Statistical aspects of supervised learning

Luís B. Almeida

January 16, 2015

1 Introduction

This document is intended as a support text for students of the Machine Learning course at Instituto Superior Técnico, Lisbon, Portugal. It deals with some of the statistical aspects of supervised systems that are trained through the minimization of a cost function that depends on the system's outputs and desired outputs. The text starts with a brief study of the Bayes decision rule, then proceeds with the study of Naive Bayes classifiers, and then goes through the estimation of conditional means, posterior probabilities of classes, conditional quantiles and conditional prediction intervals.

2 The Bayes Classifier

Assume that we have a classification problem in which the joint probability distribution of the data and the classes is known. More specifically, assume that we have patterns $\mathbf{x}$ which belong, with certain probabilities, to one of n classes, $C_1, ..., C_n$, and that the joint density $p(C, \mathbf{x})$ is known.[1] From this distribution, we can obtain the probability that some observed data $\mathbf{x}$ belongs to class C, a probability given by $P(C|\mathbf{x})$. This is normally called the *posterior* class probability, because it is the probability that we assign to the class after observing $\mathbf{x}$ (the *prior* probability is $P(C)$, which we assign to the class before observing $\mathbf{x}$). Given a pattern $\mathbf{x}$ to be classified, it is natural to think of assigning it to the class to which it has the highest probability of belonging, i.e., to the class C_i for which $P(C_i|\mathbf{x})$ is highest. This corresponds to the so-called *Bayes classifier*,also called the *Maximum a Posteriori classifier*, often abbreviated as *MAP classifier*. Formally, the Bayes classifier assigns to each pattern $\mathbf{x}$ a class[2]

$$f(\mathbf{x}) \in \arg\max_C P(C|\mathbf{x}). \tag{1}$$

Using the Bayes rule, we can write

$$P(C|\mathbf{x}) = \frac{p(\mathbf{x}|C)P(C)}{p(\mathbf{x})}. \tag{2}$$

[1] For the following discussion, it is useful to assume an exact knowledge of $p(C, \mathbf{x})$. However, it is rather uncommon, in practical situations, to know that distribution. Sometimes the distribution can be estimated from observed data or obtained from a theoretical model, but normally it is not known exactly.

[2] Some students may not be familiar with the notation that is used in this and in some following equations. The notation $\arg\max_C g(C)$ demotes the set of values of C for which the function $g(C)$ attains it absolute minimum value. Often, in the cases studied in this text, that set contains just a single value of C.

Since the denominator in the right hand side of this equation doesn't depend on C, we can write the Bayes classifier as

$$f(\mathbf{x}) \in \arg\max_C p(\mathbf{x}|C)P(C). \tag{3}$$

The Bayes classifier is often applied in this form, because it often easier to compute $p(\mathbf{x}|C)$ and $P(C)$ than to directly compute $P(C|\mathbf{x})$.

The Bayes classifier is an important concept, because it represents the classifier with the lowest possible expected error rate. In practical situations, the probabilities involved in Eqs. (1) or (3) are normally unknown, and must be estimated from the data. With that, one obtains only an approximation to the exact Bayes classifier.

3 The Bayes Decision Rule

The Bayes classifier is the one which yields the minimum expected error rate. However, in many practical situations, different classification errors may have different costs. When costs of errors are taken into account, it is often useful to speak in terms of actions to be taken, instead of classifications. Let's consider an example. Assume that the problem that we are dealing with is deciding whether to treat a patient, based on a set $\mathbf{x}$ of results of medical exams. Class C_1 corresponds to the patient having a disease that is fatal if not treated, and class C_2 to not having that disease. Assume, furthermore, that after performing exams on a certain patient, we determine that she has a 40% probability of having the disease, i.e., $P(C_1|\mathbf{x}) = 0.4$. Although this probability is lower than the probability of not having the disease, one would, most probably, decide to treat the patient. There are, in this problem, two kinds of possible errors: *false alarm*, which corresponds to classifying the patient as having the disease when she has none, and *missed detection*, which corresponds to classifying her as not having the disease when she does have it. The important issue, here, is that we assign different costs to these two kinds of errors: it is much more serious not to treat a patient who has the disease than to treat a patient who has none. Due to this, it is reasonable to assign the patient to the "disease" class, and therefore treat her, even if the probability of belonging to that class is somewhat below 50%.

To formalize this kind of situation, let us designate by $L(C_i, C_j)$ the cost of assigning the pattern to class C_j when it actually belongs to class C_i. Often we have $L(C_i, C_i) = 0$ (zero cost for correct classifications), although that may not always be the case. Denote by $f(\mathbf{x})$ a classifier, i.e., a function which assigns a class to each pattern. The classifier's expected cost is

$$L = \int \sum_i L\left[C_i, f(\mathbf{x})\right] p(C_i, \mathbf{x})\, \mathrm{d}\mathbf{x}. \tag{4}$$

The contribution of pattern $\mathbf{x}$ to the expected cost is proportional to

$$\sum_i L\left[C_i, f(\mathbf{x})\right] p(C_i, \mathbf{x}) \tag{5}$$

To minimize the expected cost we should choose a classifier

$$f(\mathbf{x}) \in \arg\min_C \sum_i L(C_i, C) p(C_i, \mathbf{x}), \tag{6}$$

which is the same as

$$\boxed{f(\mathbf{x}) = \arg\min_{C} \sum_{i} L(C_i, C) P(C_i|\mathbf{x})} \tag{7}$$

because $p(C_i, \mathbf{x}) = P(C_i|\mathbf{x})\, p(\mathbf{x})$, and $p(\mathbf{x})$ doesn't depend on C.

This is called the *Bayes decision rule*. Note that, if the costs of correct classifications are zero and the costs of all errors are equal, this reduces to the Bayes classifier. The Bayes decision rule is more general than the Bayes classifier, however.

