1 Discriminative Models

In our discussion of LDA and QDA, we focused on generative models, where we explicitly model the probability of the data $P(X,Y)$ and choose the class $c^*$ that solves $c^* = \arg\max_c P(Y = c|X)$. For example, in QDA we model $P(X|Y = c)$ as Gaussian with an estimated mean $\mu_c$ and covariance matrix $\Sigma_c$. If we choose a prior $P(Y)$, then our predicted class is $c^* = \arg\max_c P(Y = c|X) = \arg\max_c P(X|Y = c)P(Y = c)$.

With the generative method, we are modeling the distribution of the data $P(X,Y)$ - to use this model for classification, we use Bayes’ rule to maximize $P(Y = c|X)$. The generative approach is flexible and allows us to generate samples from our model from the data distribution. However, LDA and QDA can be inefficient in that they require estimation of a large number of parameters (i.e., the covariance matrices, which have $\frac{d(d-1)}{2}$ parameters). For 2-class LDA, the decision boundary is of the form $K + w^T x = 0$ where $K, w$ are estimated parameters. The decision boundary only requires $d + 1$ parameters, but we ended up estimating $O(d^2)$ parameters because we needed to determine the class-conditional Gaussian generative models. LDA is an indirect approach to classification - it estimates parameters for $P(X|Y)$ and use Bayes’ rule to compute a decision rule, which leads to the discrepancy between the number of parameters required to specify the generative model and the number of parameters required to perform classification.

This leads us to the concept of discriminative models, where we bypass learning a generative model altogether and directly learn a decision boundary for the purpose of classification. The process of constructing a discriminative model is composed of two key design choices:

1) Representation: how we represent the output of the model (for example, the output of a model could be any real-valued number that we threshold to perform classification, or the output could represent a probability)

2) Loss function: how we train and penalize errors

2 Least Squares Support Vector Machine

2.1 Model and Training

As a first example of a discriminative model, we discuss the Least Squares Support Vector Machine (LS-SVM). Consider the binary classification problem where the classes are represented by $-1$ and
One way to classify a data point $x$ is to estimate parameters $w$, compute $w^T x$, and classify $x$ to be $\text{sign}(w^T x)$ ($x$ is assumed to include a intercept term). Geometrically, the decision boundary this produces is a hyperplane, $w^T x = 0$.

We need to figure out how to optimize the parameter $w$. One simple procedure we can try is to fit a least squares objective:

$$\arg\min_w \sum_{i=1}^{n} \| y_i - \text{sign}(w^T x_i) \|^2 + \lambda \|w\|^2$$

Without the regularization term, this would be equivalent to minimizing the number of misclassified training points. Unfortunately, optimizing this is NP-hard (computationally intractable). A relaxed problem that is easy to solve is

$$\arg\min_w \sum_{i=1}^{n} \| y_i - w^T x_i \|^2 + \lambda \|w\|^2$$

This method is called the 2-class least squares support vector machine (LS-SVM).

One drawback of LS-SVM is that the hyperplane decision boundary it computes does not necessarily make sense for the sake of classification. For example, consider the following set of datapoints, color-coded according to the class:

![Diagram of data points with decision boundary]

LS-SVM will classify every data point correctly, since all the $+1$ points lie on one side of the decision boundary and all the $-1$ points lie on the other side. Now if we add another cluster of points as follows:
The original LS-SVM fit would still have classified every point correctly, but now the LS-SVM gets confused and decides that the points at the bottom right are contributing too much to the loss (perhaps for these points, \( w^T x_i = -5 \) for the original choice of \( w \) so even though they are on the correct side of the original separating hyperplane, they incur a high squared loss and thus the hyperplane is shifted to accommodate). This problem will be solved when we talk about general SVM’s.

### 2.2 Extensions

Instead of using the linear function \( w^T x \), we can use a non-linear function of our choice, such as the output of a neural network. One can imagine a whole family of discriminative binary classifiers that minimize

\[
\arg \min_w \sum_{i=1}^n \|y_i - g_w(x_i)\|^2 + \lambda \|w\|^2
\]

where \( g_w(x_i) \) can be any function that is easy to optimize. Then we can classify using the rule

\[
\hat{y}_i = \begin{cases} 
1 & g_w(x_i) > \theta \\
0 & g_w(x_i) \leq \theta 
\end{cases}
\]

Where \( \theta \) is some threshold. In LS-SVM, \( g_w(x_i) = x^T w_i \) and \( \theta = 0 \). Using a neural network with non-linearities as \( g_w \) can produce complex, non-linear decision boundaries.

We can also adapt this approach to the case where we have multiple classes. Suppose there are \( K \) classes, labeled 1, 2, ..., \( K \). One possible way to extend the approach from binary classification is to compute \( g_w(x_i) \) and round it to the nearest number from 1 to \( K \). However, this approach gives an “ordering” to the classes, even if the classes themselves have no ordering. For example, in fruit classification, suppose 1 is used to represent “peach,” 2 is used to represent “banana,” and 3 is used to represent “apple.” In our numerical representation, it would appear that peaches are less than bananas, which are less than apples. As a result, if we have an image that looks like some cross between an apple and a peach, we may simply end up classifying it as a banana. The typical
way to get around this issue is as follows: if the $i$-th observation has class $k$, instead of using the representation $y_i = k$, we can use the representation $y_i = e_k$, the $k$-th canonical basis vector. Now there is no relative ordering in the representations of the classes. This method is called one-hot vector encoding.

