1   Generative vs. Discriminative Classification

The task of **classification** differs from **regression** in that we are now interested in assigning a d-dimensional data point one of a *discrete* number of **classes**, instead of assigning it a *continuous* value. Thus, the task is simpler in that there are fewer choices of labels per data point but more complicated in that we now need to somehow factor in information about each class to obtain the classifier that we want.

Given a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ of n points, where each data point $\mathbf{x}_i \in \mathbb{R}^d$ is paired with a known discrete class label $y_i \in \{1, 2, ..., K\}$, our goal is to train a classifier which, when fed any arbitrary d-dimensional data point, classifies that data point as one of the K discrete classes.

There are two main types of classification models: generative models and discriminative models. **Generative models** have strong roots in probabilistic modeling. The idea is that we form a joint probability distribution $p(\mathbf{X}, Y)$ over the input $\mathbf{X}$ (which we treat as a random vector) and label Y (which we treat as a random variable), and we classify an arbitrary datapoint $\mathbf{x}$ with the class label that maximizes the joint probability:

$$\hat{y} = \arg\max_{k} \ p(\mathbf{x}, Y = k)$$

Generative models typically form the joint distribution by explicitly forming the following:

- A prior probability distribution over all classes:

$$P(k) = P(\text{class} = k)$$

- A conditional probability distribution for each class $k \in \{1, 2, ..., K\}$:

$$p_k(\mathbf{X}) = p(\mathbf{X}|\text{class } k)$$

Using the prior and the conditional distributions in conjunction, we have (from Bayes' rule) that maximizing the joint probability over the class labels is equivalent to maximizing the posterior probability of the class label:

$$\hat{y} = \arg\max_{k} \ p(\mathbf{x}, Y = k) = \arg\max_{k} \ P(k) \ p_k(\mathbf{x}) = \arg\max_{k} \ P(Y = k|\mathbf{x})$$

Maximizing the posterior will induce regions in the feature space in which one class has the highest posterior probability, and **decision boundaries** in between classes where the posterior probability of two classes are equal.
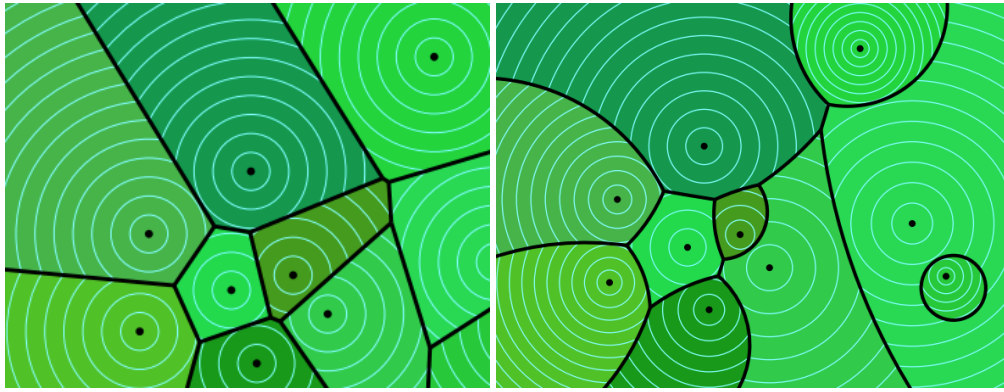
Figure 1: A collection (in dark black) of linear (left) vs quadratic (right) level set boundaries in a 2D feature space

Generative classifiers are flexible, quick to train, and can generate new samples (in order to augment the training dataset). However, they are also inefficient, because they require estimation of a large number of parameters (ie. the covariance matrices of the conditional distributions, which have $\frac{d(d+1)}{2}$ parameters). Typically, the decision boundary only requires $O(d)$ parameters, but generative models typically estimate $O(d^2)$ parameters in order to to determine the class-conditional probability distributions. As d increases, generative models tend to loose their effectiveness, as the number of parameters starts to dominate in comparison to the number of datapoints, and as a result the variance of the model increases.

This leads us to the concept of **discriminative models**, where we bypass learning a generative model altogether and directly learn a decision boundary. Discriminative models are parameterized by weights that either (1) form a posterior distribution $P(Y|\mathbf{X})$ without considering the prior or conditional distributions, or (2) directly form a hard decision boundary without considering any probabilities in the first place. In the former case, discriminative models choose the class that maximizes the posterior probability distribution:

$$\hat{y} = \arg \max_k P(Y = k|\mathbf{x})$$

Generative models also choose the class that maximizes the posterior probability distribution. The only difference is in the way generative and discriminative models form the posterior.

