An Implementation of Naive Bayes Classifier

Feng-Jen Yang
Department of Computer Science
Florida Polytechnic University
Lakeland, Florida, USA
fyang@floridapoly.edu

***Abstract*—The necessity of classification is highly demanded in real life. As a mathematical classification approach, the Naive Bayes classifier involves a series of probabilistic computations for the purpose of finding the best-fitted classification for a given piece of data within a problem domain. In this paper, an implementation of Naive Bayes classifier is described. This classifier can be used as a general tool kit and applicable to various domains of classifications. To ensure the correctness of all probabilistic computations involved, a sample data set is selected to test this classifier.**

***Index Terms*—Naive Bayes Classifier, Probabilistic Classification, Bayesian Theory**

I. Introduction

Classification is a commonly used machine learning and data mining approach. Depending on the number of target classifications that are used to classify a data set, different approaches might be chosen to perform the classification work. For binary classifications usually decision trees and support vector machines are commonly adopted, but these two approaches are under a constraint that the number of target classifications cannot go beyond two. This rigid constraint makes them hard to be generalized to fit a broad sense of real-life classification works in which the number of target classifications are usually more than two. From the standing point of acquiring a general tool kit, the Naive Bayes Classifier is more suitable for general classification expectations. There has been a good variety of successful real-life applications that are based on Naive Bayes classifier, such as weather prediction services, customer credit evaluations, health condition categorizations and so on. As long as the format of a data set within the problem domain is preprocessed into a tabular format. This mathematical classifier can go on to compute the validities of fitting a piece of new data into each possible classification. In this manner, the classification with highest fitness value can be chosen to be the best-fitted classification of this piece of data.

II. The Mathematical Background

The Naive Bayes Classifier is a kind of probabilistic classification mechanism rooted from Bayesian Theorem which is a posthumous theory of Thomas Bayes [1] [2]. From the perspective of classification, the main goal is to find the best mapping between a piece of new data and a set of classifications within a particular problem domain. For the purpose of making this mapping probabilistically computable, some mathematical manipulations are performed to transform joint probabilities into the multiplications of prior probabilities and conditional probabilities. As a machine learning and data mining approach, this mathematical transformation is kind of unnatural and might be hard for beginners to comprehend because it is turning a simple division into a long series of numerators divided by another long series of denominators. However, these unnatural transformations are necessary as prior probabilities and conditional probabilities are easy to summarize from a given data set by simply counting the number of instances with or without a given condition.

By considering a given problem domain consisting of n attributes and m classifications, the classifier will compute the validity of mathematical mapping between a vector of attributes (A_1, A_2, ..., A_n) and a set of classifications (C_1, C_2, ..., C_n). The validity of fitting a given attribute vector, ($A_1 = v_1$, $A_2 = v_2$, ..., $A_n = v_n$), into a classification C_i can be computed by performing Bayesian inference to compute the posterior probability as follows:

$$\begin{aligned}
&P(C_i|(A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n)) \\
&= \frac{P(C_i \cap (A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n))}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)} \\
&= \frac{P((A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n) \cap C_i)}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)} \\
&= \frac{P((A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n)|C_i) \times P(C_i)}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)} \\
&where \ \ 1 \le i \le m
\end{aligned}$$

After this series of probabilistic computations, the fitness value between a given attribute vector and a possible classification can be quantitatively represented.

A. An Assumption

The computation of $P((A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n)|Ci)$ is shown to be very tedious and intensively increasing the computational complexity of this classification work. In real life, usually an assumption is made to bring down the complexity of this computation.

By assuming the mutual independence among all attributes, the following mathematical transformation is hold:

$$\begin{aligned}
&P((A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)|C_i) \times P(C_i) \\
&= P(A_1 = v_1|C_i) \times P(A_2 = v_2|C_i) \times ... \times P(A_n = v_n|C_i) \times P(C_i) \\
&where \ \ 1 \le i \le m
\end{aligned}$$

978-1-7281-1360-9/18/$31.00 ©2018 IEEE
DOI 10.1109/CSCI46756.2018.00065

The posterior probability can, thus, be simplified into:

$$P(C_i|(A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n))$$
$$= \frac{P(C_i \cap (A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n))}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)}$$
$$= \frac{P((A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n) \cap C_i)}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)}$$
$$= \frac{P((A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n)|C_i) \times P(C_i)}{P(A_1 = v_1 \cap A_2 = v_2 \cap \ \cap A_n = v_n)}$$
$$= \frac{P(A_1 = v_1|C_i) \times P(A_2 = v_2|C_i) \times ... \times P(A_n = v_n|C_i) \times P(C_i)}{P(A_1 = v_1 \cap A_2 = v_2 \cap ... \cap A_n = v_n)}$$
$$where \ \ 1 \leq i \leq m$$

