Titel

Dries Naert

March 9, 2016

Contents

Abstract

Introduction

Chapter 1

Precise probabilities

1.1 Sets

1.2 Measures and probabilities

1.3 Bayesian viewpoint

1.4 Bayes' rule

In this section I will explain Bayes' rule, since this rule is the working mechanism of the Bayes classifier we'll construct later on. A simple way of explaining this rule is by letting an event correspond to a 2-dimensional geometric shape. The value of the probability of this event is then the area of the corresponding shape. The rule is here explained with 2 circular shapes. In general, this kind of setup is called an Euler diagram and a picture of the situation is drawn in **??**. Now suppose we see the picture and after that someone would assure us that event A actually has happened, some cutting and rescaling of the picture has to be done to fit it to the new information. **??** is our new state of knowledge. Everything outside A has been cut off and the picture was rescaled so that $P'(A) = 1$. This can be done by dividing all areas by the original area of A. The conditional probability of event B, given event A is its new area:

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \ . \tag{1.1}$$

This is the rule of Bayes. It gives a way of calculating the probability of event B, given event A (happened). As a result of Bayes' rule, we also have a way of calculating $P(A \cap B)$, assuming we have information about $P(B|A)$ and $P(A)$. Calculating $P(B \cap A)$ this way is called the multiplication rule.

Example. A man goes to a doctor and will undergo a test for a horrible

Table 1.1: Test result probabilities

	Sick	Not sick
+	0.9	0.15
-	0.1	0.85

disease. The result of this test can be positive (+) or negative (-). The probabilities of the test results, conditioned on the state of (not) being sick is given in table 1.1. The doctor tells the man that based on what he knows at that very moment, the probability of the man having the disease is 0.01. If the test would be positive, what is the probability of the man having the disease? We can solve this by directly applying Bayes' rule, this yields

$$\begin{aligned} P(Sick|+) &= \frac{P(+\cap Sick)}{P(+)} = \frac{P(Sick)P(+|Sick)}{P(+)} \\ &= \frac{P(Sick)P(+|Sick)}{P(+|Sick)P(Sick) + P(+|\overline{Sick})P(\overline{Sick})} \\ &= \frac{0.01 \cdot 0.9}{0.9 \cdot 0.01 + 0.15 \cdot 0.99} \approx 0.057 \ . \end{aligned}$$

As you can see from the example, to apply Bayes' rule in practice, I used the multiplication rule to find numerator and both the law of total probability and the multiplication rule to find the denominator. Being sick or not being sick is indeed a partition of the whole probability space and this partitioning allowed me to use the law of total probability.

1.5 Decision theory?

Chapter 2

Classification

The idea behind classification is that there is an unknown probability distribution $P(X_1, ..., X_n, Y)$, from which we have drawn samples/data points $d(x_1, ..., x_n, y) \equiv d(\boldsymbol{x}, y)$. We'll call the x's the features/input and y the class/output. In the future we will be given specific feature values, and we want to predict what the class is. A classifier does exactly that. It is a function that maps features on classes. The class variable always takes discrete values, the features are allowed to be both discrete or continue. An example of such a distribution-features-class triplet could be a situation where the class is a person's favorite movie. The features could then be things like 'likes this actor', 'likes that actress', 'likes westerns', etc. The distribution itself does not really exist, but it is represented by all people that have a favorite movie. We'll call such a representation of the distribution the population. It is typical in classification problems that the different classes are given. For the features you will often be able to choose whether to include a feature in the problem or not. For example, if you were with Netflix, you cannot go around the fact that you are interested in the movies your clients like. But it is up to you then to decide if features like gender or age of the client play a role in this problem.

2.1 The hypothesis set

Setting up the domain and codomain is one part of the initialization phase. Another part is determining what the hypothesis set $\mathcal{H}$ is. This hypothesis set is not anything more than a set of functions, also called hypotheses in this context, with an appropriate domain and codomain. When everything is done, we will be given one final function/hypothesis g that we will use to make future predictions. The idea of the hypothesis set is that the final hypothesis g must be in our hypothesis set $\mathcal{H}$.