A rather common situation is one in which there are only two classes and the cost of correct classifications is zero. In this case Eq. (7) indicates that we should classify $\mathbf{x}$ into class C_1 if

$$L(C_2, C_1) P(C_2|\mathbf{x}) < L(C_1, C_2) P(C_1|\mathbf{x}), \tag{8}$$

or, equivalently, if

$$P(C_1|\mathbf{x}) > \frac{L(C_2, C_1)}{L(C_1, C_2) + L(C_2, C_1)}, \tag{9}$$

which may be quite different from 50%.

In the above example of the disease diagnostic (and simplifying quite a bit), we can call t the cost of treatment and l the "cost" of the patient's life. Assuming that a patient with the disease who is not treated will certainly die and that one who is treated will certainly get cured, we can say that $L(C_1, C_1) = t$, $L(C_1, C_2) = l$, $L(C_2, C_1) = t$ and $L(C_2, C_2) = 0$. Applying Eq. (7) we obtain, for classifying $\mathbf{x}$ into C_1 (i.e., deciding for treatment), the condition

$$P(C_1|\mathbf{x}) > \frac{t}{l}. \tag{10}$$

If $l >> t$, as will normally be the case, then the decision to give the treatment will be taken even when the posterior probability of having the disease, $P(C_1|\mathbf{x})$, is much below 50%.

As we said above, usually the posterior probabilities $P(C|\mathbf{x})$ are not known exactly, and, as such, the Bayes decision rule is basically a theoretical concept. In practice, the probabilities involved in its application normally have to be estimated from the data, and therefore one obtains only an approximation to the exact Bayes decision rule.

4 Naive Bayes Classifiers

In many situations, it makes sense to consider that the input data are represented by a set of so-called *features*, which describe different aspects of the data. For example, in the processing of written documents, each document is often represented just by the set of words that occur in it. A document containing only the sentence "John is tall and Paul is short" would be represented by the set {John, is, tall, and, Paul, short}. In this case we can consider that there is a feature corresponding to each possible word, and that each feature is binary, indicating whether the corresponding word occurs in the document or not. This representation is called *bag of words*, because it indicates which words occur in the document, but not the order in which they occur nor the number of times that each word occurs. A slightly more elaborate representation would use one numerical feature per word, indicating how many times that word occurs in the document.

Let us designate the features by x_i, which are the components of the input pattern $\mathbf{x}$. In the naive Bayes classifier we assume that, given the data's class, each of the features is independent from all the other ones. Mathematically, this is expressed by

$$P(\mathbf{x}|C) = \prod_i P(x_i|C).^3 \tag{11}$$

The classifier is then expressed by

$$f(\mathbf{x}) \in \arg\max_C \prod_i P(\mathbf{x}_i|C)P(C). \tag{12}$$

The assumption of mutual independence of the features given the class, is far from being obeyed, in most practical situations. Nevertheless, it often leads to good classifiers. Further ahead, we'll discuss why this is so, and in which kinds of situations one may expect naive Bayes classifiers to be useful. Naive Bayes classifiers have the advantages of being relatively simple and of needing much fewer training data than if we were considering the features to be mutually dependent.

We'll give an example of the development of a naive Bayes classifier for the recognition of the language in which a text is written. The input to the recognizer will consist of some written text. This text is, however, not directly input to the classifier. First we'll represent it by a set of features. Each feature consists of a sequence of three consecutive characters from the text (called a *trigram*). For example, the text 'I go now' would be represented by the set of trigrams {'i g','go ','o o','o n',' no','now'}. Spaces and punctuation marks are kept. If a trigram occurs multiple times, it is counted as many times as it occurs. For example, in the sentence 'there were two cars', the trigram 'ere' is counted twice.

We'll assume that the prior probabilities of all languages are equal. Therefore, in Eq. (12), we can drop the term $P(C)$, since it does not affect the comparison between the languages. To classify the sentence "I go now", we need to compute, for each language C, the value of

$$\widehat{P}(\text{'i g'}|C)P(\text{' go'}|C)P(\text{'go '}|C)P(\text{'o n'}|C)P(\text{' no'}|C)P(\text{'now'}|C), \tag{13}$$

where and select the language that yields the highest value. The value of this function, for each language, will be called the *score* of that language.

In practice, we normally don't know the exact probabilities $P(x_i|c)$ We have to use estimates, that we'll denote by $\widehat{P}(x_i|C)$. Before using the classifier, we need to obtain these estimates for all trigrams x_i in all languages C that we wish to recognize. For this, we take a large body of text written in each language, and count the number of times that each trigram occurs in that text. That number divided by the total number of trigrams in that body of text gives an estimate of the probability that that trigram occurs in that language. Since it is easy to obtain large bodies of text in most common languages, and since the counting of trigrams is a computationally light procedure, it is relatively easy to obtain good estimates of the probabilities of occurrence of trigrams in most common languages. The bodies of text that are processed in this way constitute the training data of the classifier.

Before giving an example of results obtained with this kind of classifier, we must still address an important detail. As said above, to train the classifier, we compute the estimated probabilities

[3] We have assumed that the attributes are discrete. If they were continuous, probabilities would be replaced by probability densities.