Although the assumption of mutual independence may not be true in some real-life problem domains, this assumption is commonly adopted in most of the Bayesian related computations and the results are still very trustable. It once was applied to successfully estimate the location of a wrecked flight in 2010 [3]. The possible probabilistic distortions that might be caused by this assumption can be illustrated by computing and comparing the best-fitted cases, partial-fitted cases, and worst-fitted cases [4].

B. A Further Simplification

Since all of the posterior probabilities have the same denominator involved in the computation, for the purpose of relative comparisons, the denominator can be dropped from the fraction by computing and comparing the numerator only. So the validity of mapping a given attribute vector, $(A_1 = v_1, A_2 = v_2, ...A_n = v_n)$ to a classification C_i can be further simplified as:

$$P(A_1 = v_1|C_i) \times P(A_2 = v_2|C_i) \times ... \times P(A_n = v_n|C_i) \times P(C_i)$$
$$where \ \ 1 \leq i \leq m$$

After computing all validities, the highest validity shows the best mapping between this attribute vector and its best-fitted classification.

III. The Implementation

Mathematically, the probabilistic computations between the Naive Bayes classifier and Bayesian inference are somehow overlapped. As a result, the following implementation of Naive Bayes classifier is also kind of similar to one of my previous implementation of a Bayesian inference engine [5] [6]. The class types of prior probability and conditional probability that were implemented in the previous inference engine are reused in the implementation of this classifier.

By considering the modularity and reusability, the entire scope of implementation is modeled into four class types, namely, the $Priority\ Probability$ class type, the $Conditional\ Probability$ class type, the $Domain\ Knowledge$ class type, and the $Classification$ class type.

The class type of $Prior\ Probability$ consists of:

- A $constructor$ that is able to initialize a prior probability.
- An overloading of the $string$ operator that is able to display a prior probability in string format.

The class type of $Conditional\ Probability$ consists of:

- A $constructor$ that is able to initialize a conditional probability.
- An overloading of the $string$ operator that is able to display a conditional probability in string format.

The class type of $Domain\ Knowledge$ consists of:

- A $constructor$ that is able to initialize a list of prior probabilities and a list of conditional probabilities within an application domain.
- An overloading of the $string$ operator that is able to display the list of prior probabilities and the list of conditional probabilities in string format.

The class type of $Classification$ consists of:

- A $constructor$ that is able to initialize a domain knowledge, a given classification, and a list of attribute values.
- A $classify$ method that will compute the validity of fitting the given list of attribute values to the given classification.
- An overloading of the $string$ operator that is able to display the computed validity in string format.

These four class types are implemented as a module in Python programming language which can be brought into any other Python program and be used as a general tool kit for classification purpose. The entire source code of this module is listed in Appendix A.

IV. A Test Case

To test this classifier, an example from an artificial intelligence book is selected to verify and ensure the mathematical correctness of this probabilistic classifier [7]. The selected data set is shown in Table 1. In this table each piece of data consists of attributes x, y, and z, where x, y, and z are integers within the range from 1 to 4 and the available classifications are A, B, and C.

TABLE I
A Selected Data Set for Testing

x	y	z	Classification
2	3	2	A
4	1	4	B
1	3	2	A
2	4	3	A
4	2	4	B
2	1	3	C
1	2	4	A
2	3	3	B
2	2	4	A
3	3	3	C
3	2	1	A
1	2	1	B
2	1	4	A
4	3	4	C
2	2	4	A

To find the best-fitted classification for the given new data, (x=2, y=3, z=4), the domain knowledge in terms of prior probabilities and conditional probabilities must be inferred first. So that they can be plugged into the aforementioned probabilistic computations.

Based on Table 1, the prior probabilities of classifications A, B and C are computed as follows:

- Among the 15 sample data, 8 of them are belonging to classification A. So without checking any value of any particular attribute. The prior probability of classification A is 8/15, i.e., $P(A) = 8/15$.
- Among the 15 sample data, 4 of them are belonging to classification B. So without checking any value of any particular attribute. The prior probability of classification B is 4/15, i.e., $P(B) = 4/15$.
- Among the 15 sample data, 3 of them are belonging to classification C. So without checking any value of any particular attribute. The prior probability of classification C is 3/15, i.e., $P(C) = 3/15$.

