CS 189 Introduction to Machine Learning
Fall 2017 Note 15

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 maximizes the posterior probability: $c^* = \arg\max_c P(Y = c|X)$. For example, in QDA we model $P(X|Y = c)$ as a Gaussian with an estimated mean μ_c and covariance matrix Σ_c. If we choose a prior $P(Y)$, then our predicted class for a new test point x is:

$$c^* = \arg\max_c P(Y = c|X = x) = \arg\max_c P(X = x|Y = c)P(Y = c)$$

The generative approach is flexible and we can actually use our model to generate new samples. However, LDA and QDA can be inefficient in that they require estimation of a large number of parameters (ie. the covariance matrices, which have $\frac{d(d-1)}{2}$ parameters). For 2-class LDA, the decision boundary is of the form

$$w^T x + k = 0$$

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 $+1$. One way to classify a data point x is to estimate parameters w, compute $w^T x$, and classify x to be $\text{sign}(w^T x)$. 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:

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

Where $x_i, w \in \mathbb{R}^{d+1}$. Note that we have not forgotten about the bias term! Even though we are dealing with d dimensional data, x_i and w are $d+1$ dimensional: we add an extra "feature" of 1 to x, and a corresponding bias term of k in w. Note that in practice, we do not want to penalize the bias term in the regularization term, because the we should be able to work with any affine transformation of the data and still end up with the same decision boundary. Therefore, rather than taking the norm of w, we often take the norm of w', which is every term of w excluding the corresponding bias term. For simplicity of notation however, let's just take the norm of w.

Without the regularization term, this would be equivalent to minimizing the number of misclassified training points. Unfortunately, optimizing this is NP-hard (computationally intractable). However, we can solve a relaxed version of this problem:

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

This method is called the *2-class least squares support vector machine* (LS-SVM). Note that in this relaxed version, we care about the magnitude of $w^T x_i$ and not just the sign.

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 data points, color-coded according to the class:

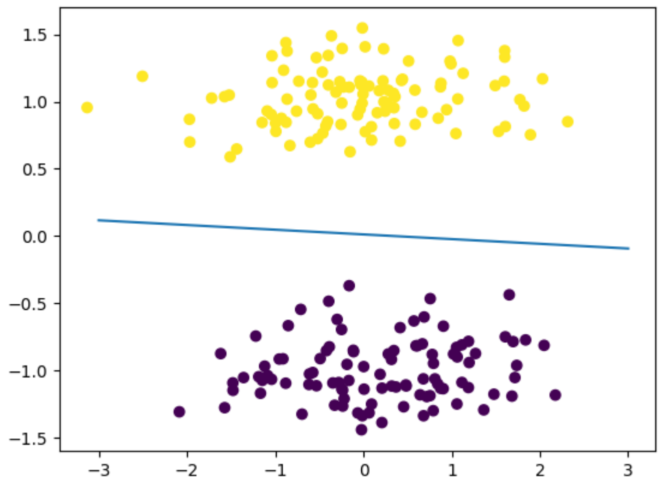

Figure 1: Reasonable fit LS-SVM

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:

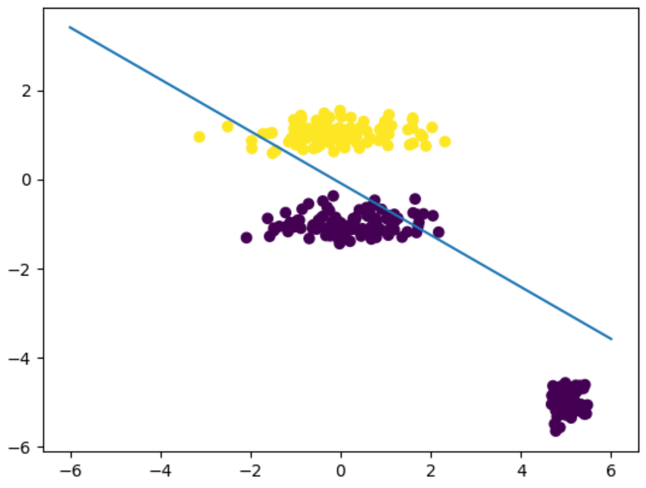

Figure 2: Poor fit LS-SVM

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 introduce **general**

Support Vector Machines (SVM's).

2.2 Feature Extension

Working with linear classifiers in the raw feature space may be extremely limiting, so we may consider adding features that that allow us to come up with nonlinear classifiers (note that we are still working with linear classifiers in the augmented feature space). For example, adding quadratic features allows us to find a linear decision boundary in the augmented quadratic space that corresponds to a nonlinear "circle" decision boundary projected down into the raw feature space.