	'i g'	' go'	'go '	'o n'	' no'	'now'	log score
Portuguese	-10.4	-7.9	-7.7	-7.0	-6.1	-10.3	-49.4
Spanish	-10.9	-8.1	-7.6	-8.1	-6.6	-10.9	-52.1
French	-10.5	-8.4	-9.3	-10.6	-6.5	-10.9	-56.2
English	-9.1	-7.0	-8.5	-9.0	-6.4	-6.9	-47.0

Table 1: Log estimated probabilities of the trigrams of the sentence "I go now" in the various languages, and log scores of the various languages for that sentence.

of occurrence of the various trigrams in each language to be recognized. Normally, some of the trigrams that occur very infrequently in the language will not occur in the body of text that we process to estimate occurrence probabilities for that language. This means that the estimates of occurrence probabilities for those trigrams in those languages will be zero. If one of those trigrams occurs in the text to be classified, the estimated score for that language will be zero, even if all the other trigrams are strongly indicative that that is the correct language. This is a significant shortcoming of the language recognizer as described so far. In order to circumvent it, we must assign small, but nonzero probabilities to the trigrams that do not occur in the training data for each language. This operation is called *smoothing* of the estimated probabilities. A commonly used form of smoothing, called *Laplacian smoothing*, consists of adding 1 to the counts of all trigrams, and making the corresponding adjustment in the total number of trigrams, for each language.

As an example of the development of a naive Bayes language recognizer, we have constructed a classifier for Portuguese, Spanish, French and English as described above. The training texts for the first three language were formed by parts of the contents of Wikipedia in those languages (the contents of Wikipedia are publicly available for download). For English, the Open American National Corpus was used.

As a first example of the results obtained with the classifier, we'll use the sentence "I go now", given above. Table 1 shows the natural logarithms of the estimated probabilities of occurrence of the various trigrams in each language, as well as the the natural logarithm of the score of each language (logarithms are used because, even for a short sentence like the one of this example, the products of estimated probabilities quickly get very small). The English language gets the highest score for this sentence, although some trigrams are more frequent in other languages than in English.

For short pieces of text, the classifier sometimes makes wrong classifications. For example, the Portuguese sentence "eu vou à praia" (meaning "I go to the beach") is classified as French, because many of the trigrams that occur in it happen to be quite frequent in French. The log scores for the various languages, for this sentence, are -101.2 for French, -104.3 for Portuguese, -131.5 for Spanish and -136.8 for English.

As the pieces of text get longer, the chance that a large percentage of the trigrams be frequent in a language different from the correct one becomes progressively smaller, and language classification errors become very infrequent. For example, the Portuguese sentence "eu vou à praia e tu ficas em casa" (meaning "I go to the beach and you stay at home") is correctly classified, yielding the log scores -240.2 for Portuguese, -261.5 for French, -278.6 for Spanish and -299.2 for English.

The difference in log scores between the highest-scored language and the second highest-scored one (which we'll call *log score margin*) gives a good indication of how trustworthy the

system's classification is. The log score margins are 2.4 for "I go now", 3.1 for "eu vou à praia", and 21.3 for "eu vou à praia e tu ficas em casa". Notice how the latter sentence yields a much larger log score margin than the other ones, reflecting the fact that its classification is much more trustworthy than those of the shorter sentences.

Some more examples of classifications made by the system are: "hoy es lunes" ("today is Monday" in Spanish), classified as Spanish with a log score margin of 2.6; "maintenant je vais manger mon déjeuner" ("now I'm going to eat my lunch" in French), classified as French with a log score margin of 35.4; "I'm reading a very interesting book on the American economy", classified as English with a log score margin of 52.0. Notice how the score margins get larger as the sentences get longer.

As we said above, the basic assumption of naive Bayes classifiers (that the probabilities of the features, given the classes, are mutually independent) is normally far from being obeyed in practical situations. For example, in our language classifier, if a trigram such as "abe" occurs, the probability that a trigram starting with "be" will also occur is much increased. Since the mentioned assumption is normally far from being obeyed, the probabilities estimated through (11) will normally be quite incorrect. One may ask, therefore, how can the naive Bayes classifier perform well, given that it uses incorrect probability estimates.

An important point to note here is that, for correct classification, we don't need correct probability estimates: we only need the probability estimate of the correct class to be higher than those of the other classes. For example, in the language classifier, we only need the probability estimate of the correct language to be larger than those of the other languages, even if all probability estimates are far from the true probability values. The example of the language classifier that we gave above may convince the reader that, as the piece of text to be classified gets longer, the ratios between the various estimated probabilities quickly get very large, the probability estimate of the correct language quickly becoming much larger than those of the other languages. For example, for the longest sentence in our example ("I'm reading a very interesting book on the American economy"), the ratio between the probability estimate of the correct language and the probability estimate of the language with the second highest score is $\exp(52.0) \approx 3.8 \times 10^{22}$, and for the second longest sentence ("maintenant je vais manger mon déjeuner"), it is $\exp(35.4) \approx 2.3 \times 10^{15}$.

Another important point to note is that the complexity and the training requirements of a classifier that took into account joint probabilities of features would normally be much higher than those of a naive Bayes classifier. We can illustrate this with the language classifier. Taking into account just letters and blank spaces, there are $27^3 \approx 2 \times 10^5$ different trigrams. For pairs of consecutive trigrams, the number is 27 times larger. If we wanted to build a classifier based on joint probabilities of pairs of consecutive trigrams, we would need approximately 27 times more storage and 27 times more training data than those we have used for the naive Bayes classifier. Each increase by 1 of the number of consecutive trigrams would bring in another factor of 27. It's easy to see that increasing the number of consecutive trigrams on which the classifier was based would quickly become impracticable.

Would a language classifier based on relatively correct probability estimates, which took into account the joint distributions of the trigrams given the classes, perform better than the naive Bayes one? Yes, it would, but the difference would be noticeable only for short sentences, because for longer pieces of text the naive Bayes classifier makes almost no errors, anyway. Since language recognizers normally are used for relatively long pieces of text, containing several sentences each, naive Bayes classifiers are quite adequate for the task, and are much simple than classifiers that

take into account joint probabilities of the features.

