Meta-Models & Naïve Bayes

Alex Sverdlov
alex@theparticle.com

1 Introduction

A model may also be constructed from a set of other models, where the output category is determined by an aggregate or majority vote of constituent base models (those that are not an ensemble of other models).

Often this collective makes much better decisions than individual models. The basic idea is that each sub-model is different from the rest: if they were all the same, they would all produce the same result. This difference often comes about by training each sub-model on a different sub-sample of the training data.

When building such ensembles we have a choice of what base classifiers to use, and how to split the training among them. There are two popular mechanisms of constructing meta-models: bagging and boosting.

Bagging creates multiple training sets by randomly sampling the original training data. There are several versions of this, sampling with replacement and without. For practical reasons (big data, distributed across multiple machines), the most common application is to ensure data is sufficiently randomized (shuffled), then just cut the training dataset into N non-overlapping partitions. The N classifiers (trained on N respective datasets) become the ensemble classifier. This allows for parallel training of the N models, as there are no dependencies between base classifiers.

Boosting is essentially sequential bagging, where each step creates a classifier, based on data that has been re-weighted by previous steps. The misclassified instances are given higher weight, so future iterations work harder to get those instances correct.

An alternate view is to sample features, instead of training instances. A sort of bagging of features, where each base classifier gets a subset (perhaps a random subset) of features to train on. This is often called the sub-space method, or feature bagging. Conceivably any base classifier can be used on such sub-sets of features (just as with ordinary bagging and boosting), but having direct control of the number of input features allows us to choose the simplest of base "classifiers": a table lookup.

The question of how to combine the results of such ensembles leads us to explore why such ensembles often work in the first place, and the key to that is naïve Bayes.

Thomas Bayes (1702-1761) introduced a new form of statistical reasoning—the inversion

of probabilities. We can view it as

$$Posterior = Likelihood \times Prior$$

where *Posterior* is the probability that the *hypothesis* is true given the evidence. *Prior* is the probability that the hypothesis was true *before* the evidence (an assumption). *Likelihood* is the probability of obtaining the observed evidence given that the hypothesis is true.

The Bayes rule codifies exactly what happens when evidence (or input data) is observed. Before invoking our classifier, there is a certain probability of seeing each category: $P(category)$. After seeing the input instance, let us call it $evidence_1$, we wish to determine the category given the input instance data. The Bayes rule says the answer is:

$$P(category|evidence_1) = \frac{P(evidence_1|category)P(category)}{P(evidence_1)}$$

Notice that both $P(category)$ and $P(evidence_1|category)$ can be estimated from training data (just count the conditionals), and the denominator $P(evidence_1)$ is only there to ensure probabilities sum to 1.

Suppose we next observe $evidence_2$ and $evidence_3$, etc. Under certain assumptions, we can just continue to chain the Bayes rule and refine the probability of category given all these pieces of evidence

$$P(category|evidence_2) = \frac{P(evidence_2|category)P(category|evidence_1)}{P(evidence_2)}$$

or more generally, making conditional independence assumption:

$$P(category|evidence_1, \ldots, evidence_N) = P(category) \prod_i^N \frac{P(evidence_i|category)}{P(evidence_i)}$$

See Section 2 for the derivation that makes this possible. The certain assumptions mentioned are the conditional independence of evidence given category. If all the evidence is highly dependent, then this sort of chaining will produce bad results. If evidence is *mostly* (but not perfectly) independent, then the results of such chaining are generally good—each piece of evidence moves us in the right direction, so the more evidence we observe the better our prediction of the category. This is actually the idea behind Naïve Bayes classifier.

The above suggests that to combine multiple ensemble results we should convert the result of each base classifier into a probability, and then multiply them together for the final answer (apply Naïve Bayes). Alternatively, to avoid floating point precision issues associated with multiplying lots of small probabilities, we can aggregate logs[1]. Often the exact probabilities are not important, just their relative scale—at which point simply aggregating the results of constituent classifiers often works best, and is actually optimal [KHDM98], keeping in mind the same assumptions as for Naïve Bayes.

[1] $a \times b = \exp^{\log a + \log b}$

2 Naïve Bayes Classifier

We can use the Bayes rule to construct a classifier. The input features $x_1, \ldots, x_n$ and target class C form a joint probability distribution $P(x_1, \ldots, x_n, C)$.

Training might involve building a table to estimate $P(x_1, \ldots, x_n|C)$, and classification would just be:

$$P(C|x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n|C)P(C)}{P(x_1, \ldots, x_n)}$$

The problem with the above is the large joint probability. It is not practical: for large n, we cannot store n dimensional arrays to do aggregates, nor will we have enough data to populate such large tables with meaningful counts.

Using the chain-rule to rewrite $P(x_1, \ldots, x_n, C)$ we get:

$$P(x_1, \ldots, x_n, C) = P(x_1|x_2, \ldots, x_n, C) \times P(x_2|x_3, \ldots, x_n, C) \times \cdots \times P(x_n|C) \times P(C)$$

If we assume that all x_i are only dependent on C (and not on other x_j), by chain-rule the above becomes:

$$P(x_1, \ldots, x_n, C) = P(x_1|C) \times P(x_2|C) \times \cdots \times P(x_n|C) \times P(C)$$

or

$$P(x_1, \ldots, x_n, C) = P(C) \prod_{i=1}^{n} P(x_i|C)$$

Since $x_1, \ldots, x_n$ is the observation, we can treat $P(x_1, \ldots, x_n)$ as constant. This means that

$$P(C|x_1, \ldots, x_n) = K \times P(C) \prod_{i=1}^{n} P(x_i|C)$$

where K is a place holder for a constant $1/P(x_1, \ldots, x_n)$. This is the constant that makes probabilities sum to 1, and we do not actually need it for a classifier.

The naïve Bayes classifier is just:

$$y = \operatorname*{argmax}_{c \in C} P(c) \prod_{i=1}^{n} P(x_i|c)$$

or to avoid floating point precision problems:

$$y = \operatorname*{argmax}_{c \in C} \log P(c) + \sum_{i=1}^{n} \log P(x_i|c)$$

Notice that in this formulation, training is only required to estimate $P(x_i|C)$ and $P(C)$ which is trivial to do from training data.

A wonderful paper on why this often works is: "Idiot's Bayes - not so stupid after all?" by Hand & Yu [HY01]. A very notable application of this approach is found in almost all spam filters [SDHH98].

References

[HY01] David J. Hand and Keming Yu. Idiot's bayes: Not so stupid after all? *International Statistical Review / Revue Internationale de Statistique*, 69(3):385–398, 2001.

[KHDM98] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(3):226–239, March 1998.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.