UASimpleClassifier: An Implementation of a Naive Bayes Classifier

Justin A. Dang
Computer and Information Sciences Department
University of Arkansas – Fort Smith

February 24, 2023

Abstract

The job of a classification algorithm is to determine a category for some set given some data. The naive Bayes' classifier (NBC) is one such classification algorithm. It works by using Bayes' Theorem under the assumption of conditional independence. Implementation of NBC in code involves building a model, which are data structures that can store certain conditional probabilities to be used during classification. Quality and quantity of training determine the accuracy and precision of the model.

1 Introduction

1.1 Classification

One of the fundamental ideas in artificial intelligence is the process of classification. Humans classify things on a daily basis throughout their entire lives on many different levels. Take for example how a college student may have a list of homework assignments they must complete. The student may classify individual homework assignments according to the courses that it belongs to. Although this might seem trivial to consider, the student (whether knowingly or unknowingly) uses certain observations in order to put their homework into course categories. These observations might be a course number written on the page, or the content questions of the assignment, or even what teacher they remembered assigned them the certain paper. This simple ability to classify things makes sure he/she doesn't end up turning in a paper on the Civil War into their Calculus I lecture.

Simply put, classification is the process of systematically categorizing a given set of input data based on one or more variables called features. As mentioned earlier, this is modeled by a great range of complexity by humans. For computers, their classification abilities are rooted in mathematics and statistics rather than intuition and sense like humans. Say that there is a set of distinct classes defined by $\hat{c} = \{c_1, c_2, c_3\}$. A computer will use some classification algorithm and some set of features $x = \{x_1, x_2, x_3, ...\}$ to determine what class in $\hat{c}$ is most likely for a given set x. This relationship would be:

$$\hat{c} = f(x)$$

To facilitate some function f, a probabilistic classification method like Naive Bayes may be used.

1.2 Naive Bayes

Bayes' theorem has been studied for hundreds of years in the field of statistics. The theorem is able to determine the conditional probability of some event occurring based on similar previous outcomes. Mathematically stated as $P(A|B)$, or the probability of event A given event B (and $P(B) \neq 0$), the following equation determines its conditional probability:

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

This concept is foundational to building this classifier algorithm. A **naive Bayes classifier** uses the theorem under the assumption that the features it computes are *conditionally independent*. When Bayes theorem is applied, conditional probabilities of each feature are calculated and then aggregated to divide against the probability of the feature appearing. This classifier model is called naive because it assumes the following for a set of classes C_t and features x_i:

$$P(x_i|C_t, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|C_t)$$

Due to this constraint, the naive Bayes classifier model can determine this rule:

$$\hat{c} = \underset{j \in i...k}{argmax} P(C_j) \prod_{i=1}^{n} P(x_i|C_j)$$

The programmatic implementation will dive into depth on how and why this rules works.

2 Background

Despite the word "naive" being a rather reductive or even condescending term in the English vernacular, the naive Bayes classifier (NBC) is far from insignificant in modern AI research. NBC can be utilized rather effectively if the classifier assumptions can be made across the set of data being analyzed. Here are just a few examples and summaries of how they have been recently used.

2.1 Chemical Named Entity Recognition and NBC

The Journal of Cheminformatics published in late 2022 an implementation of NBC in chemical naming applications[1]. Chemical Named Entity Recognition (CNER) is a technique that uses algorithms to extract chemical compound identifiers from texts. It then takes the names and creates associations with physical-chemical properties and biological use cases. It tested this new method against Mpro inhibitors for coronavirus 2 with successful identification. Taking in a large collection of abstracts and chemical libraries for feature input, the algorithm is able to classify entities as belonging to one of several types like formulas, abbreviations, systematic, and other non-CNE entities. It uses these points to build a full IUPAC chemical name for lab and practice use. This proposed method showed promising results in terms of filtering out non-CNE-related entities.

2.2 Author Detection in Tweets with NBC

Journal of Intelligent & Fuzzy Systems showed researchers applying NBC into a bigger model for determining specific authorship of tweets[2]. The classes are certain Twitter accounts. The features came in the form of words in a big list of tweets. Many other variables and values were accounted for (sensitivity, specificity, etc) to be more comprehensive. This work in NLP uses naive Bayes as the backbone of the training and testing.

2.3 Advantages to Naive Bayes

Although a variety of other techniques are available, the naive Bayes classifier remains a valid options for uses cases that can ask for simplicity, scalability, or conform to the constraints of an NBC model.

