1.3.1 Naive Bayes

In the example of the AIDS test we used the outcomes of the test to infer whether the patient is diseased. In the context of spam filtering the actual text of the e-mail x corresponds to the test and the label y is equivalent to the diagnosis. Recall Bayes Rule (1.15). We could use the latter to infer

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

We may have a good estimate of $p(y)$, that is, the probability of receiving a spam or ham mail. Denote by m_{ham} and m_{spam} the number of ham and spam e-mails in $\mathbf{X}$. In this case we can estimate

$$p(\text{ham}) \approx \frac{m_{\text{ham}}}{m} \text{ and } p(\text{spam}) \approx \frac{m_{\text{spam}}}{m}.$$

The key problem, however, is that we do not know $p(x|y)$ or $p(x)$. We may dispose of the requirement of knowing $p(x)$ by settling for a likelihood ratio

$$L(x) := \frac{p(\text{spam}|x)}{p(\text{ham}|x)} = \frac{p(x|\text{spam})p(\text{spam})}{p(x|\text{ham})p(\text{ham})}. \tag{1.17}$$

Whenever $L(x)$ exceeds a given threshold c we decide that x is spam and consequently reject the e-mail. If c is large then our algorithm is conservative and classifies an email as spam only if $p(\text{spam}|x) \gg p(\text{ham}|x)$. On the other hand, if c is small then the algorithm aggressively classifies emails as spam.

The key obstacle is that we have no access to $p(x|y)$. This is where we make our key approximation. Recall Figure 1.13. In order to model the distribution of the test outcomes T_1 and T_2 we made the assumption that they are conditionally independent of each other given the diagnosis. Analogously, we may now treat the occurrence of each word in a document as a separate test and combine the outcomes in a *naive* fashion by assuming that

$$p(x|y) = \prod_{j=1}^{\#\text{ of words in } x} p(w^j|y), \tag{1.18}$$

where w^j denotes the j-th word in document x. This amounts to the assumption that the probability of occurrence of a word in a document is independent of all other words given the category of the document. Even though this assumption does not hold in general – for instance, the word "York" is much more likely to after the word "New" – it suffices for our purposes (see Figure 1.16).

This assumption reduces the difficulty of knowing $p(x|y)$ to that of estimating the probabilities of occurrence of individual words w. Estimates for

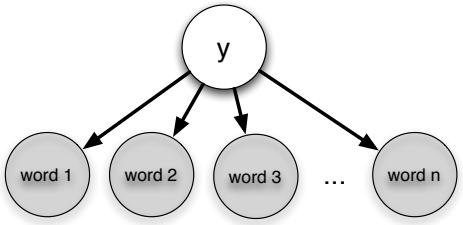

Fig. 1.16. Naive Bayes model. The occurrence of individual words is independent of each other, given the category of the text. For instance, the word *Viagra* is fairly frequent if $y =$ spam but it is considerably less frequent if $y =$ ham, except when considering the mailbox of a Pfizer sales representative.

$p(w|y)$ can be obtained, for instance, by simply counting the frequency occurrence of the word within documents of a given class. That is, we estimate

$$p(w|\text{spam}) \approx \frac{\sum_{i=1}^{m} \sum_{j=1}^{\#\text{ of words in } x_i} \left\{ y_i = \text{spam and } w_i^j = w \right\}}{\sum_{i=1}^{m} \sum_{j=1}^{\#\text{ of words in } x_i} \{ y_i = \text{spam} \}}$$