The foregoing considerations show that naive Bayes classifiers are good candidates to consider when the input data can be represented by a relatively large number of features, and each feature, by itself, carries some (possibly weak) information on the class to which the data belongs. Another good example of very successful use of naive Bayes classifiers are spam detectors for e-mail messages. In those detectors, words, instead of trigrams, are used as features. There are many words that are somewhat indicative of spam, such as "lottery", "casino" or "deal". Since most e-mail messages contain a relatively large number of words, the situation is similar to the one of our language recognizer, and naive Bayes classifiers generally perform quite well in this task. They also perform quite well in many other text classification tasks, besides language recognition and spam detection.

In the language recognizer example given above, we illustrated how to develop a naive Bayes classifier using discrete features. In such cases, the probability estimates $\widehat{P}(x_i|c)$ are normally obtained by a process that amounts, essentially, to counting occurrences in the training data. If the features are continuous instead of discrete, one has to find estimates $\hat{p}(x_i|C)$ of their probability densities. This is normally done by modeling those densities with a parametric model. A common approach is to model each density with a Gaussian, whose mean and variance are estimated from the training data.

5 Estimation of Statistical Quantities

In this Section we shall discuss how some statistical quantities that are of interest in many situations can be estimated by means of learning systems. We shall do this by finding what the output of a system converges to, in an ideal situation, when trained with a given cost function. We'll consider a system as depicted in Fig. 1, with input pattern $\mathbf{x}$, a single output o and a vector of parameters $\mathbf{w}$. Of course, we have $o = o(\mathbf{x}, \mathbf{w})$. We'll denote this function, at the end of the training (i.e., when $\mathbf{w}$ is the set of parameters obtained through minimization of the cost function), as $o^*(\mathbf{x})$. We'll only consider a system with a single output, because our reasoning can then be applied, individually, to each of the outputs of a multi-output system.

Figure 1: System under study.

In what follows, we shall assume an idealized situation. More specifically, we shall assume that:

A1 The system under consideration is flexible enough to be able to yield any function $o = o^*(\mathbf{x})$ at the end of training. An example of a system that doesn't obey this condition is a linear system, which is only able to yield outputs that are linear functions of its inputs. We assume that the system under consideration has no such limitations.

A2 There is a joint probability distribution of input patterns and corresponding desired output values, whose density we denote by $p(\mathbf{x}, d)$, where d is the desired value.[4]

[4]We use a notation corresponding to continuous random variables, and therefore $p()$ is a probability density.

A3 There is a cost $C(o, d)$ assigned to each pair of values of output and desired output. The cost function is the expected value of that cost relative to $p(\mathbf{x}, d)$, i.e., it is of the form

$$\mathcal{E}(\mathbf{w}) = \mathcal{E}\{C\left[o(\mathbf{x}, \mathbf{w}), d\right]\}, \tag{14}$$

where $\mathcal{E}(\cdot)$ denotes expectation relative to $p(\mathbf{x}, d)$.

A4 The training proceeds all the way to the global minimum of the cost function, i.e., it finds the output function $o^*(\mathbf{x})$ that yields the lowest possible value of the cost function.

These assumptions correspond to a rather idealized situation, which is useful for obtaining the results that follow. Along with the derivations of those results, we'll also discuss how they apply to more realistic situations.

We shall use a three step approach. First, we'll consider the rather simple case of a system with a single input pattern. Then we'll extend this result to systems with a discrete training set. Finally, we'll extend it to systems with a continuous training set. We'll go through the whole three steps in the following Section, which is concerned with systems trained with the squared error as cost function. We'll then see that the result of the first step (a system with a single input pattern) immediately allows us to see what the conclusion would be for the other two kinds of systems. Consequently, in the subsequent Section, we'll only perform the first step, and we'll then enunciate, without proof, what is the result for the other kinds of systems.

5.1 Estimation of Conditional Means

Step 1: Single Input Pattern

As explained above, in the first step we'll consider a system with a single input pattern $\mathbf{x}$. Since the system's input is always the same, its output is only a function of the parameter vector. At the end of the training, the output is simply a constant, o^*. In this Section we'll assume that the cost function is the squared error, i.e., referring to Eq. (14), $C(o, d) = (o - d)^2$. Since $\mathbf{x}$ is fixed, we have, in this case, $p(\mathbf{x}, d) = p(d)$. The cost function is then given by

$$E = \int (o - d)^2 p(d)\, \mathrm{d}d.^{5} \tag{15}$$

We find the minimum of the cost function by differentiating and equating to zero:

$$\begin{aligned} \frac{\partial E}{\partial o} &= 2 \int (o - d) p(d)\, \mathrm{d}d & (16) \\ &= 2 \int o\, p(d)\, \mathrm{d}d - \int d\, p(d)\, \mathrm{d}d & (17) \\ &= 2\left[o - \mathcal{E}(d)\right]. & (18) \end{aligned}$$

where the expectation is relative to $p(d)$. Equating to zero, we obtain the result

$$\boxed{o^* = \mathcal{E}(d).} \tag{19}$$

In the case of discrete variables we would have to use probabilities, but the conclusions would be the same, as can be easily verified.

[5] Note the difference between the non-italic character 'd', which denotes a differential, and the italic character 'd', which denotes a variable that corresponds to the desired value. We don't indicate the integration limits. Each integral extends over the domain of the corresponding variable.

Therefore, this system computes the expected value of the desired output, $\mathcal{E}(d)$.

We now pause a little to discuss this result. In an actual application, in which we train a system using a finite training set, we normally use, as probability distribution, the so-called *empirical distribution*, which is the distribution obtained by assigning equal probabilities to all elements of the training set (note that, in the training set, the input pattern – which, we recall, is always the same in this case – may appear more than once, possibly associated with different desired values). After our idealized training, in which we assume that we reach the global minimum of the cost function, the system's output will be $o^* = \mathcal{E}(d)$, as we have seen. This is just the mean of the desired values that appear in the various entries of the training set. One may ask what is the interest of finding that mean value in this way, when it could have been found by simply averaging those desired values directly from the training set. The importance of this result is not in the actual computation of that mean, but in the use that we shall make of the result in the steps that follow.