3 Specification

For naive Bayes' conditional probabilities to work, it relies on the outcomes of similar events that have already happened. The classification process takes three main phases.

1. Training phase scans training data and builds a model based on the discrete occurrences and continuous distributions of feature data for all classes.
2. Testing phase scans a separate but similar data set in order to accept all features. It will use the naive Bayes' classifier in order to determine which class those features are more likely to represent and check it against the data set.
3. Predication phase will take in any single set of complete features on a set number of entries. This will determine a class likelihood without checking against a given data class.

This algorithm must store conditional probabilities for use in the main classifier formula. For all discrete data, the following must be found and stored:

$$P(x_i|C_j)$$

and for continuous values, a mean and standard deviation must be obtained for the probability density function. The testing phase implements this formula based on the naive Bayes theorem to classify a given data set:

$$\hat{c} = \underset{j \in i \ldots k}{argmax} \frac{P(C_j)\prod_{i=1}^{n} P(f_i|C_j)}{\sum_k^{|C|} Pr(C_k)\prod_{i=1}^{n} P(f_i|C_k)}$$

A features X that follows some normal distribution $X\ (\mu, \sigma^2)$ will have a probability distribution function of:

$$PDF(X) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

4 Implementation

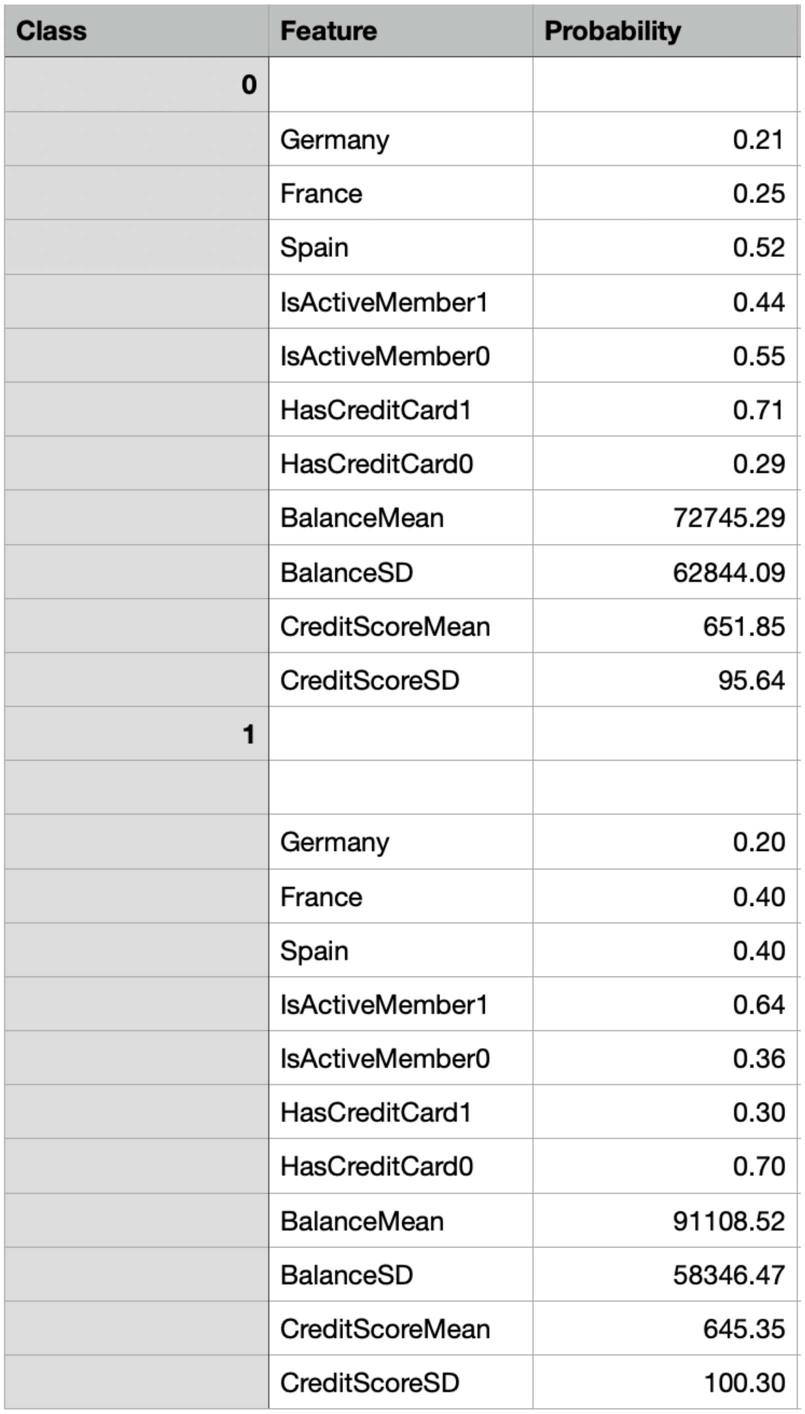
This classifier algorithm was written in Java using standard libraries. Three core methods exist in order to correspond with the 3 phases.