Example: linear functions. Take the specific case where we have 2 continuous features x and y which we will use to predict a binary class $c \in \{-1, 1\}$.

Note that the domain here is $\Re^2$, ie the plane. A possible hypothesis set for this problem could be

$$h := (x, y) \to \text{sign}(w_0 + w_1 x + w_2 y), \tag{2.1}$$

where $(w_0, w_1, w_2) \in \Re^3$, and

$$\begin{aligned} \text{sign}(z) &= 1 && \text{if } z \geq 0 \\ &= -1 && \text{else.} \end{aligned}$$

This hypothesis set corresponds to all lines in the plane. Points at one side of a line are mapped to one class while points at the other side are mapped to the other class.

2.2 Cost and utility function

A short recap of the situation. We are given a list of feature values $\boldsymbol{x}$ and will be asked to predict the class y. Giving a correct prediction of one class is a good thing, and giving a correct prediction of another class is also a good thing. However these 2 good things are not necessarily equally good. The same goes up for wrong predictions, 2 different wrong predictions are not necessarily equally avoidable either. The utility function is a way of putting numbers on how pursuable good is and how avoidable bad is. More precise, in the context of classifiers, a utility function $U(\hat{C}, C)$, accepts 2 values on the same domain. The 1st variable is what the classifier predicts, the 2nd variable is what the class really is. The output of this utility function is a scalar that can be interpreted as some reward-like variable we'd like to be high in future events.
Utility functions can often be given in a table form. An example is given in table (**??**). The classifier will have to predict whether tomorrow will be rainy or sunny. Correct predictions are treated equally good and are given a reward of 1. Predicting rain when there really will be sun is a little bit annoying, so it gets a punishment of -1. However, that wouldn't be as bad as predicting a sunny day when it really will be rainy so this one gets a punishment of -2.

Expected utility of a single hypothesis. We can now define the expected utility E_U of a single hypothesis h,

$$E_U(h) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} U(h(\boldsymbol{x}_i), y_i), \tag{2.2}$$

where $\boldsymbol{x}_i$ and y_i are drawn from an unknown distribution $P(X, Y)$. In words, this expected utility is the value we would obtain if we were to run a single hypothesis h on the entire (infinite) population and calculate our average utility with this hypothesis. In other words, it is how much utility you will get on average if you use this hypothesis as classifier.

2.3 Generalization and bias-variance

Generalization If you have the expected utility of every hypothesis in your set, solving the problem becomes trivial. You then just pick the hypothesis with highest expected utility. However, we never have the real expected utility, what we do have is a dataset $\mathcal{D}$. We can calculate the average utility for every hypothesis on this dataset. We will name the average utility on the dataset $\hat{E}_U(h)$. The hat indicates that it is not a real expected value but an estimate based on the dataset. The generalization error e is then defined, with g the final hypothesis

$$e = \hat{E}_U(g) - E_U(g). \tag{2.3}$$

When we come up with a final hypothesis, we want some guarantees that the generalization error is small. Something we can do to have a lower generalization error is to gather more data. Another thing is working with smaller hypothesis sets.
If we rewrite eq (2.3), we obtain

$$E_U(g) = \hat{E}_U(g) - e. \tag{2.4}$$

This means that our real performance is the difference of the performance on the dataset minus the generalization error. We want the generalization error e to be small and at the same time the $\hat{E}$ term to be large. This is a trade off. A large hypothesis set allows good performance on the training set (large $\hat{E}_U(g)$) but impairs the ability to generalize (large e as well). On the other hand, small hypothesis sets have a poor performance on the training set but generalize better.