Step 2: Discrete Input Set

We now consider a system whose input patterns belong to a discrete set, $\mathbf{x} \in \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^K\}$, which may be finite or infinite (in the latter case we consider $K = \infty$). The cost function is now given by

$$E = \sum_{k=1}^{K} \int \big[o(\mathbf{x}^k) - d\big]^2 p(\mathbf{x}^k, d)\, \mathrm{d}d \tag{20}$$

$$= \sum_{k=1}^{K} p(\mathbf{x}^k) \int \big[o(\mathbf{x}^k) - d\big]^2 p(d|\mathbf{x}^k)\, \mathrm{d}d. \tag{21}$$

Given our assumption that the system being trained can yield any function $o(\mathbf{x})$, we see that the values $o(\mathbf{x}^k)$, for different values of k, can be chosen independently from one another. Therefore, for minimizing E in Eq. (20), we can separately minimize each of the terms of the sum in k. For all k for which $p(\mathbf{x}^k) \neq 0$, the minimization is similar to the one performed for the single input case above, and leads to

$$o^*(\mathbf{x}^k) = \int d\, p(d|\mathbf{x}^k)\, \mathrm{d}d, \tag{22}$$

which is the conditional mean

$$\boxed{o^*(\mathbf{x}^k) = \mathcal{E}(d|\mathbf{x}^k).} \tag{23}$$

This is a generalization of the result that we have obtained in the single input case: The system's output is the mean of the desired values given the input pattern. We illustrate this in Fig. 2, using as example a situation in which each input pattern consists of a single real value, for easy visualization.

Once again, given some training set (which will always be discrete, in practice), we could find $\mathcal{E}(d|\mathbf{x}^k)$ by simply computing the mean of all the desired values that occur associated with each training pattern $\mathbf{x}^k$. In a typical practical situation, however, we are not directly interested in optimizing the system's performance in the training set. Instead, we are interested in optimizing the system's performance in the input universe, although, normally, that can only be achieved

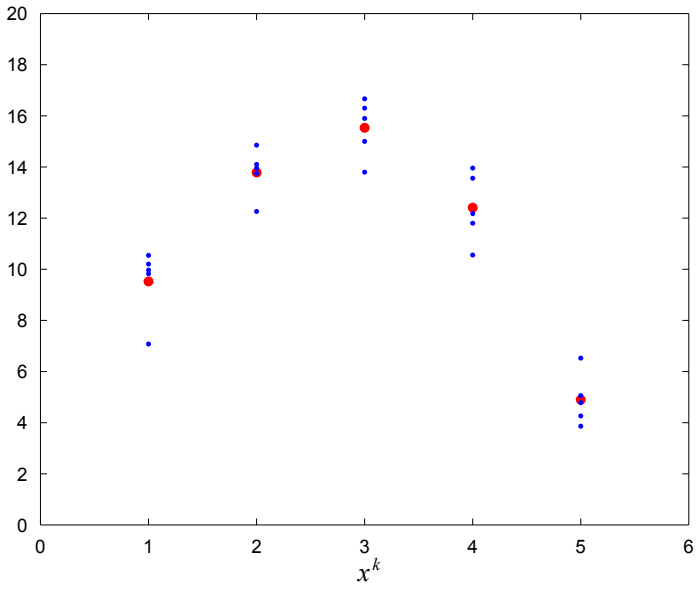

Figure 2: Example of estimation of conditional means on a discrete set. The blue circles are desired values and the red circles are the conditional means.

in an approximate way, and only by indirect means, such as through the use of some form of regularization.

If we are successful in training a system such that it performs adequately in the training set and that this performance generalizes to the whole input universe, then the results obtained above allow us to say that, in the input universe, the system's output $o(\mathbf{x}^k)$ is an estimator of the conditional mean $\mathcal{E}(d|\mathbf{x}^k)$. This is where the result (23) becomes relevant. We write this as

$$\boxed{o(\mathbf{x}^k) \approx \mathcal{E}(d|\mathbf{x}^k).} \tag{24}$$

In reality there are other reasons, besides the one presented above, for the fact that, in practical situations, $o(\mathbf{x}^k)$ is only an estimator (or, equivalently, an approximation) of $\mathcal{E}(d|\mathbf{x}^k)$. One of them is that, since we are interested in the system's generalization ability, we normally use systems that perform some form of smoothing relative to the optimal solution $o^*(\mathbf{x}^k)$. This may be achieved by means of the use of restricted-size systems or of some explicit form of regularization, for example. Another reason for $o(\mathbf{x}^k)$ to be only an approximation of $\mathcal{E}(d|\mathbf{x}^k)$ is that the optimization methods that we use for training our systems normally only have asymptotic convergence, and therefore we never train the systems all the way to the optimum, although we may stop close to it.