- `void train(String filename)` accepts training data and determines the probability of all features given a class. If the data is discrete, then it simply counts the occurrences of that feature given that class. If the data is continuous, the mean and standard deviation are stored.
- `void test(String filename` - accepts testing data. For each record in the data, the features are collected and put through `classify(f1,f2,f3,f4,f5)` to predict a class. It then computes the accuracy by comparing the realized class in the testing data.
- `String classify(f1,f2,f3,f4,f5)` - accepts five features and uses the Naive Bayes classifier formula in order to predict which class is the most likely.

Storing the conditional probabilities correctly is going to be crucial to the accuracy of the model. To accomplish, a large hash table thanks to its hash lookup times.

```
HashMap<String, HashMap<String, Double>> NBModel = new HashMap<>();
```

The top-level table stores distinct classes as keys while the inner tables store feature names and the discrete representation as they appear in the data. For continuous values, the feature name and appropriate parameters are used as keys.



Class	Feature	Probability
0		
	Germany	0.21
	France	0.25
	Spain	0.52
	IsActiveMember1	0.44
	IsActiveMember0	0.55
	HasCreditCard1	0.71
	HasCreditCard0	0.29
	BalanceMean	72745.29
	BalanceSD	62844.09
	CreditScoreMean	651.85
	CreditScoreSD	95.64
1		
	Germany	0.20
	France	0.40
	Spain	0.40
	IsActiveMember1	0.64
	IsActiveMember0	0.36
	HasCreditCard1	0.30
	HasCreditCard0	0.70
	BalanceMean	91108.52
	BalanceSD	58346.47
	CreditScoreMean	645.35
	CreditScoreSD	100.30

A few standout functions are of note here: to calculate the frequency in a set of data, this frequency counting method is used.

```java
public double freqCount(String value, ArrayList<String> fullSet) {
        return Collections.frequency(fullSet, value);
    }
```

When computing distinct values given a class, the `ArrayList<String> fullSet` parameter is a set of some class in which the frequency of one class is to be determined.

The `classify()` method contains class determination making based off running the naive Bayes classifier on all possible classes.

```java
return argMax0 > argMax1 ? "0" : "1";
```

With two possible classes, it simply picks the higher probability.

To build the structure of the `NBModel`, a reduced set is made using the amount of distinct discrete possibilities in the data set. This reduced set is referenced to look for probabilities given class and its data structure is the skeleton of the `NBModel`.

```java
public ArrayList<ArrayList<String>> reduceDataSet(ArrayList<ArrayList<String>>
     fullData) {

    ArrayList<ArrayList<String>> reducedSet = new ArrayList<>();
    for (int i = 0; i < fullData.size(); i++) {

        // Do not reduce set for continuous vales
        if (!continuousArgs.contains(i)) {
            List<String> uniqueCol = distinctCounter(fullData.get(i));
            ArrayList<String> alConvert = new ArrayList<>(uniqueCol);

            if (i == classifierSpecifier) {
                modelClasses = alConvert;
            } else {
                reducedSet.add(alConvert);
            }
        } else {
            reducedSet.add(new ArrayList<>(fullData.get(i)));

        }
    }
    return reducedSet;
}
```

For this implementation, the discrete and continuous features are handled sequentially. Continuous features are defined under a class scoped `List<Integer> continuousArgs = Arrays.asList(3, 4);` which defines which columns in a CSV file must be treated as non-discrete values. Discrete values are processed when the `!continuousArgs.contains(i)` criteria is met. Otherwise, the set's mean and standard deviation are found to build the CDF. The CDF functions are provided by Princeton's Computer Science: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne.