Here $\left\{ y_i = \text{spam and } w_i^j = w \right\}$ equals 1 if and only if x_i is labeled as spam and w occurs as the j-th word in x_i. The denominator is simply the total number of words in spam documents. Similarly one can compute $p(w|\text{ham})$. In principle we could perform the above summation whenever we see a new document x. This would be terribly inefficient, since each such computation requires a full pass through $\mathbf{X}$ and $\mathbf{Y}$. Instead, we can perform a single pass through $\mathbf{X}$ and $\mathbf{Y}$ and store the resulting statistics as a good estimate of the conditional probabilities. Algorithm 1.1 has details of an implementation. Note that we performed a number of optimizations: Firstly, the normalization by m_{spam}^{-1} and m_{ham}^{-1} respectively is independent of x, hence we incorporate it as a fixed offset. Secondly, since we are computing a product over a large number of factors the numbers might lead to numerical overflow or underflow. This can be addressed by summing over the logarithm of terms rather than computing products. Thirdly, we need to address the issue of estimating $p(w|y)$ for words w which we might not have seen before. One way of dealing with this is to increment all counts by 1. This method is commonly referred to as Laplace smoothing. We will encounter a theoretical justification for this heuristic in Section 2.3.

This simple algorithm is known to perform surprisingly well, and variants of it can be found in most modern spam filters. It amounts to what is commonly known as "Bayesian spam filtering". Obviously, we may apply it to problems other than document categorization, too.

Algorithm 1.1 Naive Bayes

Train($\mathbf{X}, \mathbf{Y}$) {reads documents $\mathbf{X}$ and labels $\mathbf{Y}$}
  Compute dictionary D of $\mathbf{X}$ with n words.
  Compute m, m_{ham} and m_{spam}.
  Initialize $b := \log c + \log m_{\text{ham}} - \log m_{\text{spam}}$ to offset the rejection threshold
  Initialize $p \in \mathbb{R}^{2 \times n}$ with $p_{ij} = 1$, $w_{\text{spam}} = n$, $w_{\text{ham}} = n$.
  {Count occurrence of each word}
  {Here x_i^j denotes the number of times word j occurs in document x_i}
  for $i = 1$ to m **do**
    if $y_i = \text{spam}$ **then**
      for $j = 1$ to n **do**
        $p_{0,j} \leftarrow p_{0,j} + x_i^j$
        $w_{\text{spam}} \leftarrow w_{\text{spam}} + x_i^j$
      end for
    else
      for $j = 1$ to n **do**
        $p_{1,j} \leftarrow p_{1,j} + x_i^j$
        $w_{\text{ham}} \leftarrow w_{\text{ham}} + x_i^j$
      end for
    end if
  end for
  {Normalize counts to yield word probabilities}
  for $j = 1$ to n **do**
    $p_{0,j} \leftarrow p_{0,j} / w_{\text{spam}}$
    $p_{1,j} \leftarrow p_{1,j} / w_{\text{ham}}$
  end for
Classify(x) {classifies document x}
  Initialize score threshold $t = -b$
  for $j = 1$ to n **do**
    $t \leftarrow t + x^j (\log p_{0,j} - \log p_{1,j})$
  end for
  if $t > 0$ **return** spam **else return** ham

1.3.2 Nearest Neighbor Estimators

An even simpler estimator than Naive Bayes is nearest neighbors. In its most basic form it assigns the label of its nearest neighbor to an observation x (see Figure 1.17). Hence, all we need to implement it is a distance measure $d(x, x')$ between pairs of observations. Note that this distance need not even be symmetric. This means that nearest neighbor classifiers can be extremely

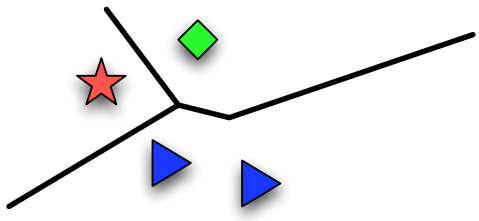

Fig. 1.17. 1 nearest neighbor classifier. Depending on whether the query point x is closest to the star, diamond or triangles, it uses one of the three labels for it.

