# Chapter 7

# Naïve Bayes

*Bayes Classifier and Naïve Bayes Classifier. Parametric and non-parametric models. Generative vs Discriminative classifiers. Comparing classifiers.*

## 7.1 Bayes rule

Let us imagine we have two random variables, $X$ and $Y$. The probability of $X = x_i$ and $Y = y_j$ is called the *joint probability* and is represented as:

$$p(X = x_i, Y = y_j)$$

The probability of $X = x_i$ is the sum of the joint probabilities of all $Y$ values and $X = x_i$:

$$p(X = x_i) = \sum_{j=1}^{N} p(X = x_i, Y = y_j)$$

This is called the *sum rule* of probability. If we imagine representing the possible values of $X$ and $Y$ in a matrix counting the probability of each combination, $p(X = x_i)$ is obtained by summing the respective column. This is called the *marginal probability* because we can imagine summing it on the margin of the matrix with the *joint probabilities*, as shown in Table 7.1.

The *conditional probability* for $Y = y_j$ given that $X = x_i$, written $p(Y = y_j|X = x_i)$, is the proportion of $p(X = x_i, Y = y_j)$ to $p(X = x_i)$:

$$p(Y = y_j|X = x_i) = \frac{p(X = x_i, Y = y_j)}{p(X = x_i)}$$

This means that

$$p(X = x_i, Y = y_j) = p(Y = y_j|X = x_i)p(X = x_i)$$

This is the *product rule*, which relates the joint probability distribution to the conditional and marginal probabilities. More briefly, these rules can be summarized as follows:

$$\begin{aligned} &\textit{sum rule} && p(X) = \sum_{j=1}^{N} p(X, Y_j) \\ &\textit{product rule} && p(X, Y) = p(Y|X)p(X) \end{aligned}$$

Table 7.1: Joint and Marginal probabilities

| Y X | 1 | 2 | 3 | 4 | P(Y) |
|---|---|---|---|---|---|
| 2 | 0,06 | 0,026 | 0,051 | 0,012 | 0,189 |
| 3 | 0,045 | 0,001 | 0,046 | 0,016 | 0,152 |
| 4 | 0,035 | 0,015 | 0,065 | 0,045 | 0,218 |
| 5 | 0,006 | 0,033 | 0,057 | 0,039 | 0,157 |
| 6 | 0,029 | 0,004 | 0,054 | 0,035 | 0,127 |
| P(X) | 0,175 | 0,079 | 0,273 | 0,147 | |

This table shows the joint probabilities for different combinations of $X$ and $Y$. The marginal probabilities are computed on the margins by summing the respective rows and columns.

Given that joint distributions are symmetric, $p(Y,X) = p(X,Y)$ (just transpose the matrix on Table 7.1), applying the *product rule* we can obtain *Bayes' rule*:

$$p(Y,X) = p(X,Y) \Leftrightarrow p(Y|X)p(X) = p(X|Y)p(Y) \Leftrightarrow p(Y|X) = \frac{p(Y)p(X|Y)}{p(X)}$$

A *frequentist* interpretation will see these probabilities as the frequency of random events in the limit of an infinite number of trials. For example, saying that a coin has a 50% probability of falling "tails" means that, as the number of trials grows to infinity, the fraction of "tails" will tend towards 0.5. But a Bayesian interpretation of probabilities sees the probability values as a measure of our knowledge about the propositions. Under this interpretation, we can see *Bayes' rule* as telling us that the probability of hypothesis $Y$ being true (i.e. our knowledge of $Y$) given evidence $X$, which is $p(Y|X)$, has been modified relative to the prior probability of $Y$, which is $p(Y)$, by the probability of $X$ given $Y$, or the *likelihood* of $Y$, written $p(X|Y)$, normalized by the probability of the data $X$.

This interpretation allows us to consider the probability of an example $x$ belonging to class $c$ as the conditional probability of class C given the features of $x$: $p(C = c|X = x)$, which would not make as much sense in a frequentist interpretation, unless we assumed the class was determined by the features only with some probability.

