Naïve Bayes

Machine Learning II

November 30, 2020

1 Description

The Naïve Bayes (NB) method is a multiclass classifier that uses the probability of observing predictor values given an outcome to estimate the probabilities of observing $Y = i$, given a set of predictor values.
This classifier doesn't use Bayesian methods, despite its name. Its name comes from the *Bayes rule*-like calculation of inferring the predictions. This formula refers to the conditional probability of our hypothesis H given our evidence e. [1]

$$P(H|e) = \frac{P(e|H) \cdot P(H)}{P(e)} \tag{1}$$

Breaking down this formula into its parts we define 4 important concepts for this algorithm:

- $P(H|e)$ is our **posterior**: how probable is H given e?
- $P(e|H)$ is our **likelihood**: how probable is e given H?
- $P(H)$ is our **prior**: how probable is H before knowing about e?
- $P(e)$ is our **marginal**: how probable is e under all possible hypotheses?

In `python` we can use this family of classifiers by importing `sklearn.naive_bayes`

```
import sklearn.naive_bayes
```

2 Key concepts

- Conditional probability $P(A|B)$: probability of the event A happening given that event B has happened.

  It's the relation between the probability of both events happening $P(A \cap B)$ and the probability of only B happening $P(B)$.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2}$$

- Posterior probability $P(H|e)$: in our case, our hypothesis H is the class to which the observation belongs to given the set of evidences e. In a more hands-on approach, H is each individual category y_i we are trying to predict and e is the set of variables x_i with which we are training our algorithm.

3 How it works?

NB is based on the Bayes rule. By adapting the Bayes formula (1), we get the following:

$$P(k_j|x_1, ..., x_n) = \frac{P(x_1, ..., x_n|k_j)P(k_j)}{P(x_1, ..., x_n)} \tag{3}$$

It's called *naïve* because we won't be using the Bayes rule as we described in (1), we'll rather **assume** that our x_i are independent and therefore the likelihood can be simplified as follows:

$$P(x_1, ..., x_n|k_j) \approx P(x_1|k_j)P(x_2|k_j)...P(x_n|k_j) = \prod_{i=1}^{i=n} P(x_i|k_j) \tag{4}$$

Our initial formula (3) can therefore be restructured using (4) as follows:

$$P(k_j|x_1, ..., x_n) = \frac{P(k_j)\prod_{i=1}^{n} P(x_i|k_j)}{P(x_1, ..., x_n)} \tag{5}$$

For example, let's assume our algorithm needs to classify observations according to three (3) classes: $\{k_1, k_2, k_3\}$. Since we have 3 classes, we'll apply our formula 3 times with the same dataset to see which class receives the highest probability, and we'll assign that class accordingly. This is called *Maximum a posteriori* (MAP) propability.

$$MAP = max\{P(k_1|x_1, ..., x_n), P(k_2|x_1, ..., x_n), P(k_3|x_1, ..., x_n)\} \tag{6}$$

$$MAP = max\{\frac{P(k_1)\prod_{i=1}^{n} P(x_i|k_1)}{P(x_1, ..., x_n)}, \frac{P(k_2)\prod_{i=1}^{n} P(x_i|k_2)}{P(x_1, ..., x_n)}, \frac{P(k_3)\prod_{i=1}^{n} P(x_i|k_3)}{P(x_1, ..., x_n)}\} \tag{7}$$

$$MAP = max\{P(k_1)\prod_{i=1}^{n} P(x_i|k_1), P(k_2)\prod_{i=1}^{n} P(x_i|k_2), P(k_3)\prod_{i=1}^{n} P(x_i|k_3)\} \tag{8}$$

$$MAP = max_j\{P(k_j)\prod_{i=1}^{n} P(x_i|k_j)\} \tag{9}$$

4 What could go wrong?

When classifying text we could find words (x_i) in the test set that weren't present in the training set, or that were present with a very low count. This will cause very low or null products when calculating $P(x_i|k_j)$. In order to avoid this we can use this options:

- Add-one approach (from (2)):

$$P(x_i|k_j) = \frac{P(x_i \cap k_j)}{P(k_j)} = \frac{count(x_i, k_j)}{\sum count(x_i, k_j)} \approx \frac{count(x_i, k_j) + 1}{\sum_i count(x_i, k_j)} \tag{10}$$

- Laplace smoothing: calculating log-odds instead in MAP (9). By using logs, we convert products into sums and therefore very low values won't drag down the total value of probability.

$$log(n \cdot m) = log(n) + log(m) \tag{11}$$

$$MAP_{smooth} = max_j\{log(P(k_j)) \sum_{i=1}^{n} log(P(x_i|k_j))\} \tag{12}$$

5 When to use it?

- As initial algorithm, just to be included in your pipelines to produce initial predictions and testing the pipelines
- When in need of something fast, reliable and robust to irrelevant features
- If the assumption we did in (4) works this will produce very good results. If not, it's a good starting point from which to use more complex approaches.

6 Summary

Type	Classification
Tolerates many features?	Weak
Parametrization	No
Memory needed	Small
Data requirements	Small
Overfitting	Low
Difficulty	Weak
Training time	Weak
Prediction time	Weak
Interpretability	Good

References

[1] P Bruce and P Bruce. *Practical Statistics for Data Scientists. 50 essential concepts.* O'Reilly, 2017.