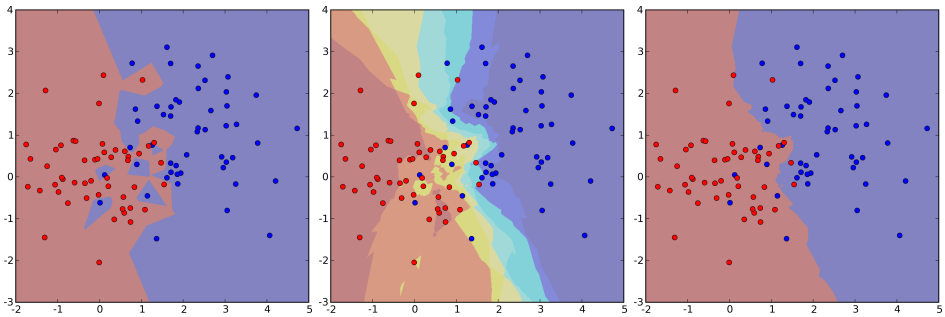

Fig. 1.18. k-Nearest neighbor classifiers using Euclidean distances. Left: decision boundaries obtained from a 1-nearest neighbor classifier. Middle: color-coded sets of where the number of red / blue points ranges between 7 and 0. Right: decision boundary determining where the blue or red dots are in the majority.

flexible. For instance, we could use string edit distances to compare two documents or information theory based measures.

However, the problem with nearest neighbor classification is that the estimates can be very noisy whenever the data itself is very noisy. For instance, if a spam email is erroneously labeled as nonspam then all emails which are similar to this email will share the same fate. See Figure 1.18 for an example. In this case it is beneficial to pool together a number of neighbors, say the k-nearest neighbors of x and use a majority vote to decide the class membership of x. Algorithm 1.2 has a description of the algorithm. Note that nearest neighbor algorithms can yield excellent performance when used with a good distance measure. For instance, the technology underlying the Netflix progress prize [BK07] was essentially nearest neighbours based.

Note that it is trivial to extend the algorithm to regression. All we need to change in Algorithm 1.2 is to return the average of the values y_i instead of their majority vote. Figure 1.19 has an example.

Note that the distance computation $d(x_i, x)$ for all observations can be-

Algorithm 1.2 k-Nearest Neighbor Classification

Classify($\mathbf{X}, \mathbf{Y}, x$) {reads documents $\mathbf{X}$, labels $\mathbf{Y}$ and query x}
  for $i = 1$ **to** m **do**
    Compute distance $d(x_i, x)$
  end for
  Compute set I containing indices for the k smallest distances $d(x_i, x)$.
  return majority label of $\{y_i$ where $i \in I\}$.

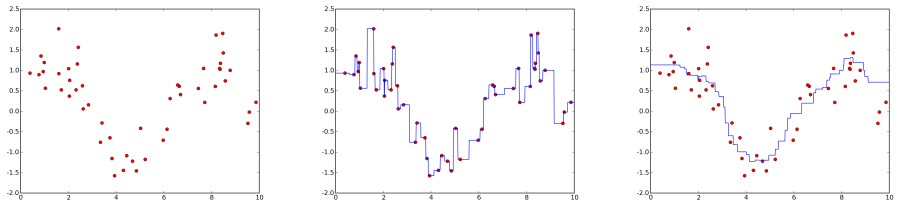

Fig. 1.19. k-Nearest neighbor regression estimator using Euclidean distances. Left: some points (x, y) drawn from a joint distribution. Middle: 1-nearest neighbour classifier. Right: 7-nearest neighbour classifier. Note that the regression estimate is much more smooth.

come extremely costly, in particular whenever the number of observations is large or whenever the observations x_i live in a very high dimensional space.

Random projections are a technique that can alleviate the high computational cost of Nearest Neighbor classifiers. A celebrated lemma by Johnson and Lindenstrauss [DG03] asserts that a set of m points in high dimensional Euclidean space can be projected into a $O(\log m/\epsilon^2)$ dimensional Euclidean space such that the distance between any two points changes only by a factor of $(1 \pm \epsilon)$. Since Euclidean distances are preserved, running the Nearest Neighbor classifier on this mapped data yields the same results but at a lower computational cost [GIM99].

The surprising fact is that the projection relies on a simple randomized algorithm: to obtain a d-dimensional representation of n-dimensional random observations we pick a matrix $R \in \mathbb{R}^{d \times n}$ where each element is drawn independently from a normal distribution with $n^{-\frac{1}{2}}$ variance and zero mean. Multiplying x with this projection matrix can be shown to achieve this property with high probability. For details see [DG03].