## 7.2 Bayes Classifier

Using *Bayes' rule*, we can write that the probability of an example with feature vector $x$ belonging to class $c$ is:

$$p(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)}$$

In other words, the probability of $x$ belonging to $c$ is the prior probability of any point belonging to $c$ multiplied by the likelihood of $C = c$ and divided by the probability of drawing example $x$ at random. Since the probability of drawing example $x$ does not depend on our classifier, we can simplify this expression to:

$$p(C = c|X = x) \propto p(C = c)p(X = x|C = c)$$

But we know from the *product rule* that $p(C = c)p(X = x|C = c)$ is the joint distribution $p(C = c, X = x)$. So if we can compute the joint distribution of the classes and examples, we can choose the best class for each example. This is the *Bayes classifier*:

$$C^{Bayes} = \underset{c \in \{0,1,...,N\}}{\text{argmax}} \ p(C = c, X = x)$$

The *Bayes classifier* is ideal in the sense that it minimizes the probability of misclassifying an example. However, it is generally not feasible to compute the joint probability of the classes and features. To understand this, imagine we want to predict if a person has diabetes. We start with a sample of healthy and diabetic individuals and have each fill in a questionnaire with 20 questions on exercise practices, food, smoking, other diseases and so on. Even if the questions are only "yes" or "no", 20 questions gives us about a million combinations. To obtain a reasonable estimate of the joint probability distribution of classes (diabetic or healthy) and all combinations of possible answers we would need millions of volunteers and questionnaires. Without simplifying assumptions we cannot do this. In short, although the *Bayes classifier* is the ideal classifier in theory, in practice it is generally impossible to use.

## 7.3 Naïve Bayes Classifier

In the previous section, we saw that we can predict the class of an example by finding the maximum of the joint probability of each class and the features of that example. We can decompose this using the *product rule* as follows, considering $x_1, ..., x_n$ to be the components of the feature vector and $C_k$ the probability of the example being in class $k$:

$$p(C_k, x_1, x_2, ..., x_n) = p(C_k)p(x_1|C_k)p(x_2|C_k, x_1)...p(x_n|C_k, x_1, x_2, ..., x_{n-1})$$

Variables $A, B$ are *conditionally independent* given $X$ if:

$$p(A, B|X) = P(A|X)P(B|X)$$

That is, if their joint probability conditioned on the other variable is just the product of their probabilities conditioned on that other variable.

An example of conditional independence could be the time two persons living in the same neighbourhood arrive at home from work. These variables may not be independent because, whenever there is a strike in the public transport system, both will arrive later. So if one arrives late it is more likely that the other arrived late too. However, if we know that there was such a strike, then knowing when one of them arrived home gives us no new information about when the other will arrive, and thus the two are independent if we know if there was or was not a strike.

So, if we assume that the feature values $x_1, ..., x_n$ are *conditionally independent* given the class, it follows that:

$$p(x_n|C_k, x_1, x_2, ..., x_{n-1}) = p(x_n|C_k)$$

for any $n$. This allows us to greatly simplify the computation of the joint distribution:

$$p(C_k, x_1, x_2, ..., x_n) = p(C_k) \prod_{j=1}^{N} p(x_j|C_k)$$

or, if we take the logarithms to prevent numeric overflow or underflow problems:

$$\ln p(C_k, x_1, x_2, ..., x_n) = \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k)$$

This means that our classifier can be:

$$C^{\text{Naïve Bayes}} = \underset{k\in\{0,1,\ldots,K\}}{\operatorname{argmax}} \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k)$$

This is called the *Naïve Bayes classifier* because of the assumption that all features are *conditionally independent* on the class. In general, this is not true. However, since we are not concerned with the absolute probability values but merely with finding the class that maximizes these values, the *Naïve Bayes classifier* tends to work rather well. In addition, it is very easy to apply. If we consider again the diabetes example of the previous section, for a *Naïve Bayes classifier* we would only need to find the probability distribution of each feature given the class. So we would only need to compute the proportions of yes and no for each answer in all questionnaires given to healthy subjects and the same for all questionnaires given to diabetic subjects. This should easily be done with a few dozen questionnaires instead of millions.

## 7.4 Naïve Bayes, example 1: continuous features

Let us consider a data set where each point has two continuous features and belongs to one of two classes. To train a *Naïve Bayes classifier*, we need to determine the conditional probability distribution of each feature given each class. With features that have continuous values we have several options. One is to use a *parametric model*. For example, if we assume that a feature is a normally distributed random variable when conditioned on the class, we can compute its probability distribution using the normal distribution:

$$p(x_j|C_k) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$$

where $\mu_k$ and $\sigma_k$ are, respectively, the mean and standard deviation of the values of feature $x_j$ for all points in class $C_k$. This is a *parametric model* because the model is completely determined by a specific set of parameters, and there are different probability distributions that we can consider. Alternatively, we can use a *nonparametric model* for the distribution. This is a model that, even though it can have parameters, it is not completely determined by the parameters. A histogram is an example of a *nonparametric model*. It has one parameter – the size of the bins used to partition the values – but it cannot be completely determined by that parameter, since we also need to count the values. Another example of a *nonparametric model* for these distributions is a *Kernel Density Estimator*, as we saw in Section 6.6. Figure 7.1 compares these three models for finding the distribution of one feature from one of the classes of our data set.