Step 3: Continuous Input Set

In this case the input patterns belong to some continuous subset of $\mathbb{R}^n$, where n is the dimensionality of the input patterns. The cost function is given by

$$E = \iint \big[o(\mathbf{x}) - d\big]^2 p(\mathbf{x}, d)\, \mathrm{d}d\, \mathrm{d}\mathbf{x} \tag{25}$$

$$= \int p(\mathbf{x}) \int \big[o(\mathbf{x}) - d\big]^2 p(d|\mathbf{x})\, \mathrm{d}d\, \mathrm{d}\mathbf{x}. \tag{26}$$

The minimization is performed in a similar way to the one used in Step 2: since we have absolute liberty to independently choose the value of $o(\mathbf{x})$ for every value of $\mathbf{x}$, we can minimize the cost function by simply minimizing the integrand of the inner integral for every $\mathbf{x}$ for which $p(\mathbf{x}) \neq 0$. Similarly to what happened in Step 2, this leads to

$$o^*(\mathbf{x}) = \int d\, p(d|\mathbf{x})\, \mathrm{d}d, \tag{27}$$

or

$$\boxed{o^*(\mathbf{x}) = \mathcal{E}(d|\mathbf{x}).} \tag{28}$$

Once again, the optimal output is the mean of the desired values given the input pattern. And once again, this result can't be directly applied to practical situations, because in those situations the training set is always discrete, and therefore a result for a continuous input set doesn't directly apply. In fact, with a continuous distribution of the input patterns, and assuming that the training data were randomly chosen according to that distribution, the probability of the same input pattern appearing more than once in the training set is zero. Therefore, assuming that no repeated input patterns occurred in the training set, we would simply have $o^*(\mathbf{x}^k) = d^k$, where d^k designates the desired value that corresponded to the pattern $\mathbf{x}^k$ from the training set. We may, however, make a comment similar to the one that we made regarding discrete input spaces: In practice we normally are not interested in optimizing the system in the training set, but are instead interested in its capacity of generalizing to the whole input space. In that case we have to use some explicit or implicit form of regularization, with performs some form of smoothing of $o^*(\mathbf{x})$, and we can say, once again, that

$$\boxed{o(\mathbf{x}) \approx \mathcal{E}(d|\mathbf{x}).} \tag{29}$$

Figure 3 illustrates this result. The training data consisted of 400 training pairs (x,d) obtained through the equation

$$d = -\cos(\pi x) + n, \tag{30}$$

where x was randomly chosen, uniformly distributed in $[-1, 1]$, and n was Gaussian noise with zero mean, and with $\sigma = 0.2$ for $x \in [-1, 0)$ and $\sigma = 0.5$ for $x \in [0, 1]$. The system being trained consisted of a multilayer perceptron (MLP) with a single input, a hidden layer with two sigmoidal units and a single linear output unit. The training was taken until very close to the optimum, and we can see that the system's output closely approximates the conditional mean, which is the function $-\cos(\pi x)$. In this case, the regularization was implicitly imposed by the very small number of hidden units of the MLP, which didn't allow $o(x)$ to oscillate much.

In Fig. 4 we show an example of inappropriate regularization. The difference relative to the previous case was that the network now had 200 hidden units, which allowed numerous oscillations in $o(x)$. A network of this size would need a much larger number of training patterns to be adequately trained.

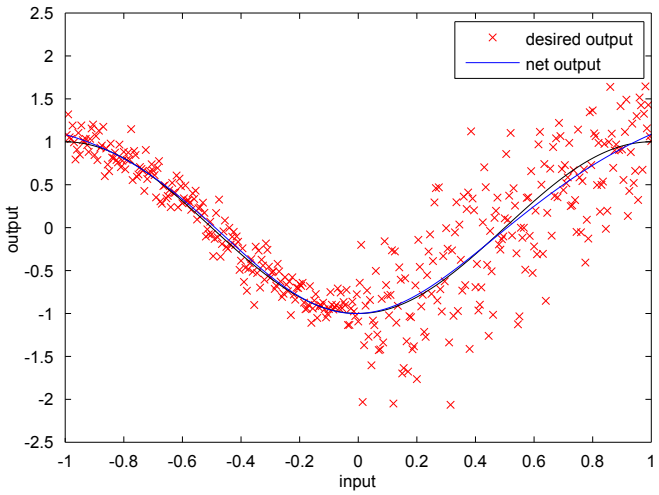

Figure 3: Estimation of the conditional mean of a sampled continuous distribution by means of a multilayer perceptron with two hidden units. The estimated mean is shown in blue and the true mean is shown in black.

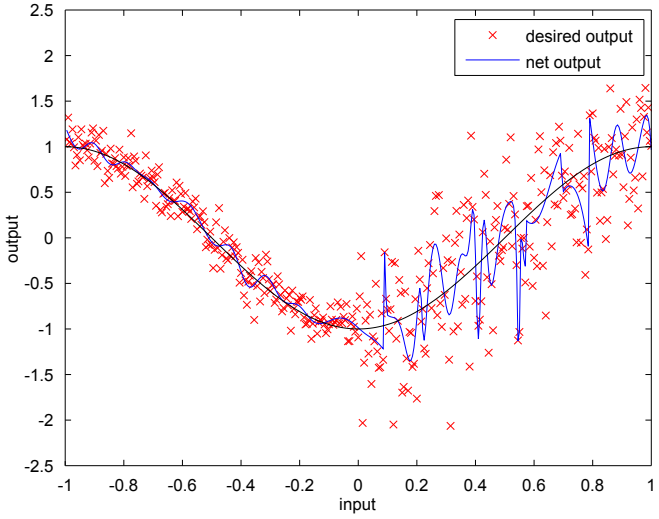

Figure 4: Estimation of the conditional mean of a sampled continuous distribution by means of a multilayer perceptron with 200 hidden units, showing overfitting. The estimated mean is shown in blue and the true mean is shown in black.

