Engineering Applications of Machine Learning and Data Analytics

Discriminant Functions

Contents

1 Introduction to Discriminant Functions

The previous set of lectures discussed how probability can be used to classify data into a discrete set of categories. For example, a dataset might be represented by pixels in an image, $\mathbf{x}$, and the label is object that is in the image, y or ω. Recall that Bayes' rule is that we decide to use the class with the greatest posterior probability:

$$\omega^* = \arg\max_{\omega\in\Omega} p(\mathbf{x}|\omega)p(\omega) = \arg\max_{\omega\in\Omega} p(\omega|\mathbf{x}) \tag{1}$$

and this rule leads to the smallest amount of risk or classification error if all mistakes are weighted the same. This is a likelihood-based approach to discriminating between multiple classes if we choose to model $p(\mathbf{x}|\omega)$, which we know can be difficult to do because of the curse of dimensionality. In using the likelihood-based approach, we can decide to model either $p(\mathbf{x}|\omega)p(\omega)$ or $p(\omega|\mathbf{x})$ and these constitute generative and discriminative models, respectively.

Generative models Generative models use the data available to estimate probabilities/probability distributions of each quantity such as priors: $p(\omega)$, likelihoods: $p(\mathbf{x}|\omega)$, and evidence $p(\mathbf{x})$. This is equivalent to attempting to directly estimate the joint distribution $p(\mathbf{x}, \omega)$ and normalizing to obtain posterior probabilities. Hence why these are called generative models because it is possible, using the estimated quantities, to generate synthetic data in the input space.

There are many cases where determining the likelihoods and priors empirically can be difficult to impossible because of the curse of dimensionality. This is especially true where the dimensionality of $\mathbf{x}$ is high. For example, imagine having a database of several thousand images that are 512×512 pixels with pictures of cats, dogs, or bears. Estimating $p(\mathbf{x}|\omega)$, requires a substantial amount of data to adequately estimate if each pixel is an element of ω.

Discriminative models Discriminative models are based on the right hand side of (1) (i.e., $p(\omega|\mathbf{x})$) and are generally easier to make because we do not attempt to fully model the joint distribution $p(\omega, \mathbf{x})$, but rather attempt to calculate/estimate the posterior directly. However, this direct estimation of the posterior prevents us from determining certain attributes of the data that could be provided with using a generative method, such as outlier detection.

Discriminant Functions Discriminant functions in contrast to Generative and Discriminative models, attempt to bypass likelihood-based approaches entirely and simply seek to provide a decision boundary by which one can classify feature vectors. Essentially, a discriminant function aims to tell you what the class is without having any idea about its probabilistic confidence in that class label.

2 Linear Discriminant Function/Regression

Linear discriminant functions are classifiers that attempt to classify data using a linear mapping of the input data $\mathbf{x}$ directly to classes ω. This method differs from discriminative models in that a function $g_i(\mathbf{x})$ is designed to classify without attempting to calculate posterior probabilities. A general form of a discriminate function of the first degree is shown in (2).

$$g_i(\mathbf{x}) = \mathbf{w}_i^\mathsf{T}\mathbf{x} + w_{0i} \tag{2}$$

where $\mathbf{w}_i$ is a weight vector and w_{0i} is a bias term. One way the function $g_i(\mathbf{x})$ can be designed to classify is by choosing a class based on the sign of the output such as (3).

$$\widehat{\omega} = \text{sign}(g_i(\mathbf{x})) \quad \text{(Linear Regression)} \tag{3}$$

How do we fit a linear model to the data? In other words, how do we determine $\mathbf{w}_i$ and w_{0i}? There are multiple methods, but the most popular is the least squares method [1]. The least squares method aims to minimize the squared error between the outputs of the discriminant functions $g_i(\mathbf{x})$ and the desired output $\boldsymbol{y}$. The loss function for this method is shown in (4).

$$L(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \mathbf{w}^\mathsf{T}\mathbf{x}_j\right)^2 = \frac{1}{2}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)^\mathsf{T}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right); \qquad \mathbf{X} \in \mathbb{R}^{n\times p} \tag{4}$$

where n is the number of data points, and p is the number of features.