1.1 Bayes' Decision Rule

While both generative and discriminative models by default maximize the posterior probability over classes, this strategy may not necessarily be desirable at all times. Rather than maximizing the posterior probability, we would really like to minimize the **risk** of our model. Recall that the risk for a given classifier h is defined as the expected loss over $\mathbf{X}$ and Y:

$$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)]$$

where $\ell(h(\mathbf{x}), y)$ measures the loss between the predicted label $h(\mathbf{x})$ and the true label y. In the context of regression, the loss function was the squared error $\ell(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$. In the

context of classification, the loss function can take many forms, but the simplest is the standard step function

$$\ell(h(\mathbf{x}), y) = \begin{cases} 0 & \text{if } h(\mathbf{x}) = y \\ 1 & \text{if } h(\mathbf{x}) \neq y \end{cases}$$

Our goal is to find a classifier that minimizes the risk, given the loss function. We can equivalently express the risk as

$$R(h) = \int \left(\sum_{k=1}^{K} L(r(\mathbf{x}), k) P(Y = k|\mathbf{x}) \right) p(\mathbf{x}) d\mathbf{x}$$

The **Bayes' classifier** h^* will minimize the risk. Given an arbitrary $\mathbf{x}$, the Bayes' classifier will pick

$$h^*(\mathbf{x}) = \arg\min_j \sum_{k=1}^{K} L(j, k) P(Y = k|\mathbf{x})$$

Effectively, the Bayes' classifier will pick the class that minimizes the expected loss for the given $\mathbf{x}$. In the special case where the loss function is the standard step function (as described above),

$$h^*(\mathbf{x}) = \arg\min_j \sum_{k \neq j} P(Y = k|\mathbf{x}) = \arg\min_j 1 - P(Y = j|\mathbf{x}) = \arg\max_j P(Y = j|\mathbf{x})$$

This is equivalent to selecting the class that maximizes the posterior distribution!

Depending on which loss function we are using, the optimal classifier may or may not maximize the posterior probability. For example, consider the case of cancer diagnosis, where a patient's diagnosis for cancer can come up as positive or negative. There are four possible cases:

1. Classify the patient cancer +, and in reality the patient is cancer + (Correct Classification)

2. Classify the patient cancer −, and in reality the patient is cancer − (Correct Classification)

3. Classify the patient cancer +, but in reality the patient is cancer − (False positive)

4. Classify the patient cancer −, but in reality the patient is cancer + (False negative)

Classifying the patient's condition correctly is ideal, so we can reasonably set the loss for those cases to 0. The false positive and false negative cases are bad, and there should be a loss for these cases. But should these cases have the same loss value or should we weigh them differently? A false negative diagnosis would be significantly worse than a false positive, because a false negative diagnosis would go undiagnosed and would probably be fatal. Therefore, the associated loss for the false negative case should be higher than the associated loss for the false positive case. In this case, the goal is no longer to maximize the posterior probability, because otherwise we would be treating the false negative and false positive cases the same.

2 Least Squares Support Vector Machine

As a first example of a simple, non-probabilistic 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 $\mathbf{x}$ is to estimate parameters $\mathbf{w}$, compute $\mathbf{w}^\top\mathbf{x}$, and classify $\mathbf{x}$ to be $\text{sign}(\mathbf{w}^\top\mathbf{x})$. Geometrically, the decision boundary this produces is a hyperplane, $\mathbf{w}^\top\mathbf{x} = 0$.

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

$$\underset{\mathbf{w}}{\arg\min} \ \sum_{i=1}^{n} \|y_i - \text{sign}(\mathbf{w}^\top\mathbf{x}_i)\|^2 + \lambda\|\mathbf{w}\|^2$$

Where $\mathbf{x}_i, \mathbf{w} \in \mathbb{R}^{d+1}$. Note that we have not forgotten about the bias term! Even though we are dealing with d dimensional data, $\mathbf{x}_i$ and $\mathbf{w}$ are $d + 1$ dimensional: we add an extra "feature" of 1 to $\mathbf{x}$, and a corresponding bias term of k in $\mathbf{w}$. 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 $\mathbf{w}$, we often take the norm of $\mathbf{w}'$, which is every term of $\mathbf{w}$ excluding the corresponding bias term. For simplicity of notation however, let's just take the norm of $\mathbf{w}$.

Without the regularization term, this would be equivalent to minimizing the number of misclassified training points. Unfortunately, the "sign" term makes this optimization problem non-convex, and in fact this optimization problem is NP-hard (computationally intractable). Instead we can solve a relaxed version of this problem:

$$\underset{\mathbf{w}}{\arg\min} \ \sum_{i=1}^{n} \|y_i - \mathbf{w}^\top\mathbf{x}_i\|^2 + \lambda\|\mathbf{w}\|^2$$

