BAYES DECISION CLASSIFIER

A PREPRINT

Enrico Cecchetti
Department of Computer Science
Politecnico di Torino
PoliTO, Torino

February 11, 2019

ABSTRACT

In machine learning, naive Bayes classifiers are a family of simple *probabilistic classifiers* based on applying **Bayes Theorem** with strong (naive) independence assumptions between the features.

1 Bayes Rule (or Theorem) and Bayes Decision

Bayes Theorem is stated mathematically as the following equation:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Where A and B are events and $P(B) \neq 0$

- P(A|B) represent the probability of A over B, or better, the likelihood of event A considering B is true
- P(A) or P(B) probability of A and B indipendently each other

So P(A|B) or P(B|A) probabilities are also called conditional probability and they are describing the probability of A under the condition of B.
So lets see that on an example: Lets say that that A stands for the probability, that if you look outside your house "you will see at least one person". Lets assume that B stands for "it is raining".
Then P(A|B) stands for the probability that you will see at least one person outside your house if it is raining.

An interesting example to understand is the Drug Test Sample:
Suppose that a test for using a particular drug is 99% sensitive and 99% specific. That is, the test will produce 99% true positive results for drug users and 99% true negative results for non-drug users. Suppose that 0.5% of people are users of the drug. What is the probability that a randomly selected individual with a positive test is a drug user?

$$\begin{aligned} P(\text{User} \mid +) &= \frac{P(+ \mid \text{User}) P(\text{User})}{P(+)} \\ &= \frac{P(+ \mid \text{User}) P(\text{User})}{P(+ \mid \text{User}) P(\text{User}) + P(+ \mid \text{Non-user}) P(\text{Non-user})} \\ &= \frac{0.99 \times 0.005}{0.99 \times 0.005 + 0.01 \times 0.995} \\ &\approx 33.2\% \end{aligned}$$

So given the Bayes Decision Rule, if we have a classifier for which:
If $P(w1|X) > P(w2|X)$ then $w1 = true$
If $P(w1|X) < P(w2|X)$ then $w2 = true$

Therefore we can understand that the probability of error is
If P(error|X) = P(w1|X) if we decide w2
If P(error|X) = P(w2|X) if we decide w1

That means that with Bayes we are choosing the outcome that *maximize the probability of Success* and also that *minimize the probabilty of errors*.

The conclusion is that the Classifier will choose:
w1 if P(w1 | X) > P(w2 | X) otherwise w2.

That means that the Bayer Decision will be taken from this statement:
P(error | X) = min[P(w1 | X), P(w2 | X)]

The Bayesian way then rather that estimating a single event look for the whole distribution of the possible values. Which means, that we will need a lot of trials(experiments) to have a solid understanding of the event.
A simple dimostration is the Flip-Coin event. We know from theory that the probability of that event is 50%T and 50%H, but if we flip the coins 3 time we may obtain something like 3H 2T.
Does it means that Head probability is larger that the Tails one?!
Obviously not... we only need so much experiment to caliber our classifier in the right way.

2 Gaussian Naive Bayes Classifier

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.
This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.
Tipically, we calculate the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.
This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

3 Gaussian NB from Scratch

The Gaussian NB is not a difficult algorithm so it could be summarized in few steps:

- Dataset configuration
- Split Dataset onto training and test data
- Make predictions
- Check accuracy

3.1 Dataset configuration

For this step I used a dataset predisposed and given from our university containing image from 4 different categories: dog, house, guitar and person.
The function used for load those is the following:

```
from PIL import Image
import numpy as np
import glob
def load_data_into_dataset(matrix, lable_array, id_class, url):
    # index to return
    offset = 0
    for filename in glob.glob(url):
        offset += 1
        img_data = np.asarray(Image.open(filename))
        # just store the whole matrix obtained by the 3-D to 1-D trasformation
        matrix.append(img_data.ravel())
        lable_array.append(id_class)
    return offset
```

It takes as parameter "Matrix" which is the dataset matrix we are building, then the lable_array that contains the class_lable associated to every Image (so dog if the image is a dog ecc...).

3.2 Split Dataset onto training and test data

Next we need to split the data into a training dataset that Naive Bayes can use to make predictions and a test dataset that we can use to evaluate the accuracy of the model. We need to split the data set randomly into train and datasets with a ratio of 67% train and 33% test (this is a common ratio for testing an algorithm on a dataset).

