \phi \end{aligned}$$

Now we apply some algebra to get the distribution in a form we can pattern match against the exponential family equation.

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= \exp\left(y \log \phi + (1 - y) \log(1 - \phi)\right) \\ &= \exp(\log(\frac{\phi}{1 - \phi})y + \log(1 - \phi)) \end{aligned}$$

We can now begin to match this will the exponential family components

$$\begin{aligned} \eta &= \log(\frac{\phi}{1 - \phi}) \\ T(y) &= y \\ a(\eta) &= -\log(1 - \phi) \\ &= \log(1 + e^{\eta}) \\ b(y) &= 1 \end{aligned}$$

Notice that if we solve for η in terms of ϕ we derive the sigmoid function for logistic regression, something that is no coincidence.

We can now apply the same procedure for a Gaussian distribution with $\sigma^2 = 1$

$$\begin{aligned} p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(y - \mu)^2)) \\ &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}y^2) \exp(\mu y - \frac{1}{2}\mu^2) \end{aligned}$$

Which gives us the following settings for the exponential family parameters

$$
\begin{aligned}
\eta &= \mu \\
T(y) &= y \\
a(\eta) &= \frac{\mu^2}{2} \\
&= \frac{\eta^2}{2} \\
b(y) &= \frac{1}{\sqrt{2\pi}} \exp(\frac{-y^2}{2}
\end{aligned}
$$

4.2 Constructing GLM's

We will now generalize our approach even further allowing us solve classes of modeling problems just by picking a distribution that is appropriate for the data we are trying to model.

Consider a classification or regression problem where we seek to predict the value of a random variable y as a function of some input data x, to derive a GLM we will make some fundamental assumption about the conditional distribution of y given x and our model

1. $y|x;\theta$ ExponentialFamily(η). I.e given x and θ the distribution of y follows some exponential family distribution.
2. Given x we seek to predict the expected value of $T(y)$. In most examples $T(y) = y$ so we want our prediction $h(x)$ to satisfy $h(x) = E[y|x]$.
3. The natural parameter and x are related linearly $\eta = \theta^T x$.

4.2.1 Ordinary Least Squares

We can now derive ordinary least squares as a special case of the GLM family by assuming that y is continuous and that $y|x;\theta$ is modeled by the Gaussian distribution, then we have that

$$
\begin{aligned}
h_\theta(x) &= E[y|x;\theta] \\
&= \mu \\
&= \eta \\
&= \theta^T x
\end{aligned}
$$

Note that the first equality follows from the Gaussian distribution, and the other 3 follow from facts derived or assumed above.

4.2.2 Logistic Regression

We can now derive logistic regression by assuming that $y \in \{0, 1\}$ and modeling the conditional distribution by the Bernoulli distribution.

$$
\begin{aligned}
h_\theta(x) &= E[y|x;\theta] \\
&= \phi \\
&= \frac{1}{1+e^{-\eta}} \\
&= \frac{1}{1+e^{-\theta^T x}}
\end{aligned}
$$

4.2.3 Softmax Regression

We can now look at a totally new example that arises from doing classification where the discrete response variable y can take on one of k values, i.e $y \in \{1, 2, \ldots, k\}$. We will model this using a multinomial distribution. We will parameterize over the k possible outcomes using only $k-1$ parameters since the last outcome is always uniquely determined given all other outcomes. We define $\phi_1, \ldots, \phi_{k-1}$ to be $\phi_k = p(y = i; \phi)$ and $\phi_k = p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$, note that ϕ_k is not a parameter.