To minimize the loss function, the derivative of the expression can be taken with respect to w and set equal to 0, leaving us with the following:

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}) = -\mathbf{X}^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \implies \widehat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} \tag{5}$$

However, this model imposes the following restrictions on us:

1. $\mathbf{X}^\mathsf{T}\mathbf{X}$ must be invertible, which is not guaranteed
2. Implementation: Training data $\widehat{\mathbf{w}} = (\mathbf{X}_{tr}^\mathsf{T}\mathbf{X}_{tr})^{-1}\mathbf{X}_{tr}^\mathsf{T}\mathbf{y}$, where the "tr" underscript denotes the training data; Testing Data: $y = \widehat{\mathbf{w}}^\mathsf{T}\mathbf{x}_{test}$
3. If either p or n (or both) are large, it would be computationally expensive to implement this model?

The last point can become a particular headache for several reasons. First, if n is very large we need to be able to load all of the data into memory. For data sets from the UCI repo this may not be a problem; however, for may real-world data sets this can become a burden very quickly. Second, if p is very large then we need to deal with computing the inverse of a large matrix, which could prove to be challenging. Therefore, we need to find a way to cope with these larger data sizes.

[1] As opposed to the wildly unpopular most squares method

3 Online vs Batch

Restriction #3 from Section 2 indicates that all of the data are required at the same time to compute $\widehat{\mathbf{w}}$. This leads us into two different types of algorithms that process data in different ways. Batch-based algorithms are those that process all of the data at one time. Online-based algorithms process one data point at a time. Online algorithms enable us to calculate the error for each sample, which can be determined if we have a $\mathbf{w}$. Again, using least squares error, the error for the ith sample can be determined by doing the following computations:

$$E_i = (y_i - \mathbf{w}^T\mathbf{x}_i)^2 \implies \nabla E_i = -2(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i \qquad i \in \{1, ..., n\} \tag{6}$$

where i is the ith data point out of n data points. The total error E_{total} is then simply the sum of all of the errors $E = \sum_{i=1}^{n} E_i$.

3.1 Stochastic Gradient Descent

Standard Gradient Descent (SGD) is a popular optimization algorithm that finds extrema of N-Dimensional functions by extending the basic calculus notion of finding where the derivative is zero. As one can recall, for a given function $F(\mathbf{x})$, the gradient of F, ∇F, is a vector in the direction of greatest increase of F. F is a function of the variables $\mathbf{x}$ and our objective is to minimize F. Intuitively, if we are at a point $\mathbf{x}$ and it is not a mimima then we could take the derivative to obtain the gradient. Heading in the direction of the gradient could lead you to maxima, but this is not the direction we need to go if we are to reach a mimimum. Therefore, we take a step negative to the direction of the gradient. This insight led to the development of the standard gradient descent update equation given by

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla F(\mathbf{x}_n) \tag{7}$$

where $\eta > 0$ is the *learning rate* that roughly controls the rate at which the algorithm converges as well as how close it is able to get to the minimum. The issue with standard Gradient Descent with regards to machine learning applications, though, is that $\mathbf{x}$ is commonly very highly dimensional or of large sample size, and as a result, the calculation of $\nabla F(\mathbf{x})$ in any given iteration can be a computationally intensive process since all of the gradients need to be accumulated. To help with this, an alternative algorithm called Stochastic Gradient Descent was developed with the goal of breaking down the Batch computations of the standard algorithm into an iterative online procedure that takes iterative steps negative to the gradient.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla E_i \tag{8}$$

where E_i is a loss function that characterizes the error between the ith expected value and the fitted model evaluated at the i-th data point as described above. Thus, taken together, this leads to the following algorithm for Stochastic Gradient Descent:

In SGD, T specifies the maximum number of iterations before exiting and the "check convergence" criterion can be chosen to be whatever exit metric the user desires. Commonly, it is along the lines of checking that $||\mathbf{w}_{new} - \mathbf{w}_{old}|| < \epsilon$ for some user defined ϵ.