Bias and variance I will call the target function f_t the function that satisfies

$$E_U(f_t) \geq E_U(g), \tag{2.5}$$

where g is just any function with the correct domain and codomain. The target function is the best function we can have for a given set of features. When this function is not in the hypothesis set we say that we have bias. If you now call h_{best} a hypothesis that satisfies

$$E_U(h_{\text{best}}) \geq E_U(h), \tag{2.6}$$

where here h is any other function in the hypothesis set. This is the best function inside the hypothesis set. The amount of bias b can then be defined as

$$b = E_U(f_t) - E_U(h_{\text{best}}). \tag{2.7}$$

We will refine this definition a bit in the next section, because the bias has some dependence on the training algorithm as well. To explain what the variance is, think of the hypothesis set as being a darts board. The bulls eye corresponds to the best hypothesis h_{best}. Throwing a dart is then the equivalent of coming up with a final hypothesis. Now suppose that on average you throw in the center, ie on average you have the best hypothesis of the set. But that does not mean you are doing anything good. Fig (**??**) shows the situation why not. Both throw on average to the center, but you'd be happier with the first one in your team. We can intuitively decompose the average utility of a the final hypothesis U_{av} on a way like

$$U_{\mathrm{av}} = U_{\mathrm{max}} - U_{\mathrm{bias}} - U_{\mathrm{var}}. \tag{2.8}$$

This puts the trade off in a different perspective. Training on more data leads to a lower U_{var} but leaves U_{bias} unaffected. Working with larger hypothesis sets will in general increase variance and reduce bias, but this depends on the training algorithm as well.

2.4 Training algorithm

2.5 Cross-Validation

Chapter 3

Naive Bayes classifier

A Naive Bayes classifier is, like all classifiers, a function. This classifier works in 2 steps. First it maps its input onto a set of probabilities, 1 for each of what we will call the classes (of the object). In a 2nd step, the classifier will use a decision rule to pick one of the classes, based on the previously calculated probabilities. A popular decision rule is to pick the class that has the highest probability, aimed at maximizing the number of correct classifications. One of the things we will do with this classifier, is to play with different decision rules. We'll also introduce indecisiveness, that is we allow the classifier to say 'I don't know what class this object belongs to'. The classifier will eventually have to estimate some parameters, which will end up not being different from certain probabilities.

3.1 Bayes classifier

If you would construct a Bayes classifier yourself, one of the things you will have to know is $P(C|F_1, F_2, ..., F_n)$. Read this as the probability that C is in a certain class, given a combination of some features, the F_i's, occurred. Comma's between the F_i's have the same meaning as the intersection operator '$\cap$' in this context. A Bayes classifier will calculate this conditional probability by using, surprisingly, Bayes' rule. After applying the multiplication rule on the numerator, the formula becomes

$$P(C|F_1, F_2, ..., F_n) = \frac{P(C)P(F_1, F_2, ..., F_n|C)}{P(F_1, F_2, ..., F_n)} \ . \tag{3.1}$$

The denominator will always be calculated by using the law of total probability, conditioning on the class. Now let's take a look at the expression $P(F_1, F_2, ..., F_n|C)$. This is a function with a discrete domain $D = \{C \times F_1 \times F_2 \times ... \times F_n\}$ and continuous codomain $Co = [0, 1]$. In general, machine learning methods do not try to construct a look-up table of input-output relations.

Instead, typical methods parametrize an unknown function eg with radial basic functions (see also chapter 2 for more information about this). We will do something similar here. But before simplifying things, let's manually count how many parameters we'd currently have to estimate. Note that these parameters I am talking about are not different from the probabilities of expression (3.1). Say the class we are interested in has n_c possible values and we have N features, with each feature having n_i possible values. The denominator of (3.1) can be calculated if we can calculate the nominator, via the law of total probability, so let's not spend a parameter on the denominator. $P(C)$ is completely determined by $n_c - 1$ parameters. To determine $P(F_1, F_2, ..., F_n|C)$ we have to estimate for every element in C, $n_1 \cdot ... \cdot n_N - 1$ parameters. Add everything together and we end up with $n_c \cdot n_1 \cdot ... \cdot n_N - 1$ parameters to estimate. It should be clear that this number easily blows up because of the multiplications. This number will be reduced by the assumption of conditional independence as explained in section 3.2.

