Pattern Recognition and Machine Learning

James L. Crowley

ENSIMAG 3 - MMIS
Lesson 1

Fall Semester 2016
5 October 2016

Learning and Evaluation of Pattern Recognition Processes

Outline

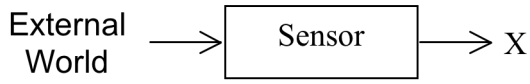
Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
K	Number of classes
C_k	The k^{th} class
$\vec{X} \in C_k$	Statement that an observation $\vec{X}$ is a member of class C_k
$\hat{C}_k$	The estimated class
$\tilde{C}_k$	The true class.
$\hat{C}_k = R(\vec{X}, \vec{w})$	A recognition function that predicts $\hat{C}_k$ from $\vec{X}$
$\{\vec{X}_m\}$ $\{y_m\}$	Training samples of $\vec{X}$ for learning, along with ground truth $\vec{y}$
M	The number of training samples.
$h(\vec{X})$	A multidimensional histogram of $\vec{X}$
Q	The number of cells in $h(\vec{X})$

1. The Pattern Recognition Problem

Pattern Recognition is the process of assigning observations to categories.

Observations are produced by some form of sensor.
A sensor is a transducer that transforms physical phenomena into digital measurements. These measurements are classically called "Features".



Features may be Boolean, natural numbers, integers, real numbers or symbolic labels. An observation is sometimes called an "entity" or "data" depending on the context and domain.

In most interesting problems, the sensor provides a feature vector, $\vec{X}$,

Composed of *D* properties or features $\vec{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$

Our problem is to build a function, called a classifier or recognizer, $R(\vec{X})$, that maps the observation, $\vec{X}$ into a statement that the observation belongs to a class $\hat{C}_k$ from a set of K possible classes. $R(\vec{X}) \rightarrow \hat{C}_k$

$$\vec{X} \rightarrow \boxed{R(\vec{X})} \rightarrow \hat{C}_k$$

In most classic techniques, the class $\hat{C}_k$ is from a set of K known classes $\{C_k\}$.

Almost all current classification techniques require the set of classes, $\{C_k\}$, to be predefined. An open research problem is how to design classification algorithms that allow $\{C_k\}$ to be an open set that grows with experience.

Discriminant and Decision Functions

The classification function $R(\vec{X})$ can typically be decomposed into two parts:

$$\hat{C}_k \leftarrow d\left(\vec{g}\left(\vec{X}\right)\right)$$

where $\vec{g}\left(\vec{X}\right)$ is a discrminant function and $d\left(\vec{g}\left(\vec{X}\right)\right)$ is a decision function.

$\vec{g}\left(\vec{X}\right)$: A discriminant function that transforms: $\vec{X} \rightarrow \mathbb{R}^K$
(A vector of real numbers)

$d\left(\vec{g}\left(\vec{X}\right)\right)$: A decision function $\mathbb{R}^K \rightarrow \hat{C}_k \in \{C_k\}$

The discriminant is typically a vector of functions, with one for each class.

$$\vec{g}(\vec{X}) = \begin{pmatrix} g_1(\vec{X}) \\ g_2(\vec{X}) \\ \vdots \\ g_K(\vec{X}) \end{pmatrix}$$

The decision function, $d\left(\vec{g}\left(\vec{X}\right)\right)$, can be an arg-max{}, a sigma function, a logistics function, or any other function that selects C_k from $\vec{X}$.
For today we will use arg-max{}.

$$\hat{C}_k = d\left(\vec{g}\left(\vec{X}\right)\right) = \arg\max_{C_k}\left\{g_k\left(\vec{X}\right)\right\}$$

In some problems, there is a notion of "cost" for errors that the cost is different for different decisions. In this case we will seek to minimize the cost of an error rather than the number of errors by biasing the classification with a notion of risk.