This method is called the binary **least squares support vector machine** (LS-SVM). Note that in this relaxed version, we care about the magnitude of $\mathbf{w}^\top\mathbf{x}_i$ 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:
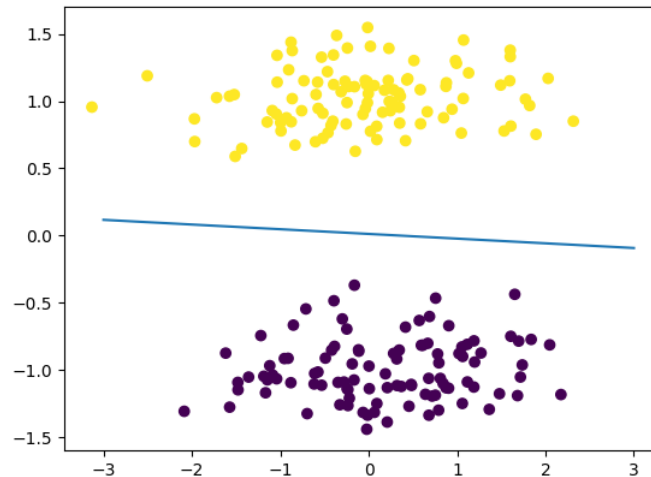
Figure 2: Reasonable fit LS-SVM

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:
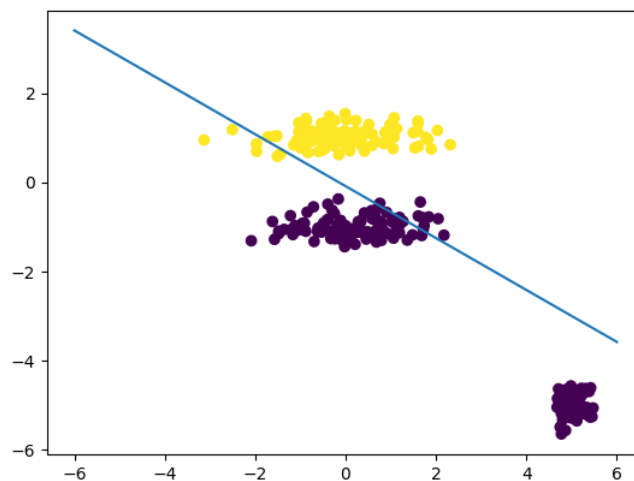

Figure 3: Poor fit LS-SVM

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, $\mathbf{w}^\top \mathbf{x}_i = -5$ for the original choice of $\mathbf{w}$ 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)**.

2.1 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.
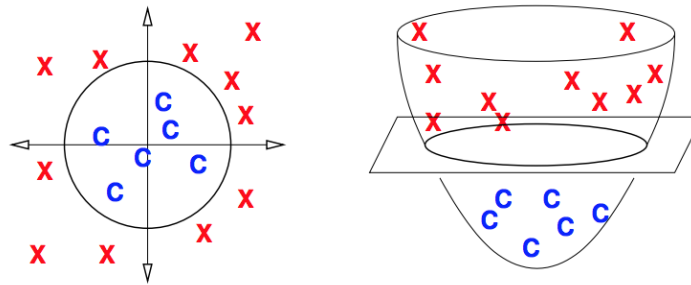

Figure 4: Augmenting Features, image courtesy of Prof. Shewchuk

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

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n} \|y_i - \mathbf{w}^\top \phi(\mathbf{x}_i)\|^2 + \lambda \|\mathbf{w}\|^2$$

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

2.2 Neural Network Extension

Instead of using the linear function $\mathbf{w}^\top \mathbf{x}$ 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

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n} \|y_i - g_{\mathbf{w}}(\mathbf{x}_i)\|^2 + \lambda \|\mathbf{w}\|^2$$

where $g_{\mathbf{w}}(\mathbf{x}_i)$ can be any function that is easy to optimize. Then we can classify using the rule

$$\hat{y}_i = \begin{cases} 1 & g_{\mathbf{w}}(\mathbf{x}_i) > \theta \\ -1 & g_{\mathbf{w}}(\mathbf{x}_i) \le \theta \end{cases}$$

Where θ is some threshold. In LS-SVM, $g_{\mathbf{w}}(\mathbf{x}_i) = \mathbf{x}^\top \mathbf{w}_i$ and $\theta = 0$. Using a neural network with non-linearities as $g_{\mathbf{w}}$ can produce complex, non-linear decision boundaries.

2.3 Multiclass 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 $g_{\mathbf{w}}(\mathbf{x}_i)$ 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 $\mathbf{y}_i = \mathbf{e}_k$, 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 $\mathbf{y}_i$ is a K-dimensional one-hot vector, so for LS-SVM, we instead have a $K \times (d + 1)$ weight matrix to optimize over:

$$\arg \min_{\mathbf{W}} \sum_{i=1}^{n} \|\mathbf{y}_i - \mathbf{W}\mathbf{x}_i\|^2 + \lambda\|\mathbf{w}\|^2$$

To classify an arbitrary input $\mathbf{x}$, we compute $\mathbf{W}\mathbf{x}$ and see which component k is the largest:

$$\hat{y} = \max_{k} \mathbf{w}_k^{\top}\mathbf{x}$$