A kernel density estimator seems to be the best option, and it generally is unless we know the distribution function and can use a parametric model. So now we load the data and find the distributions for each of the two features in each of the two classes. The product of these distributions, for each class, is our estimate of the joint probability distribution under the naïve assumption that the features are conditionally independent given the class, which is the assumption used in the *Naïve Bayes classifier*. Figure 7.2 shows the data, the four KDE computed (two classes times two features) and the 3D plot showing the products of the probability distributions for each class, which, under the assumption of conditional independence, are the estimates of the joint probability distributions of the features given each class. The KDE was computed using a gaussian kernel and the Nadaraya-Watson estimator, as illustrated in Section 6.6.

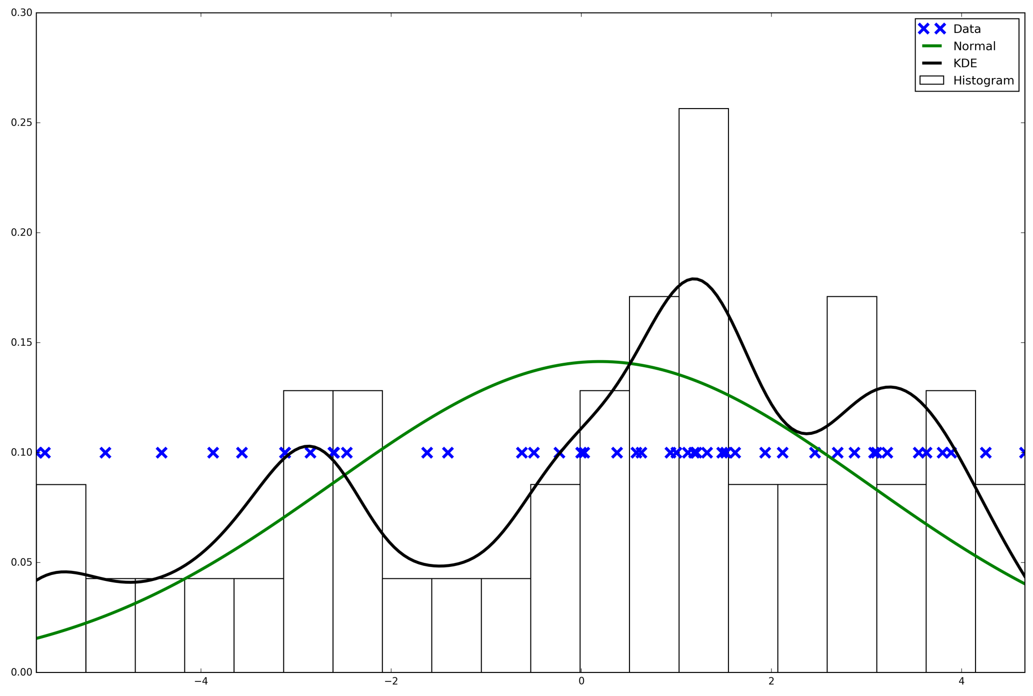

Figure 7.1: Different distribution estimates for one feature in one class of our data set. The data values are in one dimension, and represented with Y = 0.1 just to make it easier to see them.