The core data extraction methods are made to be highly dynamic. The model can store any number of discrete and continuous features, number of classes, and more distinct discrete buckets. Only until the `classift()` method will it become specialized for this data set.

5 Evaluation

The accuracy of this model is dependent on the quantity and quality of the training data. Provided a training set with 10,000 entries, a model is created. The testing set is 2,000 entries. After the simple classifier algorithm, a total percentage is calculated by taking the amount of correct predictions divided by the total number of predictions made.

```
Total Accuracy:  1609 correct / 2000 total = 80.45% Accuracy
```

Upon closer inspection, the model predicts `class0` for all given entries. This would be problematic if the algorithm is guessing the most commonly occurring class without feature consideration. However, real probabilities are being computed and compared: one class is produced throughout the set because the training set is heavily biased towards `class0` with little to no correlation between features. This is displayed by the probability counters for each given test entry.

```
Testing phase:
-------------------------------------------------------------------------
F1      F2      F3      F4        F5       CLASS    PREDICT  PROB    RESULT
---     ---     ---     -----     -------  -----    -------  -----   ---------
Germany 1       1       102603    747      class0   class0   79.2%   CORRECT
Spain   0       1       0.0       707      class0   class0   87.8%   CORRECT
Spain   0       0       0.0       590      class0   class0   86.3%   CORRECT
Spain   1       1       105000    603      class1   class0   89.2%   INCORRECT
France  1       1       0.0       615      class0   class0   93.9%   CORRECT


Total Accuracy: 1609 correct / 2000 total =  80.45%  Accuracy
```

For prediction, the bias towards `class0` is still prevalent. Experimentally, these results show that the classifier is working as expected. Another test set was built where the features do affect the class label of the record. There are drawbacks to this approach, but it does manage to start having the classifier produce a good variety of both class 1 and class 0. This is because in this self-made data set the following are true:

- France and Germany are more likely to produce a 0 while Spain is more likely to produce a 1.
- If the person doesn't have a credit card and isn't an active member, it's more likely to produce a 0.

This was done using a separate data generation Java program using the `Random` package. The code in this data generation is truly naive but it does make it so that at least one or two features do tell how the class might turn out. This does however break the fundamental assumption of conditional independence in the naive Bayes classifier. Some other data correlation could be done (like if a person has a balance under 50,000 it might be more likely for them to not have exit, class 0) but the goal was to show that the classifier is working at a fundamental level.

```
F1      F2      F3      F4         F5     CLASS   PREDICT PROB    RESULT
---     ---     ---     -----      ------ -----   ------- -----   ---------
France  1       1       32152.32   799    class0  class1  59.8%   INCORRECT
France  1       0       84968.58   646    class0  class0  63.1%   CORRECT
France  1       1       67.85      806    class1  class1  59.0%   CORRECT
France  1       0       22178.43   551    class1  class0  63.0%   INCORRECT
Germany 1       0       2828.31    833    class1  class0  63.9%   INCORRECT
```

```
Total Accuracy: 1308 correct / 2000 total =  65.40%  Accuracy
```

Note that the accuracy has dropped because it is still assumed that every feature has equal weight on the outcome of the class. Conditional independence between features was not modeled properly during the generation of the new data set.

6 Conclusions

The Naive Bayes classier is a simple yet effective way to label a set of given features. It assumes conditional independence between predictors: the value of one feature is independent of the value of another. Both in a mathematical and programmatic context, continuous data must be treated separately when calculating the probabilistic outcome of events. The quality and quantity of a data set determine the accuracy and precision of the model. It is therefore advantageous to provide a large and indicative data set up front. It can be thrown away later as only the model is needed to classify future data sets.

Bibliography

[1] Tarasova OA, Rudik AV, Biziukova NY, Filimonov DA, Poroikov VV. Chemical named entity recognition in the texts of scientific publications using the naïve Bayes classifier approach. Journal of Cheminformatics [Internet]. 2022 Aug 13 [cited 2023 Feb 23];14(1):1–12. Available from: https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=158508612

[2] Abascal-Mena R, López-Ornelas E, Pinto D, Singh V, Perez F. Author detection: Analyzing tweets by using a Naïve Bayes classifier. Journal of Intelligent & Fuzzy Systems [Internet]. 2020 Aug [cited 2023 Feb 23];39(2):2331–9. Available from: https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=145429364