3.2 Conditional Independence

Conditional independence is the assumption that makes the Bayes classifier a Naive Bayes classifier. To calculate the Bayesian probability we had the troubling function $P(F_1, F_2, ..., F_n|C)$. Now let's expand this thing by applying Bayes' rule first and then the multiplication rule multiple times.

$$\begin{aligned}
P(F_1, F_2, ..., F_n|C) &= \frac{P(C \cap F_1 \cap ... \cap F_n)}{P(C)} \\
&= \frac{P(F_1|F_2 \cap ... \cap F_n \cap C)P(F_2 \cap ... \cap F_n \cap C)}{P(C)} \\
&= \frac{P(F_1|F_2 \cap ... \cap F_n \cap C)P(F_2|F_3 \cap ... \cap F_n \cap C)P(F_3 \cap ... \cap F_n \cap C)}{P(C)} \\
&= ... \\
&= \prod_{i=1}^{n} P(F_i|\bigcap_{i+1}^{n} F_i \cap C)
\end{aligned}$$

The defining assumption that we will call the conditional independence assumption is

$$P(F_j|\bigcap_{i \neq j} F_i \cap C) = P(F_j|C), \tag{3.2}$$

where the intersection can be taken over any subset of F_i's. This way equation (3.1) reduces to

$$P(C|F_1, ..., F_n) = \frac{\prod_i P(F_i|C)}{\sum_j \prod_i P(F_i|C_j)} \ . \tag{3.3}$$

This is the general form of the Naive Bayes classifier and will be used thoroughly. The numbers of parameters to estimate is reduced. Take the same example where we had N features, with each feature n_i possible values and n_c classes. The total number of parameters V to estimate then becomes

$$V = n_c(n_1 + ... + n_N - N) + n_c - 1. \tag{3.4}$$

The $-N$ in the sum arise from the N relations where $\sum P(F_i|C) = 1$. Whereas in the case without the independence assumption we had as general formula for number of parameters V',

$$V' = n_c(n_1 \cdot ... \cdot n_N) - 1. \tag{3.5}$$

3.3 Examples

In this section I will show some fictitious examples to get some better understanding of the whole mechanism, when it can fail and when it should not fail. All features and classes in the examples will be binary unless stated otherwise, a binary feature I will call an attribute.

Example 1. This example illustrates how 2 uninformative attributes are not so uninformative after all because the have perfect complementary information. In the setup we have 2 attributes A and B and a class C such that

$$P(C) = P(A|C) = P(A|\overline{C}) = P(B|C) = P(B|\overline{C}) = \frac{1}{2}$$

$$\begin{aligned} A \text{ and } B &\Rightarrow C \\ \overline{A} \text{ and } \overline{B} &\Rightarrow C \\ A \text{ xor } B &\Rightarrow \overline{C}. \end{aligned}$$

In this situation the naive classifier will always tell us that $P(C|A,B) = \frac{1}{2}$. If instead the classifier would be allowed to drop the conditional independences, the function becomes deterministic and we can obtain a perfect accuracy. In situations like these we should join the two attributes to one feature that takes 4 values, 1 for every combination of the attributes.

Example 2. This example is somehow the adverse of the previous. We begin with 2 truly conditional independent attributes A and B. Let

$$\begin{aligned} P(A|C) = P(B|C) = P(\overline{A}|\overline{C}) = P(\overline{B}|\overline{C}) &= 0.8 \\ P(\overline{A}|C) = P(\overline{B}|C) = P(A|\overline{C}) = P(B|\overline{C}) &= 0.2 \\ P(C) &= \frac{1}{2} \; . \end{aligned}$$

We calculate the conditional probabilities of C,

$$\begin{aligned}
P(C|A,B) &= \frac{0.5 \cdot 0.8 \cdot 0.8}{0.5 \cdot 0.8 \cdot 0.8 + 0.5 \cdot 0.2 \cdot 0.2} = 0.941 \\
P(C|A,\overline{B}) &= P(C|\overline{A},B) = 0.5 \text{ (because of the symmetry)} \\
P(C|\overline{A},\overline{B}) &= 1 - P(C|A,B) = 0.059 \text{ (because of the symmetry).}
\end{aligned}$$

Now we want to do better than that. I propose we add an attribute A' to the classifier. The problem with this attribute A' is that it is perfectly correlated with attribute A. Because the relation ($A \Leftrightarrow A'$), this one does actually not learn us anything at all and the real probabilities do not change. However, the estimated ones will.