5.2 Estimation of Posterior Class Probabilities in Classification Problems

We'll now see that, in the case of classification problems, the results obtained above allow the estimation of the posterior probabilities of classes. We'll start by considering a binary classifica-

tion problem, where we designate the two classes by C_0 and C_1, respectively, and we assign the desired values $d = 0$ and $d = 1$ to the patterns of those two classes, respectively. The training uses the squared error as cost function. As seen above, we have $o^*(\mathbf{x}) = \mathcal{E}(d|\mathbf{x})$. Expanding,

$$\begin{aligned} o^*(\mathbf{x}) &= \mathcal{E}(d|\mathbf{x}) \\ &= 0\, P(C_0|\mathbf{x}) + 1\, P(C_1|\mathbf{x}) \\ &= P(C_1|\mathbf{x}). \end{aligned} \tag{31}$$

We recall that this is the result that would be obtained in an ideal situation. In a real-life situation, we will have

$$\boxed{o(\mathbf{x}) \approx P(C_1|\mathbf{x}).} \tag{32}$$

Therefore, a system trained with the quadratic error as cost function can estimate posterior class probabilities. If the classification is not binary, we just have to use a system with one output for each class, assigning a desired value of 1 to the output that corresponds to the correct class and desired values of 0 to the other outputs. In this way, every output acts as a classifier of the corresponding class against all other classes, and therefore estimates the posterior probability of the corresponding class.

The squared error is not the only cost function for which results (31), (32) stay true. In fact there is an infinite number of cost functions for which these results are true. A commonly used one is the so-called cross-entropy cost function,

$$\begin{aligned} C(\mathbf{x}, d) &= (1-d)\log(1-o) + d\log o \\ &= \begin{cases} -\log(1-o) & \text{if } d = 0 \\ -\log o & \text{if } d = 1. \end{cases} \end{aligned}$$

for $o \in [0, 1]$.

Figure 5 shows an example of the estimation of the posterior probability of a class by means of a multilayer perceptron. The two Gaussians are the densities $P(C_i|x)$ for $i = 0, 1$. The blue and red curves show the estimated posteriors $P(C_0|x)$ and $P(C_1|x)$, respectively.

Given that the optimal classification, given by the Bayes decision rule (7), is expressed in term of the posterior class probabilities, one might think that the best way to create a classifier from training data would be to estimate the posterior probabilities through the method that we've just described, and then to use the Bayes decision rule. This is not the best way to create a classifier, however. The reason for that may be expressed, intuitively, by noting that such a classifier would "waste" many of its resources in estimating $P(C_i|x)$ far from the classification boundary, where an accurate estimation is unimportant. A system trained directly with the costs $L(C_i, C_j)$ (see Section 3) will concentrate all its resources in the decision boundary, which is all that matters in a classifier.[6]

5.3 Estimation of Quantiles and Prediction Intervals

In regression problems, d normally represents some real-world variable that we want our system to estimate. The estimate $\mathcal{E}(d|\mathbf{x})$ that we obtained above, which indicates the value that we

[6] These costs form a segmentally constant, discontinuous function, which cannot be optimized by means of gradient based methods. Normally one replaces that function with a continuous, differentiable approximation.

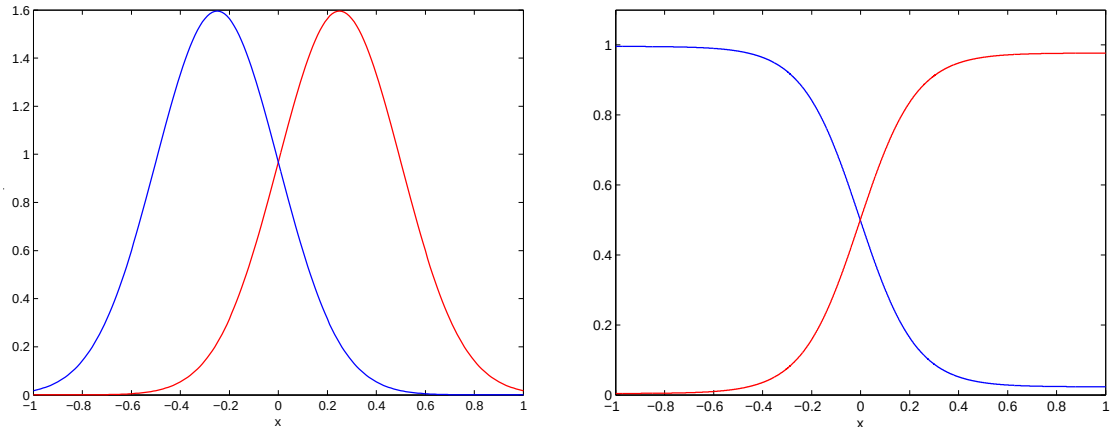

Figure 5: Estimation of the posterior probabilities of two classes as a function of an input x. Left: Conditional distributions of the data given the classes. Right: Posterior distributions of the classes estimated by an MLP with two hidden units with a training set of 2000 points.

expect to observe on average, is useful in many cases, but there are numerous situations in which it may not provide the information that we want. Imagine, for example, that we are estimating a pressure that, we know, cannot go above a certain value (say, 10 atmospheres) to guarantee the security of a system. In that case, estimating the pressure to be, on average, 8.2 atmospheres is not a very useful information, because it doesn't tell us up to what value the pressure might be with significant probability. It would be much more useful if we could say, for example, that the pressure was estimated to be below 9.5 atmospheres with 99% confidence. This can be done by estimating conditional quantiles instead of conditional means. We'll now see how to make such an estimation.

The q quantile of a continuous random variable D is the value o such that $P(D < o) = q$. Quantiles are often called percentiles when q is expressed as a percentage. Most people with very young children are used to see conditional percentiles in their children's health bulletins. For example, these bulletins normally contain a graph where the child's height is plotted as a function of age, and the plotting area also show the 25%, 50% and 75% quantiles as a function of age, so that one can easily see whether the evolution of the height is normal.

We shall now see that quantiles can be estimated by means of a cost of the form

$$C(\mathbf{x}, d) = \begin{cases} -a(o-d) & \text{if } o < d \\ b(o-d) & \text{if } o \geq d, \end{cases} \tag{33}$$

where $a, b > 0$, so that C is always non-negative and has its minimum (equal to zero) for $o = d$. For simplicity, we'll now only perform step 1 of the derivation, i.e., perform the step that considers a system with a single input pattern. The other steps would then follow in a manner entirely similar to what was done above for the quadratic error, and there's no need to explicitly make them here. We shall, however, state their result, for completeness.