Generative and discriminative approaches to recognition

Let $\tilde{C}_k$ be the True class for $\vec{X}$. Then the classification $\mathrm{R}(\vec{X})$ is TRUE if $\hat{C}_k = \tilde{C}_k$ and FALSE or F if $\hat{C}_k \neq \tilde{C}_k$.

if $\hat{C}_k = \tilde{C}_k$ then T else F.

When $\hat{\omega}_k \neq \omega_k$ the classification is an Error.

The learning problem can be formulated as choosing a function $R(\vec{X})$ that minimizes the number of errors.

A Generative approach, uses estimates of probability and Bayes rule to explicitly estimate the probability of error:

Baye's Rule tells us: $P(\vec{X} \in C_k \mid \vec{X} = \vec{x}) = \dfrac{p(\vec{X} = \vec{x} \mid \vec{X} \in C_k) p(\vec{X} \in C_k)}{p(\vec{X} = \vec{x})}$

This is commonly simplified by using the symbol C_k to represent $\vec{X} \in C_k$, and $\vec{X}$ for $\vec{X} = \vec{x}$ giving:

$$P(C_k \mid \vec{X}) = \frac{p(\vec{X} \mid C_k) p(C_k)}{p(\vec{X})}$$

Because

$$p(\vec{X} \in C_k \mid \vec{X}) + p(\vec{X} \notin C_k \mid \vec{X}) = 1$$

We can minimize the number of errors by explicitly maximizing the probability $P(\vec{X} \in C_k \mid \vec{X})$

$$\hat{C}_k = \arg\max_{C_k} \left\{ p(C_k \mid \vec{X}) \right\}$$

This is called a "generative" approach, because it involves GENERATING or learning a model for the $p(\vec{X} \in C_k \mid \vec{X})$

An alternative is to choose the function $\vec{g}(\vec{X})$ that is chosen to minimize the number of errors in some representative training set without explicitly calculating $p(C_k \mid \vec{X})$, $P(C_k \mid \vec{X})$, $p(\vec{X} \mid C_k)$, $p(\vec{X} \in C_k)$, or $p(\vec{X})$

This is called a DISCRIMINANTIVE approach. We will see several discriminative approaches to recognition in this course.

Supervised vs Unsupervised Learning

Supervised Learning

Most classical methods for learning a recognition function, learn from a set of labeled training data, composed of M independent examples, $\{\vec{X}_m\}$ for which we know the true class $\{y_m\}$.

The set $\{\vec{X}_m\}$, $\{y_m\}$ is said to compose the training data.

The danger with supervised learning is that the model may not generalize to data outside the training set.

The quality of the recognizer depends on the degree to which the training data $\{\vec{X}_m\}$ represents the range of variations of real data.

Unsupervised Learning

Unsupervised Learning techniques learn the recognition function without a labeled training set. Such methods typically require a much larger sample of data for learning.

Semi-Supervised Learning.

A number of hybrid algorithms exist that initiate learning from a labeled training set and then extend the learning with unlabeled data.

In this course we will concentrate on Supervised learning techniques.

Training and Validation

In most learning algorithms, we use the training data both to estimate the recognizer and to evaluate the results. However, there is a FUNDAMENTAL RULE in machine learning:

NEVER TEST WITH THE TRAINING DATA !

A typical approach is to use cross validation (also known as rotation estimation) in learning.

Cross validation partitions the training data into N folds (or complementary subsets). A subset of the folds are used to train the classifier, and the result is tested on the other folds. A taxonomy of common techniques include:

- Exhaustive cross-validation
  - Leave p-out cross-validation
  - Leave one-out cross-validation

- Non-exhaustive cross-validation
  - k-fold cross-validation
  - 2-fold cross-validation
  - Repeated sub-sampling validation

We will discuss these techniques more as we begin programming exercises.

2. Supervised Learning with Baye's Rule

Bayes rule provides a primary tool for Generative Learning.