$$\begin{aligned}
P(C|A,B,A') &= \frac{0.5 \cdot 0.8 \cdot 0.8 \cdot 0.8}{0.5 \cdot 0.8 \cdot 0.8 \cdot 0.8 + 0.5 \cdot 0.2 \cdot 0.2 \cdot 0.2} = 0.985 \\
P(C|A,\overline{B},A') &= \frac{0.5 \cdot 0.8 \cdot 0.2 \cdot 0.8}{0.5 \cdot 0.8 \cdot 0.2 \cdot 0.8 + 0.5 \cdot 0.2 \cdot 0.8 \cdot 0.2} = 0.8 \\
P(C|\overline{A},\overline{B},\overline{A'}) &= 1 - P(C|A,B,A') = 0.015 \text{ (because of the symmetry).}
\end{aligned}$$

If we compare the probabilities we'd like the classifier to give us and the ones with the extra, but very dependent, feature, we see the mechanism of how dependent features can bias probabilities.

3.4 Decision rules

For the time being, we assume perfect knowledge about $P(C|F)$ and focus our attention on how to decide. I'll show 3 decision rules, where the second and third rule will appear to be equivalent generalizations of the first one. To avoid confusion, class decisions made by the classifier will be accompanied with the hat symbol, so read $\hat{C}$ like 'output of the classifier is $\hat{C}$'.

Maximize total accuracy A first way of picking a class C, is to pick the one for which $P(C|F)$ is highest. Symbolically, this rule gives

$$\hat{C} = \operatorname*{argmax}_C P(C|F). \tag{3.6}$$

I will show informal why this rule gives the highest number of classifications in the long run, also called total accuracy. Remember that for a frequentist there is no difference between $P(C)$ and $\lim\limits_{N\to\infty} \frac{N_C}{N}$, where N is the total amount of cases presented and N_C is the amount of times the object turns out to be in class C. This interpretation allows us to calculate the total accuracy,

$$TotAcc = \sum_i \max_C P(C|F_i)P(F_i). \tag{3.7}$$

The 'max' comes from the decision rule. Given the probabilities $P(C|F_i)$, we pick the class with the highest probability. The value of $\max_C P(C|F_i)$ is then the accuracy we would obtain if only the F_i feature combination would occur. What I do then, is to take a weighted sum of these partial accuracies over all possible feature combinations that can occur. The weights, which are the $P(F_i)$'s, represent the amount of times we will have these corresponding partial accuracies. If we now look at equation (3.7), the only way the decision rule affected the total accuracy is when we took the maximum of $P(C|F)$. We had no saying in the $P(F_i)$'s, the world decides these values for us. Now we see that no other rule than picking the class with highest probability can give a higher total accuracy.

Threshold values We generalize the maximize total accuracy decision rule (3.6) with threshold values as follows

$$\hat{C} = \operatorname*{argmax}_{C \in a} P(C|F), \text{ where} \tag{3.8}$$

$$a = \{C \text{ for which } P(C|F) > t_C\} \tag{3.9}$$

In words this means that if $P(C|F)$ is larger than some threshold t_C, the classifier puts C in a set of candidates. Note that t_C may be different per class. Then it picks, as before, the class with highest probability in this set of candidates. An alternative to this would be to not choose when there is more than 1 candidate, however this does not generalize the first decision rule very well. The situation might occur that none of the probabilities passes its threshold and the set of candidates is empty. We call the classifier undecided in these situations. With decision rules that allow indecisiveness, we must clearly distinguish between the accuracy and total accuracy. In the following, let Acc be the accuracy, $TotAcc$ the total accuracy, d stands for decisiveness, N_+ is the number of correct classifications, N is the number of cases presented and finally N_d is the number of cases that were decided. Then we have the relations

$$\begin{aligned} Acc &= \frac{N_+}{N_d} \\ TotAcc &= \frac{N_+}{N} \\ d &= \frac{N_d}{N}. \end{aligned} \tag{3.10}$$