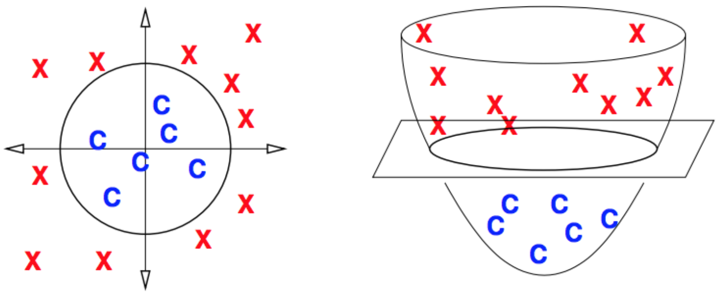

Figure 3: Augmenting Features, image courtesy of Prof. Shewchuk

In order implement this idea, we re-express our objective as

$$\arg\min_{w} \sum_{i=1}^{n} \|y_i - w^T \phi(x_i)\|^2 + \lambda \|w'\|^2$$

Note that ϕ is a function that takes as input the data in raw feature space, and outputs the data in augmented feature space.

2.3 Neural Network Extension

Instead of using the linear function $w^T x$ or augmenting features to the data, we can also directly use a non-linear function of our choice in the original feature space, such as 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

$$\hat{y}_i = \begin{cases} 1 & g_w(x_i) > \theta \\ -1 & g_w(x_i) \le \theta \end{cases}$$

Where θ 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.

2.4 Multi-Class Extension

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 natural ordering. This is clearly a problem. 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 one-hot vector, so for LS-SVM, we instead have a $K \times (d+1)$ weight matrix to optimize over:

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

To classify a data point, 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

Logistic regression is a discriminative classification technique that has a direct probabilistic interpretation. Suppose that we have the binary classification problem where classes are represented by 0 and 1. Note that we instead of using $-1/+1$ labels (as in LS-SVM), in logitistic regression we use $0/1$ labels. Logistic regression makes more sense this way because it directly outputs a probability, which belongs in the range of values between 0 and 1.

In logistic regression, 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 $w^T x$ and convert it to a number between 0 and 1 by applying a sigmoid transformation $s(w^T x)$, where $s(x) = \frac{1}{1+e^{-x}}$. Thus to classify x_i after learning the weights w, we would estimate the probability as

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

and classify x_i as

$$\hat{y}_i = \begin{cases} 1 & \text{if } s(w^T x_i) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

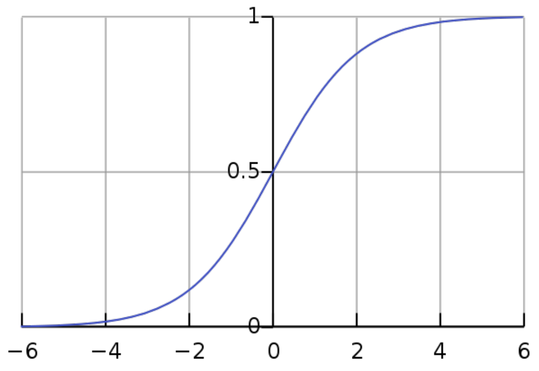

Figure 4: Logistic function. For our purposes, the horizontal axis is the output of the linear function $w^T x_i$ and the vertical axis is the output of the logistic function, which can be interpreted as a probability between 0 and 1.

The classifier equivalently classifies x_i as

$$\hat{y}_i = \begin{cases} 1 & \text{if } w^T x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

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

$$L(w) = \sum_{i=1}^n y_i \ln\left(\frac{1}{s(w^T x_i)}\right) + (1 - y_i) \ln\left(\frac{1}{1 - s(w^T x_i)}\right)$$

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

$$L(w) = -\sum_{i=1}^n y_i \ln p_i + (1 - y_i) \ln(1 - p_i)$$

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 data point is

$$L_i = \begin{cases} -\ln p_i & \text{when } y_i = 1 \\ -\ln(1 - p_i) & \text{when } y_i = 0 \end{cases}$$

Intuitively, if $p_i = y_i$, then we incur 0 loss. However, this is never actually the case. The logistic function can never actually output a value of exactly 0 or 1, and we will therefore always incur some loss. If the actual label is $y_i = 1$, then as we lower p_i towards 0, the loss for this data point approaches infinity. The loss function can be derived from a maximum likelihood perspective or an information-theoretic perspective.

3.1 Logistic Loss: Maximum Likelihood Approach

Logistic regression can be derived from a maximum likelihood model. Suppose we observe $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^n$, and each observation is sampled from a distribution $Y_i \sim Bernoulli(p_i)$, where p_i is a function of x_i. We can view $P(Y_i = 1) = P(Y = 1|x_i)$ as the posterior probability that x_i is classified as 1, and similarly $P(Y_i = 0)$ as the posterior probability that x_i is classified as 0. The observation y_i is simply sampled from this posterior probability distribution Y_i. Thus our observation y_i, which we can view as a "sample," has probability:

$$P(Y_i = y_i) = \begin{cases} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0 \end{cases}$$

One convenient way to write the likelihood of a single data point is

$$P(Y_i = y_i) = p_i^{y_i}(1 - p_i)^{(1-y_i)}$$

which holds no matter what y_i is.