Baye's Rule tells us: $P(\vec{X} \in C_k \mid \vec{X} = \vec{x}) = \dfrac{p(\vec{X} = \vec{x} \mid \vec{X} \in C_k) p(\vec{X} \in C_k)}{p(\vec{X} = \vec{x})}$

This is commonly simplified as:

$$P(C_k \mid \vec{X}) = \frac{p(\vec{X} \mid C_k) p(C_k)}{p(\vec{X})}$$

Our classifier is then $\hat{C}_k = \arg\max_{C_k} \left\{ \dfrac{p(\vec{X} \mid C_k) p(C_k)}{p(\vec{X})} \right\}$

When applied directly, without any other domain knowledge, this is sometimes called a "naïve Bayesian classifier". This does not mean that the technique is "stupid" or "simplistic". Naïve is a technical term that means that the classifier does not use domain knowledge.

We can note that $p(\vec{X})$ is common to all the classes and does not change the arg-max. We can eliminate it to obtain:

$$\hat{C}_k = \arg\max_{C_k} \left\{ p(\vec{X} \mid C_k) p(C_k) \right\}$$

This is called a maximum likelihood classifier because $p(\vec{X} \mid C_k) p(C_k)$ is a likelihood but not a probability.

To apply Baye's rule, we require a representation for the probabilities $p(\vec{X} \mid C_k)$, $p(\vec{X})$, and $p(C_k)$. The term $p(C_k)$ is a number that represents the probability of encountering an observation from class C_k. For a training set of M samples of which M_k are from class k, this is simply the frequency of occurrence of class k.

$$p(C_k) = \frac{M_k}{M}$$

The terms $p(\vec{X} \mid C_k)$, $p(\vec{X})$, are more subtle.
One of the simplest approaches is to use histograms computed from the training data to represent $p(\vec{X} \mid C_k)$ and $p(\vec{X})$.

Baye's Rule as a Ratio of Histograms

Consider an example with K classes where observations are described by a single feature, X, with N possible values. (D=1)

In classic examples, X is an integer in the range [0, N-1].
For the moment, let us assume X is an integer in the range [0, N-1].
Note that with modern program languages such as Python we can use any set of N labels.

Assume that we have a "training set" of M samples, $\{\vec{X}_m\}$ with labels $\{y_m\}$.
In this example, let ,$\{y_m\}$ be the index of the class [1, K].

For each class k, we allocate a table, $h_k()$, with N cells.
To estimate the probability for X, we count the number of times each value occurs in each class:

$$\forall_{m=1}^{M} : \forall_{k=1}^{K} : \text{if } \mathrm{y_k} = \mathrm{k} : h_k(X_m) \leftarrow h_k(X_m) + 1; M_k \leftarrow M_k + 1$$

For data in the training set, when observing a member of the class C_k, the probability of observing a value X=x is

$$p(X = x \mid C_k) = \frac{1}{M_k} h_k(x)$$

For simplification we will write this as: $p(X \mid C_k) = \frac{1}{M_k} h_k(x)$

The combined probability for all classes is the sum of the histograms.

Let $h(x) = \sum_{k=1}^{K} h_k(x)$ and $M = \sum_{k=1}^{K} M_k$

Then $p(X = x) = \frac{1}{M} h(x)$

we will write this as $p(X) = \frac{1}{M} h(x)$

$p(X \in C_k)$ can be estimated from the number of members in the training set for each class. (We assume that the training set is representative of real data.

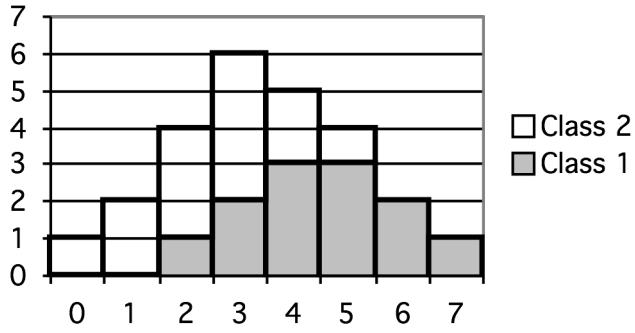
$p(X \in C_k) = \frac{M_k}{M}$. We will write this as $p(C_k) = \frac{M_k}{M}$

the probability that an observation X belongs to class C_k is

$$p(C_k \mid X) = \frac{p(X \mid C_k)p(C_k)}{p(X)} = \frac{\frac{1}{M_k}h_k(x)\frac{M_k}{M}}{\frac{1}{M}h(x)} = \frac{h_k(x)}{h(x)} = \frac{h_k(x)}{\sum_{k=1}^{K} h_k(x)}$$

