Using log-probabilities for Naive Bayes

Recall that a Naive Bayes classifier (NBC) is set up as follows.

We have a number of hypotheses (or classes), $H_1, \ldots, H_n$.

We have a set of features, $F_1, \ldots, F_m$.

For the spam classification task, we have two hypotheses, spam and not-spam, and m words in our vocabulary, F_1 through F_m.

During the training phase, the NBC estimates the prior probabilities of each hypothesis and the probability of each word given each hypothesis:

$$P(\text{spam}) = \frac{\text{\# of emails labeled as spam}}{\text{total \# of emails}}$$

(and similarly for not-spam.)

The NBC also estimates the probability of each word given each hypothesis:

$$P(F_j \mid H_i) = \frac{\text{\# of } H_i \text{ emails with word } F_j + 1}{\text{total \# of } H_i \text{ emails} + 2}$$

(and similarly for $\neg F_j$, which means the absence of word F_j.)

Naive Bayes then uses the MAP hypothesis to predict a class:

$$\begin{aligned}
H^{\text{MAP}} &= \operatorname*{argmax}_i P(D \mid H_i) P(H_i) \\
&= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} P(F_1, \ldots, F_m \mid H_i) P(H_i) \\
&= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \Big[P(F_1 \mid H_i) \cdots P(F_m \mid H_i) \Big] P(H_i) \\
&= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \Big[\prod_{j=1}^{m} P(F_j \mid H_i) \Big] P(H_i)
\end{aligned}$$

When you implement this calculation in code, a problem arises when m (the size of the vocabulary) gets very big. Notice that because each $P(F_j \mid H_i)$ term is between 0 and 1, when you start multiplying them together, the overall product starts approaching zero. This is exacerbated by each new feature that is added: with 1,000 features, multiplying 1,000 probabilities together quickly results in an extremely tiny number, one which often cannot be represented in a `double` or `long double`. This issue is not limited to C++; every language which uses a standard floating-point data type with a fixed number of bits will eventually run out of precision with extremely tiny numbers.

A common work-around for this situation is to represent probabilities as *log-probabilities* instead. That is, instead of using $P(F_j \mid H_i)$ and $P(H_i)$, we we use $\ln P(F_j \mid H_i)$ and $\ln P(H_i)$ (ln means the natural logarithm, though any logarithm works). This transformation works (and is rather elegant) for a number of reasons:

- Because a probability is always between 0 and 1, a log-probability is always between $-\infty$ and 0 (a much wider range of numbers than between 0 and 1).
- If $a < b$, then $\ln a < \ln b$. This property of logarithms means that comparing probabilities to figure out which one is the biggest (as argmax does) is equivalent to comparing the log-probabilities.
- Finally, the math works out nicely because the logarithm of a product is the sum of the logarithms (that is, $\ln(a \cdot b) = \ln a + \ln b$):

$$\begin{aligned} H^{\text{MAP}} &= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \Big[\prod_{j=1}^{m} P(F_j \mid H_i)\Big] P(H_i) \\ &= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \ln\left(\Big[\prod_{j=1}^{m} P(F_j \mid H_i)\Big] P(H_i)\right) \\ &= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \ln\Big[\prod_{j=1}^{m} P(F_j \mid H_i)\Big] + \ln P(H_i) \\ &= \operatorname*{argmax}_{i \in \{\text{spam,not-spam}\}} \Big[\sum_{j=1}^{m} \ln P(F_j \mid H_i)\Big] + \ln P(H_i) \end{aligned}$$

- The resulting sums (rather than products) are usually manageable numbers, unlike than the very tiny probabilities that result without this transformation.

Note that because we are taking the logarithm of numbers between 0 and 1, the log-probabilities will always be *negative*. This is OK. Just do your calculations and comparisons as you would normally: a "more negative" number (meaning a negative number greater in absolute value) corresponds to a smaller probability than a "less negative" number, so you don't have to switch the comparison.

Example: say you have the probabilities 0.2 and 0.3. Note that $\ln(0.2) \approx -1.6$ and $\ln(0.3) \approx -1.2$. The comparisons are preserved: $0.2 < 0.3$ and $-1.6 < -1.2$.