We need a model for the dependency of p_i on x_i. We have to enforce that p_i is a transformation of x_i that results in a number from 0 to 1 (ie. a valid probability). Hence p_i cannot be, say, linear in x_i. One way to do achieve the 0-1 normalization is by using the sigmoid function

$$p_i = s(w^T x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

Now we can estimate the weights w via maximum likelihood. We have the problem

$$\begin{aligned} w_{LR}^* &= \arg\max_w \prod_{i=1}^n P(Y_i = y_i) \\ &= \arg\max_w \prod_{i=1}^n p_i^{y_i}(1 - p_i)^{(1-y_i)} \\ &= \arg\max_w \ln\left[\prod_{i=1}^n p_i^{y_i}(1 - p_i)^{(1-y_i)}\right] \\ &= \arg\max_w \sum_{i=1}^n y_i \ln p_i + (1 - y_i)\ln(1 - p_i) \\ &= \arg\min_w - \sum_{i=1}^n y_i \ln p_i + (1 - y_i)\ln(1 - p_i) \end{aligned}$$

3.2 Logistic Loss: Information-Theoretic Perspective

The logistic regression loss function can also be justified from an information-theoretic perspective. To motivate this approach, we introduce **Kullback-Leibler (KL) divergence** (also called **relative entropy**), which measures the amount that one distribution diverges from another. Given *any* two discrete random variables P and Q, the KL divergence from Q to P is defined to be

$$D_{KL}(P||Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$$

Note that D_{KL} is not a true distance metric, because it is not symmetric, ie. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ 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. The true class label can be viewed as a sampled value from the "true" distribution $P_i \sim Bernoulli(y_i)$. P_i is not a particularly interesting distribution because all values sampled from it will yield y_i: $P(P_i = y_i) = 1$. Logistic regression yields a distribution $Q_i \sim Bernoulli(p_i)$, which is the posterior probability that estimates the true distribution P_i. Be careful not to mix up the notation between the random variable P_i and the quantity p_i! In the context of this problem p_i is associated with the estimated distribution Q_i, not the true distribution P_i.

The KL divergence from Q_i to P_i provides a measure of how closely logistic regression can match the true label. We would like to minimize this KL divergence, and ideally we would try to choose our weights so that $D_{KL}(P_i||Q_i) = 0$. However, this is impossible for two reasons. First, if we want $D_{KL}(P_i||Q_i) = 0$, then we would need $p_i = y_i$, which is impossible because p_i is the output of a logistic function that can never actually reach 0 or 1. Second, even if we tried tuning the weights so that $D_{KL}(P_i||Q_i) = 0$, that's only optimizing one of the data points – we need to tune the weights so that we can collectively minimize the totality of all of the KL divergences contributed by all data points.

Therefore, our goal is to tune the weights w (which indirectly affect the p_i values and therefore the estimated distribution Q_i), in order to minimize the total sum of KL divergences contributed by all data points:

$$\begin{aligned}
w^*_{LR} &= \arg\min_w \sum_{i=1}^n D_{KL}(P_i||Q_i) \\
&= \arg\min_w \sum_{i=1}^n y_i \ln\frac{y_i}{p_i} + (1-y_i)\ln\frac{(1-y_i)}{(1-p_i)} \\
&= \arg\min_w \sum_{i=1}^n y_i(\ln y_i - \ln p_i) + (1-y_i)(\ln(1-y_i) - \ln(1-p_i)) \\
&= \arg\min_w \sum_{i=1}^n (-y_i \ln p_i - (1-y_i)\ln(1-p_i)) + (y_i \ln y_i + (1-y_i)\ln(1-y_i)) \\
&= \arg\min_w - \sum_{i=1}^n y_i \ln p_i + (1-y_i)\ln(1-p_i) \\
&= \arg\min_w \sum_{i=1}^n H(P_i, Q_i)
\end{aligned}$$

Note 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. Therefore, we are effectively minimizing the sum of the **cross entropies** $H(P_i, Q_i)$. We conclude our discussion of KL Divergence by noting the relation between KL divergence and cross entropy:

$$D_{KL}(P_i||Q_i) = H(P_i, Q_i) - H(P_i)$$

where $D_{KL}(P_i||Q_i)$ is the KL divergence from Q_i to P_i, $H(P_i, Q_i)$ is the cross entropy between P_i

and Q_i, and $H(P_i)$ is the **entropy** of P_i. Intuitively, we can think of entropy as the expected amount of "surprise" of a distribution. Mathematically, we have that

$$H(P) = \sum_x -P(x) \ln P(x) = \sum_x P(x) \ln \frac{1}{P(x)} = E_P\left[\ln \frac{1}{P(x)}\right]$$

To simplify the argument, assume that P is a Bernoulli distribution. If $P \sim Bernoulli(1)$ or $P \sim Bernoulli(0)$, there is no surprise at all because we always expect the same value every time we sample from the distribution. This is justified mathematically by the fact that entropy is minimized: $H(P) = 0$. However, when $P \sim Bernoulli(0.5)$, there is a lot of uncertainty/surprise and in fact, entropy is maximized.