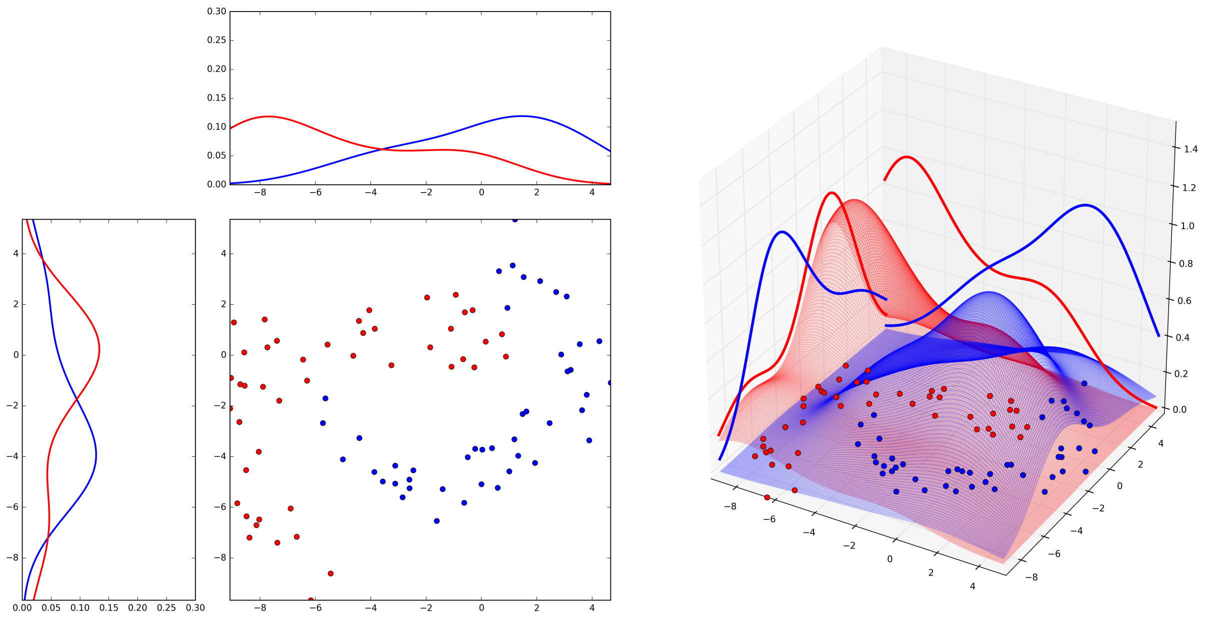

Figure 7.2: Kernel density estimation of the four distributions and the estimated joint distributions conditioned on the class (red or blue) under the Naïve Bayes assumption that the features are conditionally independent given the class. Note that the Z scale in the vertical plot was normalized to a maximum of 1 so the shape of the product plots are easier to see.

Now we just need to consider the proportion of red and blue class points in our data (the $p(C_k)$ term and find, for each point to classify, the class that maximizes:

$$C^{\text{Naïve Bayes}} = \underset{k\in\{0,1,\ldots,K\}}{\operatorname{argmax}} \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k)$$

However, the KDE we used has one parameter $h$, which determines the width of the kernel function, and different values of $h$ lead to different classifiers. Figure 7.3 shows the result of the classifier with different values of $h$.

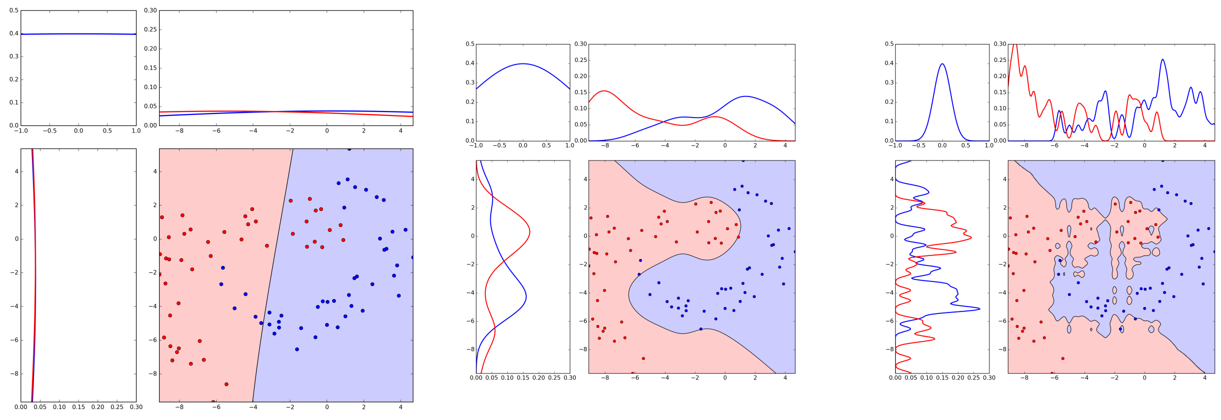

Figure 7.3: The Naïve Bayes classifier trained with this data set using different values of $h$ for the KDE. In each panel, the top-left plot depicts the kernel function resulting from the respective $h$ value.