Let us, then, consider a system with a single input pattern, and designate the system's output by o, as usual. The cost function is given by

$$E = \int_{-\infty}^{o} b(o-d)\,\mathrm{d}d - \int_{o}^{+\infty} a(o-d)\,\mathrm{d}d. \tag{34}$$

Differentiating relative to o raises the issue of how to differentiate an integral that has the variable relative to which we want to differentiate both in one of the limits of integration and in the integrand. That is not difficult to solve. We rewrite the cost function as.

$$E = \int_{-\infty}^{o_1} b(o_2 - d)\,\mathrm{d}d - \int_{o_1}^{+\infty} a(o_2 - d)\,\mathrm{d}d. \tag{35}$$

with $o_1 = o_2 = o$, and use the chain rule of differentiation for functions of more than one variable. We have

$$\begin{aligned}
\frac{\mathrm{d}E}{\mathrm{d}o} &= \frac{\partial E}{\partial o_1}\frac{\mathrm{d}o_1}{\mathrm{d}o} + \frac{\partial E}{\partial o_2}\frac{\mathrm{d}o_2}{\mathrm{d}o} \\
&= b(o_2 - o_1) + a(o_2 - o_1) + \int_{-\infty}^{o_1} b\,\mathrm{d}d - \int_{o_1}^{+\infty} a\,\mathrm{d}d \\
&= b(o - o) + a(o - o) + b\,P(d < o) - a\,P(d > o) \\
&= b\,P(d < o) - a[1 - P(d < o)].
\end{aligned}$$

Equating to zero, we obtain

$$P(d < o^*) = \frac{a}{a + b}. \tag{36}$$

Therefore, the optimum output is the $a/(a + b)$ quantile of d. The result for a system with multiple input patterns, already taking into account the comments made above regarding real-life situations, is

$$\boxed{P(d < o|\mathbf{x}) \approx \frac{a}{a + b}.} \tag{37}$$

Figure 6 illustrates the estimation of the 80% quantile for the same training data that we have previously used.

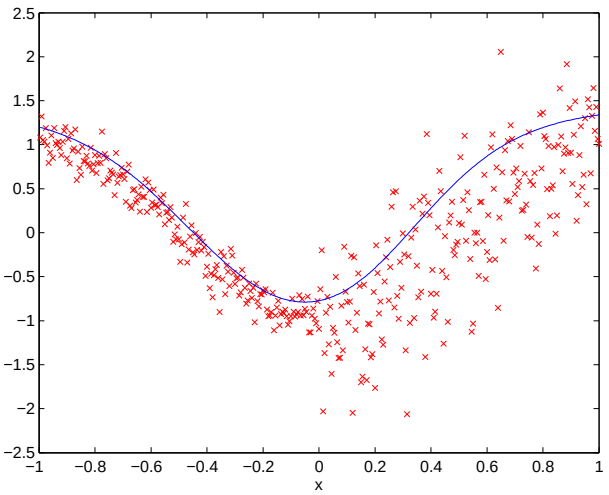

Figure 6: Estimation of the 80% quantile from a set of data. The estimation was performed by an MLP with three hidden units.

In many situations, it is important to estimate a prediction interval for the data, and not simply a quantile.[7] For example, it might be important to be able to say that we estimate, with 90% confidence, that the temperature of a certain system is between, 250°C and 270°C. This can be done by estimating a lower and an upper quantile whose levels differ by the desired confidence level. For example, for a 90% confidence level we would normally choose the 5% and 95% quantiles.[8] Figure 7 gives an example of the estimation of a 90% prediction interval on the same data as above. We can see that the interval is much wider in the right half of the image, where the noise variance is higher, than in the left half of the image. We can even count the number of data points that fall outside the interval, and confirm that it is very close to 40, which is 10% of the number of training patterns.

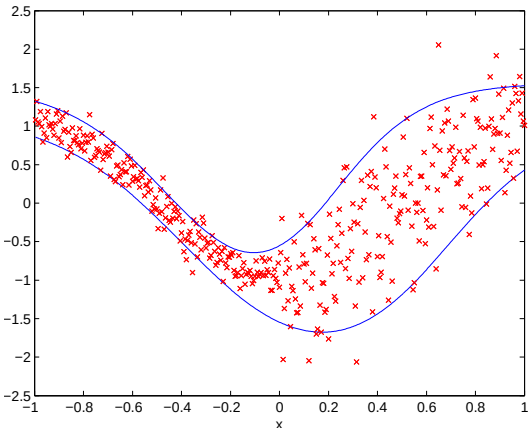

Figure 7: Estimation of a 90% prediction interval from a set of data. The estimation of both the 5% and 95% quantiles was performed by MLPs with three hidden units.

6 Final comments

We have presented a few of the statistical aspects of systems trained in a supervised way. There are many more connections between machine learning and statistics. In fact, machine learning can be viewed as statistical inference, and therefore as a branch of statistics. We'll see some more connections when we study maximum likelihood, maximum a posteriori and minimum message length estimators, later in this course. Two good books to expand on this subject are [1] and [2].

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 1995.

[7] The concept of prediction interval is somewhat similar to the one of confidence interval. An important difference is that prediction intervals refer to values of the random variable itself, while confidence intervals refer to values of a parameter of a probability distribution (for example, to values of the mean of a Gaussian distribution with known variance).

[8] We could choose other values, for example 2% and 92%, but this would only make sense if we knew there was an asymmetry between the two tails of the distribution $p(d|\mathbf{x})$, because otherwise we would normally obtain a wider – and therefore less accurate – prediction interval than with symmetrical quantiles.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, 2001.