When we have multiple classes, each $y_i$ is a $K$-dimensional vector, so for LS-SVM, we instead have a $K \times d$ weight matrix to optimize over:

$$\arg\min_w \sum_{i=1}^n \|y_i - Wx_i\|^2 + \lambda \|w\|^2$$

To classify a datapoint, we compute $Wx_i$ and see which component $j$ is the largest - we then predict $x_i$ to be in class $j$.

3 Logistic Regression

Suppose that we have the binary classification problem where classes are represented by 0 and 1. We would like our model to output an estimate of the probability that a data point is in class 1. We can start with the linear function $w^Tx$ and convert it to a number between 0 and 1 by applying a sigmoid transformation $s(w^Tx)$, where $s(x) = \frac{1}{1+e^{-x}}$. Thus to classify $x_i$ after learning the weights $w$, we would estimate the probability as

$$P(y_i = 1|x_i) = s(w^Tx_i)$$

and classify $x_i$ as 1 if $P(y_i = 1|x_i) > .5$, 0 otherwise.

In order to train our model, we need a loss function to optimize. One possibility is least squares:

$$\arg\min_w \sum_{i=1}^n \|y_i - s(w^Tx_i)\|^2 + \lambda \|w\|^2$$

However, this may not be the best choice. Ordinary least squares regression had theoretical justifications such as being the maximum likelihood objective under Gaussian noise. Least squares for this classification problem does not have a similar justification.

The loss function we use for logistic regression is called the log-loss, or cross-entropy:

$$L_w(x,y) = \sum_{i=1}^n y_i \log\left(\frac{1}{s(w^Tx_i)}\right) + (1 - y_i) \log\left(\frac{1}{1 - s(w^Tx_i)}\right)$$

If we define $p_i = s(w^Tx_i)$, then using the properties of logs we can write this as

$$L_w(x,y) = -\sum_{i=1}^n y_i \log p_i + (1 - y_i) \log(1 - p_i)$$

For each $x_i$, $p_i$ represents our predicted probability that its corresponding class is 1. Because $y_i \in \{0, 1\}$, the loss corresponding to the $i$-th datapoint is $-\log p_i$ when $y_i = 1$ and $-\log(1 - p_i)$ when $y_i = 0$. If $p_i = y_i$, then we incur 0 loss. Otherwise we will incur some loss - for example, if the actual label is $y_i = 1$, then as we lower $p_i$ towards 0, the loss for this datapoint approaches infinity. The loss function can be derived from an information-theoretic perspective or a maximum likelihood perspective - some details are provided in the supplementary section below.
4  Supplementary: Logistic Regression Loss Function

4.1  Information Theoretic Approach

We can derive the logistic regression loss function from an information-theoretic perspective. We first consider the following problem: suppose we perform \( n \) tosses of a coin with bias \( q \). If \( X_i \) are the outcomes of the tosses where \( X_i = 1 \) if the \( i \)-th coin lands heads and 0 otherwise, we estimate the bias \( q \) as \( p = \frac{1}{n} \sum_{i=1}^{n} X_i \). We want to compute the probability of estimating \( q \) as some number at least equal to \( p \) - that is, we want the probability of confusing the true distribution \( \text{Bernoulli}(q) \) with the estimated distribution \( \text{Bernoulli}(p) \). We can upper bound this probability using a Chernoff bound:

\[
P(\sum_{i=1}^{n} X_i \geq np) \leq \frac{E(e^\sum_{i=1}^{n} X_i)}{e^{np}} = \left( \frac{(1-q) + q e^t}{e^p} \right)^n
\]

By taking a log and derivative, we find that the \( t \) that minimizes this upper bound satisfies

\[
e^t = \frac{p(1-q)}{q(1-p)}
\]

By plugging this into the above and performing some simplification, we obtain

\[
P(\sum_{i=1}^{n} X_i \geq np) \leq e^{-n(p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q})}
\]

We see that the dependence of this upper bound on \( p \) and \( q \) is in the term \( p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q} \) - we call this the Kullback-Leibler (KL) Divergence of \( P \) from \( Q \), denoted \( D_{KL}(p||q) \) (the notation is to make it clear that \( D_{KL} \) is not symmetric, i.e. \( D_{KL}(p||q) \neq D_{KL}(q||p) \) in general). In the context of classification, if the class label \( y_i \) is interpreted as the probability of being class 1, then logistic regression provides an estimate \( p_i \) of the probability that the data is in class 1. By setting \( p = y_i \) and \( q = p_i \) in the formula for KL divergence, we get the quantity

\[
y_i \ln \frac{y_i}{p_i} + (1-y_i) \ln \frac{1-y_i}{1-p_i} = -(y_i \ln p_i + (1-y_i) \ln (1-p_i)) + (y_i \ln y_i + (1-y_i) \ln (1-y_i))
\]

We are allowed to modify our estimates \( p_i \). But this means that the \( (y_i \ln y_i + (1-y_i) \ln (1-y_i)) \) component of the KL divergence is a constant, independent of our changes to \( p_i \). If we want to minimize the total KL divergence of the \( y_i \) from \( p_i \), we thus compute

\[
\min -\sum_{i=1}^{n} y_i \ln p_i + (1-y_i) \ln (1-p_i)
\]

This is the cross-entropy loss! Thus logistic regression computes estimates \( p_i = s(w^T x_i) \) and optimizes over \( w \) such that the KL divergence of \( y \) from \( p \) is minimized. It is worth noting that in information theory, the constant quantity \(- (p \ln p + (1-p) \ln (1-p)) \) is called the entropy of \( p \), denoted \( H(p) \), and so the cross-entropy between \( p \) and \( q \), denoted \( H(p,q) \), can be written as

\[
H(p,q) = H(p) + D_{KL}(p||q)
\]