The least squares method was chosen to derive the gradient descent algorithm because it is easy to implement. However, using the least squares method invites the possibility that overfitting may occur. Overfitting outputs a model that minimizes the least squares error by deriving a more complicated model for the data. To prevent overfitting, one method used is regularization, which just involves adding a term that is not dependent on the data to the expression as shown below:

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^P}{\arg\min}\, E_D(\mathbf{w}) + \lambda E_R(\mathbf{w}), \quad \lambda \geq 0 \tag{9}$$

Algorithm 1 Stochastic Gradient Descent

procedure SGD($\{(\mathbf{x}_i, y_i)\}_{i=1\ldots n}$)
  Initialize $\mathbf{w}_{new}$ to a random weight vector; Input the maximum number of iterations to run.
  for $t \in 1, \ldots, T$ **do**
    shuffle data
    for $i \in 1, \ldots, n$ **do**
      $\mathbf{w}_{old} = \mathbf{w}_{new}$
      $\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla E_i(\mathbf{x}_i, y_i, \mathbf{w}_{old})$
      if $\|\mathbf{w}_{old} - \mathbf{w}_{new}\|_2^2 \leq c$ **then**
        break;
      end if
    end for
  end for
end procedure

where $\lambda > 0$ is a constant chosen to impose how strongly the regularization term affects the expression, $E_D(\mathbf{w})$ is the error computed from the data and $E_R(\mathbf{w})$ is the regularization term trying to penalize larger values for $\mathbf{w}$. A popular form of regularization is L_2 regularization, where $E_R(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$. Using this term coupled with the least mean squares method results in the following equation:

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^P}{\arg\min} \frac{1}{2} \sum_{j=1}^{n} \left(y_j - \mathbf{w}^\mathsf{T}\mathbf{x}_j\right)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{10}$$

Figure 1 displays L_2 and L_1 regularization. An L_2 penalty tends to promote solutions with smaller values whereas L_1 attempts to drive more values to zero. We will discuss regularization in more detail in a later lecture.

Acknowledgement

An initial draft of this document was prepared by students in *Engineering Applications of Machine Learning and Data Analytics* in 2018.

References

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Crammer et al., 2006] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. John Wiley & Sons, Inc., 2nd edition.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[Vapnik, 1989] Vapnik, V. (1989). *Statistical Learning Theory*. Wily-Interscience.

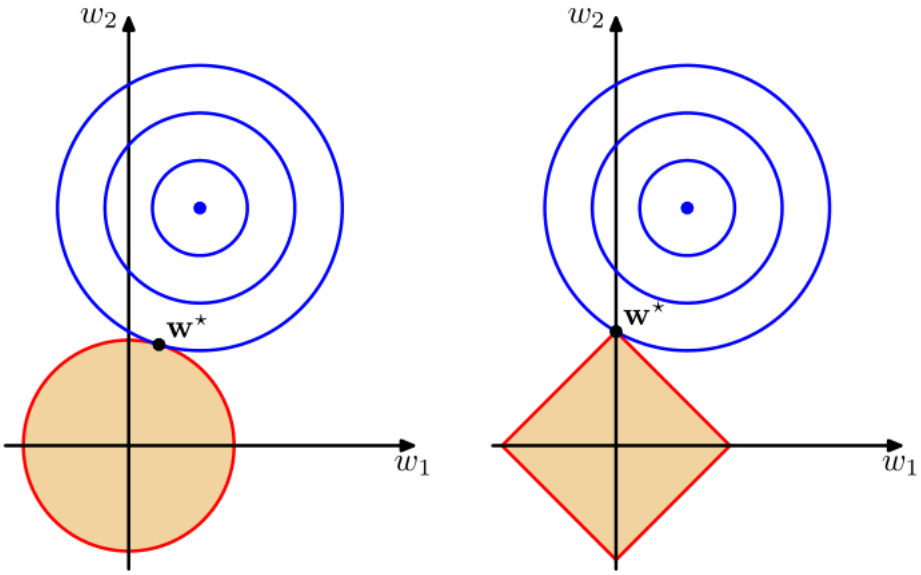

Figure 1: Comparison of L_2 and L_1 Norms [Bishop, 2006]