CSE 252A: Recognition Again

Lecturer: David Kriegman

Scribed by Owen Jow on November 27, 2018

1 Evaluating Classifiers

1.1 Binary

A binary classifier can be right in two ways (true positive tp, true negative tn) and wrong in two ways (false positive fp, false negative fn). Depending on the application, it might be worse to some degree to have false positives, worse to some degree to have false negatives, or equally bad to have either.

Among a number of metrics built around false positives and false negatives, we can evaluate the binary classifier in terms of **precision**[1] $tp/(tp+fp)$ and/or **recall**[2] $tp/(tp+fn)$.

1.2 Multiclass

We can visualize the accuracy of a multiclass classifier using a *confusion matrix*. In a confusion matrix, the rows and columns represent the classification labels in the same order, and the entry at location i, j is the proportion of times the classifier chooses label j when the answer is label i. Thus a perfect classifier will yield 0s everywhere except on the diagonal.

A confusion matrix makes it easy to see which classes are being confused for others.

2 Bayes Classifier

ω_i	event that the object is of class i	(*cause*)
$\mathbf{x}$	a vector of observed features	(*effect*)
$P(\omega_i)$	prior probability of ω_i	
$P(\mathbf{x} \mid \omega_i)$	class-conditional density function	

To make a decision, convert the likelihood $P(\mathbf{x} \mid \omega_i)$ and prior $P(\omega_i)$ into a **posterior probability**:

$$P(\omega_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \omega_i)P(\omega_i)}{P(\mathbf{x})}$$

[1] "of what I say is positive, what proportion is actually positive?" / high precision → low false positive rate

[2] "of all of the positives, how many did I get?" / high recall → low false negative rate

If there are n categories,

$$P(\mathbf{x}) = \sum_{i=1}^{n} P(\mathbf{x} \mid \omega_i) P(\omega_i)$$

so the denominator can be computed in terms of the distributions in the numerator.

- *To obtain the likelihood and prior*: determine empirically based on training data

We choose the class with the highest posterior probability $P(\omega_i \mid \mathbf{x})$ (**MAP classification**).

2.1 Analysis

The Bayes classifier is optimal, because *picking anything other than what it says to pick* can only result in an identical or higher probability of error. However, it's not really used in practice because it's hard to approximate $P(\mathbf{x} \mid \omega_i)$ – especially when $\mathbf{x}$ is high-dimensional (need a lot of samples).

3 Alternative Classifiers

3.1 Parameterized Posterior Probabilities

We can assume a certain posterior probability model, e.g. normal distribution with mean vector and covariance matrix, and then just estimate the parameters of the model. If $\mathbf{x}$ is n-dimensional, the mean vector has n parameters and the covariance matrix has $n(n+1)/2$ parameters. So in this case, the number of parameters would grow quadratically with the dimensionality of the feature space.

3.2 k-NN

Given $\mathbf{x}$, look at its k nearest neighbors in the training set. Use the most common class as $\mathbf{x}$'s class.

- As $k \to \infty$ and the dataset expands to include more and more data, k-NN approximates the posterior probability better and better and approaches Bayesian optimality.

3.3 SVM

With a Bayes classifier, the decision boundary between two classes is the set of values in observation space for which the posterior probabilities of the two classes are (1) identical and (2) higher than the probabilities of all other classes. Instead of obtaining decision boundaries as a byproduct of computing posterior probabilities, we can imagine estimating them directly using an **SVM** approach.

Let's say there are two classes, -1 and 1. We would like to find a hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ which divides the set of points $(\mathbf{x}_i, y_i)$ (where $y_i \in \{-1, 1\}$ is the class) into camps of each class. If the data is linearly separable, we can solve for a hyperplane which maximizes the margin $\frac{1}{\|\mathbf{w}\|}$.

The points $\mathbf{x}_i$ which are closest to the decision boundary $\left(\frac{1}{\|\mathbf{w}\|} \text{ away}\right)$ are called **support vectors**.

3.4 Deep Network

A deep network learns class scoring functions directly, without explicitly computing any probabilities.

3.5 Dimensionality Reduction

Let's go back to Bayesian classification for a moment. For good results, we need a good estimate of the posterior probability $P(\omega_i \mid \mathbf{x})$, which we obtain via training samples.

- If we have 10,000 training samples $(\mathbf{x}_i \in [0,1], y_i)$, we might approximate the likelihood by histogramming our samples into 100 bins of width 0.01. Then, on average we have 100 samples per bin. If one sample is mislabeled per bin, we're losing 1% of accuracy in our likelihood model.
- In the same scenario (10,000 training samples, width 0.01 bins) except with $\mathbf{x}_i \in [0,1]^3$, there are now 100^3 bins and *0.01* samples per bin on average. Our likelihood model (and by extension our classifier) is going to be terrible! The curse of dimensionality strikes again.

One way to make classification more tractable is to *reduce the dimensionality of the feature space.*

3.5.1 Linear Projection

We project $\mathbf{x} \in \mathbb{R}^n$ to a lower-dimensional space $\mathbb{R}^m$ via matrix multiply $\mathbf{W}^T\mathbf{x}$, where $\mathbf{W} \in \mathbb{R}^{n \times m}$. To determine $\mathbf{W}$, we can use PCA, LDA, ICA...

3.5.2 PCA

We can look at PCA in a facial recognition setting. Our feature vectors are flattened images $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. We compute the mean vector and covariance matrix of our data as

$$\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

$$\Sigma = \frac{1}{n-1}\sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

If we order the eigenvectors of Σ by associated eigenvalue from greatest to least, they are the mutually orthogonal directions of greatest to least variance (spread). If we take the first k eigenvectors, those are the directions in which the data varies the most. We will do this, and call those eigenvectors the *principal components.* They'll form a k-dimensional basis for a new smaller feature space.

To compute this naively is intractable, since the covariance matrix might be enormous. For efficiency, we should take a thin SVD of the $d \times n$ data matrix

$$\begin{bmatrix} | & \cdots & | \\ (\mathbf{x}_1 - \mu) & \cdots & (\mathbf{x}_n - \mu) \\ | & \cdots & | \end{bmatrix}$$

& the singular vectors corresponding to the highest singular values will be our principal components. Finally, regardless of how they're computed, the principal components form the columns of $\mathbf{W}$.

In facial recognition, we can project the training images and test images to a lower-dimensional space using $\mathbf{W}$ (computed from training images), then use some classification method to great success.

3.5.3 LDA / Fisher's Linear Discriminant

PCA is the best linear approximation to the data in a least-squares sense. But it doesn't take the discriminative nature of our task into account. If we're trying to do recognition and not just

compression, we can project using Fisher's linear discriminant and actually *pull the classes apart* during compression. Instead of maximizing the total projected spread, Fisher's linear discriminant maximizes the ratio of between-class spread to within-class spread.

This makes discrimination easier.