To determine the best value we can use *cross-validation*. Figure 7.4 shows the result of 10-fold cross validation, depicting the training and validation errors as a function of the value of $h$. The best value, minimizing the validation error, was $h = 1.8$. The right panel shows the classifier retrained with the complete training set and using $h = 1.8$ for the kernel density estimators.

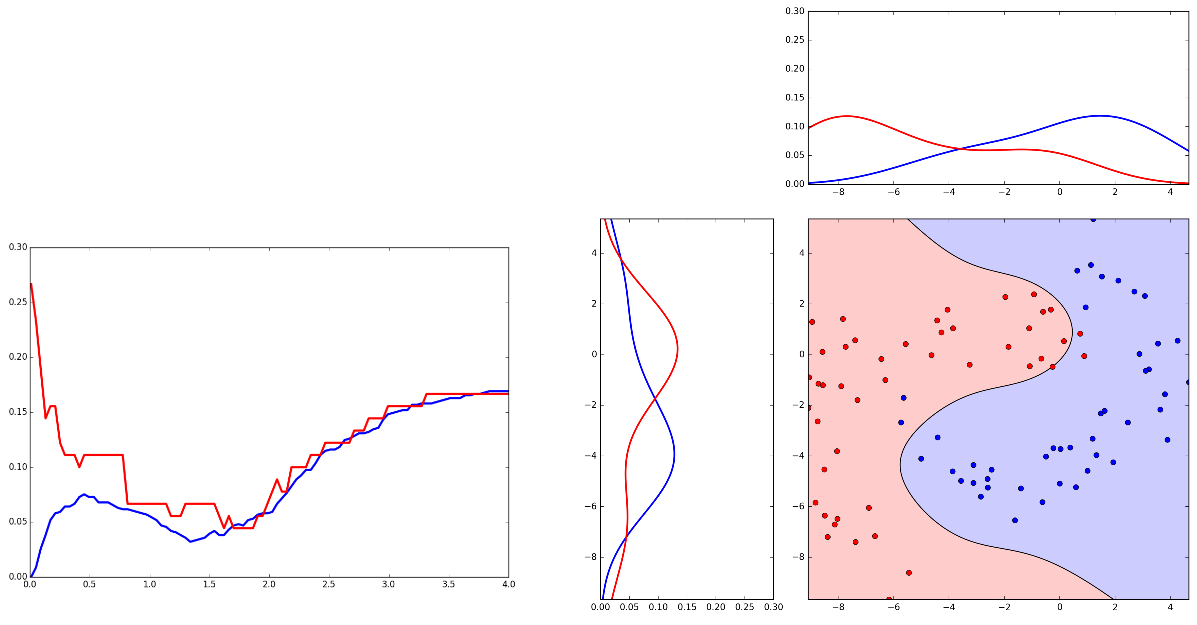

Figure 7.4: Cross-validation results and the final Naïve Bayes classifier for $h = 1.8$.

## 7.5 Naïve Bayes, example 2: categorical featues

For this example, we will be using a data set describing mushroom samples with 22 categorical features, each labelled as edible or poisonous[1]. We will be using a *Naïve Bayes classifier* to try to predict if a mushroom is edible. The features are all categorical and described in a features file:

[1]From the UCI machine learning repository: http://archive.ics.uci.edu/ml/datasets/Mushroom

```
1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
[...]
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d
```

The data is stored as strings with one sample per line. The first character indicates the class, with `p` for poisonous and `e` for edible. The rest of the line indicates the value for each feature with the corresponding character codes, separated by commas.

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
[...]
```

First we will load the information on the possible values for each feature. We will also add a possible value of "?" because, in some cases, the value is missing and missing values are indicated by this character. This function reads all the lines in the features file (specially modified so that each feature description is in a single line in the text file), splits each line on the = character and stores the following character.

```python
def get_features():
    lines = open('agaricus-lepiota.features').readlines()
    features = []
    for lin in lines:
        ft_vals = '?'
        fragments = lin.split('=')
        for frag in fragments[1:]:
            ft_vals = ft_vals+frag[0]
        features.append(ft_vals)
    return features
```

With the list of strings describing the possible values for the features, we can now load the data. This function removes the commas separating the attribute values then fills in a matrix with the index of each code. Before returning the features and class matrices, this function also shuffles the ordering of the rows. The purpose of this is to remove any correlations present in the ordering of the data file.