Example: K=2 classes, N=8 Values

To illustrate, consider an example with 2 classes (K=2) and where X can take on 8 values (N=8, D=1).



If we obtain an observation with X=2, then $p(C_k=C_1 \mid X=2) = 1/4$

Generalization to Vectors of Features.

We can easily generalize to the case of multiple features. For example, each person in this class has a height, weight and age. We can represent these as three integers x_1, x_2 and x_3. Thus each person is represented by the "feature" vector $\vec{X} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$.

We can build up a 3-D histogram, $h(x_1, x_2, x_3)$, for the M persons in this lecture as:

$$\forall m = 1, M : h(\vec{X}_m) = h(\vec{X}_m) + 1$$

or equivalently: $\forall m=1, M : h(x_1, x_2, x_3) := h(x_1, x_2, x_3) + 1;$

and the probability of a specific vector is $P(\vec{X} = \vec{x}) = \frac{1}{M}h(\vec{x})$

When each of the D features can have N values, the total number of cells in the histogram will be $Q = N^D$

Number of samples required

Using Histograms requires two assumptions.

1) That the observing conditions do not change with time (stationary),
2) That the training set is large enough to include all possible values.

How many training samples are needed?

A classic rule of thumb is that there are 10 samples for each histogram cell.

When each feature has N values, with D features, the histogram will have

$Q = N^D$ cells. This is sometimes called the histogram "capacity".

10 samples per cell states that $M > 10\ Q = 10\ N^D$

However, it is often easier to work with powers of two.
I recommend using $2^3 = 8$ samples per cell.

Then the rule of thumb is $M \geq 8Q = 8\ N^D = N^{D+3}$

Explanation: In the worst case Root Mean Square error is proportional to $O(\frac{Q}{M})$.

This can be estimated by comparing the observed histograms to an ideal parametric model of the probability density or by comparing histograms of subsets samples to histograms from a very large sample. Let *p(x)* be a probability density function. The RMS (root-mean-square) sampling error between a histogram and the density function is

$$E_{RMS} = \sqrt{E\left\{\left(h(x) - p(x)\right)^2\right\}} \approx O(\frac{Q}{M})$$

The worst case occurs for a uniform probability density function.

For most applications, $M \geq 8\ Q$ (8 samples per "cell") is reasonable
(less than 12% RMS error).
We also assumed that the feature values were integer numbers in the range [0, N-1].
This can be easily obtained from any features.

Symbolic Features

If the features are symbolic, *h(x)* is addressed using a hash table, and the feature and feature values act as a hash key. As before *h(x)* counts the number of examples of each symbol. When symbolic x has N possible symbols then

$$p(X = x) = \frac{1}{M} h(x) \quad \text{as before}$$

"Bag of Features" methods are increasingly used for learning and recognition.
The only difference is that there is no "order" relation between the feature values.

Unbounded and real-valued features

If X is real-valued and unbounded, we can bind it to a finite interval and quantize it. We can quantize with a function such as “trunc()” or “round()”. The function trunc() removes the fractional part of a number. Round() adds ½ then removes the factional part.

To quantize a real X to N discrete values : [1, N]

/* first bound x to a finite range */

If ($x < x_{min}$) then $x := x_{min}$;
If ($x > x_{max}$) then $x := x_{max}$;

$$n = round\left((N-1) \cdot \frac{x - x_{\min}}{x_{\max} - x_{\min}} \right) + 1$$

Alternatively, we can present $p(\vec{X})$, $p(\vec{X} \mid C_k)$ as Probability Density Functions.