We then define $T(y) \in \mathbb{R}^{k-1}$ as

$$
T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots 0 \end{bmatrix}, \ldots T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots 1 \end{bmatrix}, T(k) = \begin{bmatrix} \\ 0 \\ 0 \\ \vdots 0 \end{bmatrix}
$$

We no longer have that $T(y) = y$ and we will write $(T(y))_i$ to denote the ith element of the resulting vector. We can now show the multinomial is a member of the exponential family.

$$
\begin{aligned}
p(y;\phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \ldots \phi_k^{1\{y=k\}} \\
&= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \ldots \phi_k^{1-\sum_{i=1}^{k-1}(T(y))_i} \\
&= \exp((T(y))_1 \log(\phi_1) + \cdots + (1 - \sum i = 1^{k-1}(T(y))_i) \log(\phi_k)) \\
&= \exp((T(y))_1 \log(\phi_1/\phi_k) + \cdots + log(\phi_k))
\end{aligned}
$$

Then we have that

$\mathrm{a}(\eta) = -\log(\phi_k)$
$b(y) = 1$

To derive the hypothesis function we first note that

$$\eta_i = \log \frac{\phi_i}{\phi_k}$$
$$e_i^{\eta} = \frac{\phi_i}{\phi_k}$$
$$\phi_k e_i^{\eta} = \phi_i$$
$$\phi_k \sum i = 1^k e^{\eta_i} = \sum_{i=1}^{k} \phi_i = 1$$
$$\phi_i = \frac{e^{\eta_i}}{\sum j = 1^k e^{\eta_j}}$$

And using our third assumption about linearity yields the final hypothesis

$$h_\theta(x) = E[T(y)|x;\theta]$$
$$= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ phi_{k-1} \end{bmatrix} = \begin{bmatrix} \frac{e^{\theta_1^T x}}{\sum j=1^k e^{\theta_j^T x}} \\ \frac{e^{\theta_2^T x}}{\sum j=1^k e^{\theta_j^T x}} \\ \vdots \frac{e^{\theta_{k-1}^T x}}{\sum j=1^k e^{\theta_j^T x}} \end{bmatrix}$$

5 Generative Learning Algorithms

Thus far the learning algorithms we have studied have attempted to model $p(y|x;\theta)$ we call such algorithms **discriminative**, linear and logistic regression are such examples. In this section we will examine algorithms that attempt to model $p(x|y)$ and $p(y)$ which we call **generative** algorithms. We call $p(y)$ **class prior**. Once we have learned the class prior and the features we can use Bayes rule to derive the distribution on y given x

$$p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$$
$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

When it comes to making a particular prediction we can drop the denominator to reduce the calculation time.

6 Gaussian Discriminant Analysis

The first algorithm that we will look at is Gaussian Discriminant Analysis (GDA). In this model well assume that $p(x|y)$ is distributed according to a multivariate normal distribution.

6.1 The Multivariate Normal Distribution

The Gaussian distribution extended to n dimensions is parameterized by a mean vector $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ where Σ is symmetric and positive semi definite. The density is given by

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))$$

We have the mean given by

$$E[X] = \mu$$

and the covariance which generalizes the notion of an expected value is given by $\text{Cov}(x) = \Sigma$. If we let Σ be the identity matrix and μ be the zero vector then we obtain the standard normal distribution

6.2 The GDA Model

Assume we have a classification problem in which the features x are continuous valued variables, we then model $p(x|y)$ using the following distributions

$$\begin{aligned} y &\ \text{Bernoulli}(\phi) \\ x|y=0 &\ N(\mu_0, \Sigma) \\ x|y=1 &\ N(\mu_1, \Sigma) \end{aligned}$$

writing out the distributions formally we have

$$\begin{aligned} p(y) &= \phi^y (1-\phi)^{1-y} \\ p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)) \\ p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)) \end{aligned}$$

Here the parameters of the model are $\phi, \Sigma, \mu_0, \mu_1$. We can then take the log likelihood of the data

$$\begin{aligned} l(\phi, \Sigma, \mu_0, \mu_1) &= \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{m} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi) \end{aligned}$$

Maximizing l with respect to each parameter gives

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$
$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

It turns out that if we define $p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma)$ as a function of x then it can be expressed as

$$p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-\theta^T x)}$$

where θ is a function of $\phi, \mu_0, \mu_1, \Sigma$. This is exactly the form of the logistic regression. This leads us to ask when GDA performs better than logistic regression. GDA has the advantage of taking less time to train . The accuracy of the algorithm is fully determined by how closely $p(x|y)$ follows a Gaussian. If the fit is close then GDA is the most efficient choice of algorithm, however if the data is not Gaussian then logistic regression will perform better.

7 Naive Bayes