```python
def load_data(features,class_codes):
    lines = open('agaricus-lepiota.data').readlines()
    feat_vals = np.zeros((len(lines),22)).astype(int)  # to store indexes
    classes = np.zeros(len(lines))
    for row,lin in enumerate(lines):
        s = lin.replace(',','').strip()
        classes[row] = class_codes.index(s[0])
        for column,fv in enumerate(s[1:]):
            feat_vals[row,column] = features[column].index(fv)
    ixs = list(range(feat_vals.shape[0]))
    np.random.shuffle(ixs)
    return feat_vals[ixs,:],classes[ixs]
```

Now we can estimate the conditional probability distributions of the values for each feature given the class. The following function receives the feature value matrix and the list of possible codes for each feature. Since the features are all categorical, it is best to use histograms. The only detail to remember here is to avoid having values with a probability of zero. This can happen if the value is absent from the training set. To prevent this, we can use *additive smoothing*. Instead of simply computing the fraction of occurrences of each value, we also add a constant $\alpha$:

$$\hat{p}(x_j = k) = \frac{count(k) + \alpha}{N + \alpha d}$$

where $d$ is the number of possible values in feature $j$. This function creates a list of vectors, each vector counting the occurrences of the different possible values of the corresponding feature, starting with 1 as the value of the $\alpha$ constant. After counting, the function computes the logarithm of the fraction for each value. Logarithms are useful in this case so we can sum instead of multiplying the values.

```
def make_hists(data,features):
    hists = []
    for feat in features:
        hists.append(np.ones(len(feat)))
    for row in range(data.shape[0]):
        for column in range(data.shape[1]):
            hists[column][data[row,column]] +=1
    for ix in range(len(hists)):
        hists[ix] = np.log(hists[ix]/float(data.shape[0]+len(features[ix])))
    return hists
```

Now we need to load the data and split it into a training and test set. Previously, we have done this with *random sampling*, which consists of splitting the sets at random. However, it is best to have the same proportion of the two classes in the training and test set. So this time we will use *stratified sampling*. First we split the data in two sets, corresponding to the edible and poisonous examples. Then we draw the same fraction of each set for the test set. Since the `load_data` function shuffles the examples at random, this is easy to do by simply splitting the matrices in two.

```
def split_data(features,test_fraction):
    feat_vals,classes = load_data(features,'ep')
    edible = feat_vals[classes==0,:]
    poison = feat_vals[classes==1,:]
    e_test_points = int(test_fraction*edible.shape[0])
    e_train = edible[e_test_points:,:]
    e_test = edible[:e_test_points,:]
    p_test_points = int(test_fraction*poison.shape[0])
    p_train = poison[p_test_points:,:]
    p_test = poison[:p_test_points,:]
    return e_train,p_train,e_test,p_test
```

Now all we need is a function to classify examples. The function `classify` receives the histograms with the logarithms of the estimated probabilities and the logarithm of the prior probability of an example belonging to either class, $p(C_k)$ . This is simply the logarithm of the fraction of each class in the data. This function sums all the terms in this equation:

$$C^{\text{Naïve Bayes}} = \underset{k\in\{0,1,\ldots,K\}}{\text{argmax}} \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k)$$

and determines the class according to the maximum value found.

```
def classify(e_class,e_log,p_class,p_log,feat_mat):
    classes = np.zeros(feat_mat.shape[0])
    for row in range(feat_mat.shape[0]):
        e_sum = e_log
        p_sum = p_log
        for column in range(feat_mat.shape[1]):
            e_sum = e_sum + e_class[column][int(feat_mat[row,column])]
            p_sum = p_sum + p_class[column][int(feat_mat[row,column])]
        if e_sum<p_sum:
            classes[row]=1
    return classes
```

Now we put it all together and evaluate the performance of our classifier on the test set by computing the percentage of misclassifications.