For doing that we used the function of sklearn.train_test_split() as:

```
    from sklearn.model_selection import train_test_split
[...]
X_train, X_test, y_train, y_test = train_test_split(X_matrix,
                                                    Y_matrix,
                                                    test_size=slice_size,
                                                    random_state=seed)
```

3.3 Make predictions

Now its time to make predictions using the summaries prepared from our training data.
Making predictions as we mentioned before involves calculating the probability that a given data instance belongs to each class, then selecting the class with the largest probability as the prediction.
We can use a Gaussian function to estimate the probability of a given attribute value, given the known mean and standard deviation for the attribute estimated from the training data.
For this part we used:

```
from sklearn.naive_bayes import GaussianNB
# Instantiate the classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
```

In this portion we create a Gaussian NB classifier from the sklearn.NV lib and we fit that by means of our training data (X_training and their own y_lable). So, we are ready to make the predition based on gaussian probability.

3.4 Check accuracy

For testing the accuracy we will make prediction on the whole test-dataset, which result we just know.
Then for each one of the value we check the result with the associated class_lable that we own stored in y_test.

The accuracy will be given by this formula:

For doing that we used the function of sklearn.train_test_split() as:

```
((len(y_test) - (y_test != y_pred).sum()) * 100 / (len(y_test))
```

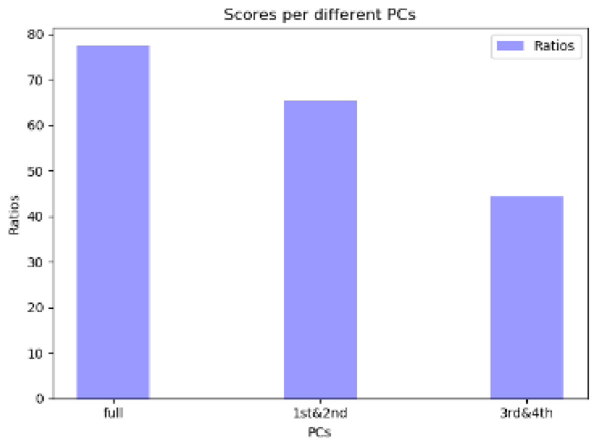

Figure 1: Result of a Bayes classifier Execution of the example Dataset.

4 Uses and problem

In this section we pro and cons of this Classifier.
The main limitation are:

- The main disadvantage is the **number of data** dependency<br/> To have a good model we need a lot of data sample. Remember the example of the coin showed before in case of doubt.
- The second disadvantage is that the Naive Bayes classifier makes a very strong **assumption on the shape of your data distribution**, i.e. any two features are independent given the output class. Due to this, the result can be (potentially) very bad - hence. In our case is recognising a dog instead of a person... but if you think of a simil algorithm used for a robot AI it could bring some big trouble
- Another problem happens due to **data scarcity**. For any possible value of a feature, you need to estimate a likelihood value by a frequentist approach. This can result in probabilities going towards 0 or 1, which in turn leads to numerical instabilities and worse results. In this case, you need to smooth in some way your probabilities (e.g. as in sklearn[1]), or to impose some prior on your data, however you may argue that the resulting classifier is not naive anymore.
- A third problem arises for continuous features. It is common to use a binning procedure to make them discrete, but if you are not careful you can throw away a lot of information.

Is a famous paper dealing with some additional problems of Naive Bayes, e.g. what happens if your classes are imbalanced (resulting in skewed probabilities). Basically, NB offers absurd simplicity in exchange for many possible drawbacks, making it viable in general only as a cheap baseline - this is why it was called "the favorite punching bag" of classifiers

References

[1] Jason Brownlee, PhD How To Implement Naive Bayes From Scratch in Python In *Code Machine Learning Algorithms From Scratch - December 8, 2014* https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

[2] Shikit Learn Naive Bayes https://scikit-learn.org/stable/modules/naive_bayes.html

[3] Barbara Caputo Class slides

[4] Wikipedia Bayes' theorem Fast classification of handwritten on-line arabic characters. https://en.wikipedia.org/wiki/Bayes%27_theorem