Maximize expected utility In this context, the utility function $U(\hat{A}, C)$ should be read as what you will get as payoff (utility) when an agent (the classifier) decides to undertake action $\hat{A}$ and the class ends up being C. After we observed the feature combination F and the probabilities $P(C|F)$ are calculated, we can calculate expected payoffs for different actions $\hat{A}$

$$E_{\text{U}}(\hat{A}|F) = \sum_i U(\hat{A}, C_i) P(C_i|F). \tag{3.11}$$

The rule here is to pick the action with highest expected value

$$\hat{A} = \underset{\hat{A}}{\operatorname{argmax}}\, E_{\mathrm{U}}(\hat{A}|F). \tag{3.12}$$

I'd like to stress that although C can only take a number of values equal to the number of classes, $\hat{A}$ can take any number of values with general utility functions. This is because $\hat{A}$ does not represent a class with utility functions but an action that someone undertakes. Here, we will only work with utility functions where $\hat{A}$ can take the same values as C plus 1 extra value, corresponding to the no decision made situation. The action in the utility function is then equivalent to the classifier (not) choosing a class. This way $\hat{A} = \hat{C}$, where $\hat{C}$ is still the classifiers output.

3.5 Training the classifier

So far, I have ignored the way we obtain the probabilities $P(C)$ and $P(F|C)$. In this section we look at some possibilities for getting these values.

Guessing If someone has no data available but happens to be an 'expert' in the matter, he can try to guess the probabilities. Whether this is a good idea might depend on the 'expert'. I included this method just to show that the next one is not unique.

Use data In normal situations we have a set of input-output/feature-class pairs. To estimate all probabilities $P(F|C)$, we will give every feature its own table, I'll call these the feature tables. The number of rows in each one of these is then equal to the number of values that particular feature can have. The number of columns are the same for every table and are equal to the number of classes. Now we have to fill in some numbers in the tables. We do this by going over the given feature-class pairs one by one. One of these pairs looks like $(f_1, f_2, ..., c)$. Then, we go for every feature to its table and add 1 to the entry of which the row and column correspond to the one's of the feature-class pair we were considering. Table 3.5 is an example of how one such feature table looks like after the numbers are filled in. Estimating the actual probabilities goes as follows. The probability $P(F_1 = f_1|C = c)$ eg is estimated by looking in the table of feature F_1. In this table take the number of the row associated with f_1 and column associated with c and divide this number by the sum of numbers in the column of c. The probabilities $P(C)$ can be estimated by looking in any one of the tables. $P(C = c)$ is estimated by taking the sum of the elements of the column corresponding to c (in any table) and divide this number by the sum of all elements in the table. I illustrate this method by a simple example, see also table 3.5. This would be the feature table for a second feature, which can take 3 values and were we have 3 classes. If we were interested in, let's say,

Table 3.1: Feature table

	C_1	C_2	C_3
F_2= a	5	1	7
F_2= b	6	2	8
F_2= c	9	4	3

$P(F_2 = b|C_3)$ the calculation would be

$$P(F_2 = b|C_3) = \frac{8}{7 + 8 + 3}.$$

The probability of $P(C = C_2)$ would then become

$$P(C = C_2) = \frac{1 + 2 + 4}{45},$$

because the sum of all elements in the table is 45.
This simple illustration should make an nice property of the Bayes classifier clear. Namely, we can train it by reading a line of the dataset, do a single plus one addition in every one of our feature tables and repeat for all data available in the world. In the end, or at any time we want, we do only a small amount of summations and divisions to get the probabilities. This very simple training algorithm has the property that the time it takes to train is proportional with the size of the dataset.

Laplace smoothing

Chapter 4

Imprecise probabilities

(long intro)

4.1 Formalisms

4.2 Geometric interpretation

Chapter 5

Imprecise Naive Bayes classifier

(long intro)

5.1 Decision rules

5.2 Training algorithm

Chapter 6

Performance and comparison of precise and imprecise methods

(short intro)

6.1 Measures of performance

6.2 Precise Naive Bayes

6.3 Imprecise Naive Bayes

6.4 Discussion