```
def do_bayes():
    features = get_features()
    e_train,p_train,e_test,p_test = split_data(features,0.5)
    e_hists = make_hists(e_train,features)
    p_hists = make_hists(p_train,features)
    tot_len = e_train.shape[0]+p_train.shape[0]
    e_log = np.log(float(e_train.shape[0])/tot_len)
    p_log = np.log(float(p_train.shape[0])/tot_len)
    c_e = classify(e_hists,e_log,p_hists,p_log,e_test)
    c_p = classify(e_hists,e_log,p_hists,p_log,p_test)
    errors = sum(c_e)+sum(1-c_p)
    error_perc = float(errors)/(len(c_e)+len(c_p))*100
    print('%d errors;' % errors, ' %.2f%% error rate' % error_perc)
```

We can also look at the *confusion matrix* by counting the correct and incorrect classifications of edible and poisonous mushrooms:

| | | Real class | |
|---|---|---|---|
| | | Edible | Poisonous |
| Predictions | Edible | 2089 | 221 |
| | Poisonous | 15 | 1737 |

From the *confusion matrix* we can see that most of the mistakes in classification are in classifying as edible mushrooms that are poisonous. This is a more costly mistake than mistaking edible mushrooms for poisonous ones, and it suggests one problem that we have not considered so far, which is that minimizing misclassification alone is not the ideal option when different errors have different costs.

## 7.6 Discriminative and Generative classifiers

So far we saw three different classifiers. Logistic regression and k-Nearest Neighbours predict the class of an example from an estimate of the conditional probability of a point belonging to a class given the features. These are examples of *discriminative classifiers*. Naïve Bayes is a *generative classifier*, because in this case the classifier first estimates the joint probability distribution of the classes and features values, and then predicts the class from this joint probability. The reason why this type of classifier is called *generative* is that the joint probability distribution can be used to generate synthetic examples for each class. Figure 7.5 shows an example of training a *Naïve Bayes classifier* and then using it to generate synthetic data.

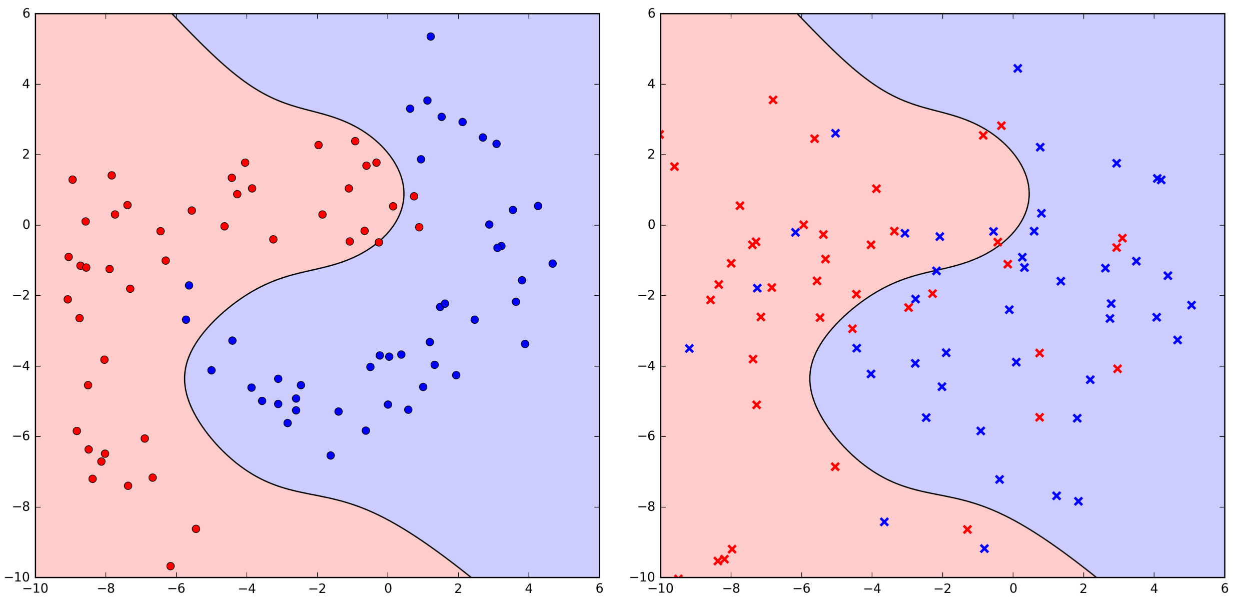

Figure 7.5: Naïve Bayes classifier trained with the data on the left panel, used to generate the set of points on the right panel.

## 7.7 Comparing classifiers

Figure 7.6 shows three different classifiers trained and tested on the same data: Logistic Regression, k-Nearest Neighbours and Naïve Bayes. These classifiers make, respectively, 10, 6 and 1 misclassification errors on the test set. The question we need to address is whether any of these classifiers is significantly better than the others. One solution is to use an *approximate normal test*. Since the number of errors result from the sum of independent random variables, the number of errors tends towards a normal distribution with a mean equal to the expected number of errors. If the true probability of misclassification is $p_0$, then the mean will be $Np_0$ and the standard deviation is $\sqrt{Np_0(1-p_0)}$:

$$\frac{X - Np_0}{\sqrt{Np_0(1-p_0)}} \approx Z$$

where $X$ is the number of misclassified examples and $N$ is the total size of the test set. With this approximation we can estimate a confidence interval for the expected number of errors in the given classifiers, $Np_0$. For a 95% confidence interval:

$$X - 1.96\sigma < Np_0 < X + 1.96\sigma$$

with $\sigma = \sqrt{Np_0(1-p_0)}$, which we can estimate by estimating $p_0 = X/N$. If the intervals computed for two classifiers do not intersect, we can exclude the hypothesis that they have the same expected error rate $p_0$. Applying this to our classifiers, we get the following 95% confidence intervals:

$$X^{LogReg} = 10 \pm 5.4 \qquad X^{kNN} = 6 \pm 3.5 \qquad X^{NB} = 1 \pm 1.9$$

This means that we cannot exclude the hypothesis that the first two classifiers have the same true error, since their intervals intersect. Naïve Bayes seems to be a better classifier than Logistic Regression. However, when X is a very small number, this test is not very reliable. As a rule of thumb, X should be above 5, approximately, for this test to be useful.

An alternative method is *McNemar's test*. Let $e_{01}$ be the number of examples the first classifier misclassifies but the second classifies correctly, and $e_{10}$ be the number of examples the second classifier classifies incorrectly but the first classifier classifies correctly. The difference divided by the total follows approximately a chi-squared distribution with one degree of freedom:

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \approx \chi^2_1$$

The $-1$ term is a continuity correction term because the error counts are discrete and the $\chi^2$ distribution is continuous. If the value is greater than 3.84, we can reject the null hypothesis (that the two classifiers perform identically) with 95% confidence. In our case, the results are:

$$LogReg\ vs\ kNN = 0.8 \qquad kNN\ vs\ NB = 2.3 \qquad NB\ vs\ LogReg = 7.1$$

This means we can conclude there is likely to be a difference between the performance of the Naïve Bayes and the Logistic Regression classifiers, but that the difference is not significant in the other cases.

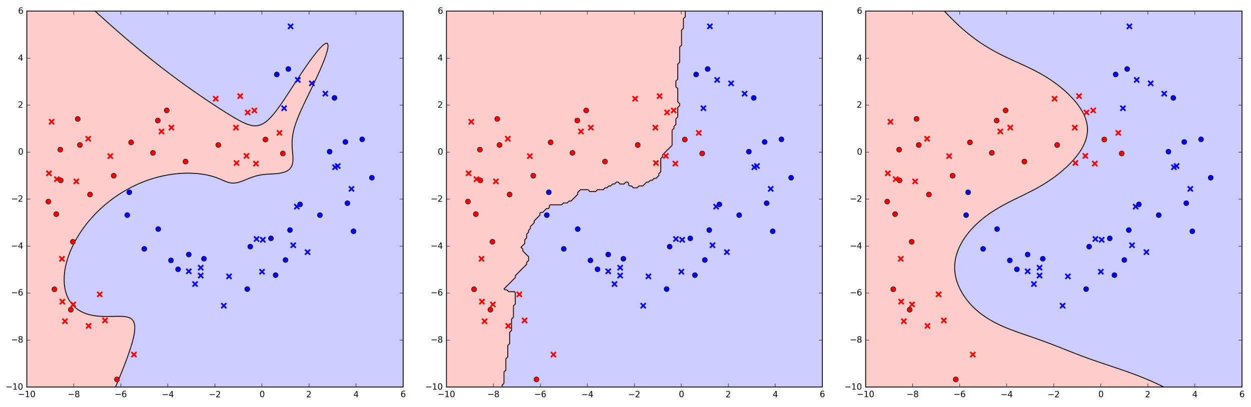

Figure 7.6: Three classifiers: logistic regression, k-NN and Naïve Bayes. All were trained on the set marked as circles and tested on the points marked as crosses.

## 7.8 Further Reading

1. Bishop [4], Section 1.2
2. Alpaydin [2], Section 14.6
3. Mitchell [18], Section 6.9
4. Marsland [17], Section 8.1.2

# Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.