TABLE II
THE SUBSET DATA OF CLASSIFICATION A

x	y	z	Classification
2	3	2	A
1	3	2	A
2	4	3	A
1	2	4	A
2	2	4	A
3	2	1	A
2	1	4	A
2	2	4	A

To figure out the conditional probabilities with the constraint of data belonging to classification A only, as shown in Table 2, the following three conditional probabilities are computed:

- Among the 8 sample data that are belonging to classification A, 5 of them are having x=2. So given the condition of classification A, the probability of having x=2 is 5/8, i.e., $P(x = 2|A) = 5/8$.
- Among the 8 sample data that are belonging to classification A, 2 of them are having y=3. So given the condition of classification A, the probability of having y=3 is 2/8, i.e., $P(y = 3|A) = 2/8$.
- Among the 8 sample data that are belonging to classification A, 4 of them are having z=4. So given the condition of classification A, the probability of having z=4 is 4/8, i.e., $P(z = 4|A) = 4/8$.

TABLE III
THE SUBSET DATA OF CLASSIFICATION B

x	y	z	Classification
4	1	4	B
4	2	4	B
2	3	3	B
1	2	1	B

To figured out the conditional probabilities with the constraint of data belonging to classification B only, as shown in Table 3, the following three conditional probabilities are computed:

- Among the 4 sample data that are belonging to classification B, 1 of them are having x=2. So given the condition of classification B, the probability of having x=2 is 1/4, i.e., $P(x = 2|B) = 1/4$.
- Among the 4 sample data that are belonging to classification B, 1 of them are having y=3. So given the condition of classification B, the probability of having y=3 is 1/4, i.e., $P(y = 3|B) = 1/4$.
- Among the 4 sample data that are belonging to classification B, 2 of them are having z=4. So given the condition of classification B, the probability of having z=4 is 2/4, i.e., $P(z = 4|B) = 2/4$.

TABLE IV
THE SUBSET DATA OF CLASSIFICATION C

x	y	z	Classification
2	1	3	C
3	3	3	C
4	3	4	C

To figured out the conditional probabilities with the constraint of data belonging to classification C only, as shown in Table 4, the following three conditional probabilities are computed:

- Among the 3 sample data that are belonging to classification C, 1 of them is having x=2. So given the condition of classification C, the probability of having x=2 is 1/3, i.e., $P(x = 2|C) = 1/3$.
- Among the 3 sample data that are belonging to classification C, 2 of them are having y=3. So given the condition of classification C, the probability of having y=3 is 2/3, i.e., $P(y = 3|C) = 2/3$.
- Among the 3 sample data that are belonging to classification C, 1 of them is having z=4. So given the condition of classification C, the probability of having z=4 is 1/3, i.e., $P(z = 4|C) = 1/3$.

With the support of these prior probabilities and conditional probabilities, the validities of fitting (x=2, y=3, z=4) to classifications A, B, and C can be computed by the Python program listed in Appendix B. The resultant outputs are shown in Appendix C.

By comparing these three validities, the best-fitted classification for (x=2, y=3, z=4) is concluded to be A. These three validities seem to be small from the standing point of probabilities. However, after dropping the denominator from the computation of posterior probabilities, these three values are no longer representing probabilities. They are just for the purpose of supporting relative comparisons of how well is a given piece of data fitted into each possible classification.

V. Conclusion

Bayesian-based probabilistic computations have been widely adopted to predict outcomes under uncertainties. By taking advantage of the powerful built-in programming constructs in Python programming language, this implementation of Naive Bayes classifier is done without much intensive coding. The main contribution of this implementation is to provide a general tool kit that can be applied in a big variety of classification domains. This well-modeled implementation is not only suitable for classroom demonstrations but also suitable for real-life applications.

REFERENCES

[1] T. Bayes, "An Essay Towards Solving a Problem in the Doctrine of Chances," Philosophical Transactions of the Royal Society, Volume 53, Issue 1, 1763, pp. 370-418.

[2] J. Tabak, "Probability and Statistics: The Science of Uncertainty," NY: Facts On File, Inc., USA, 2004, pp. 46-50.

[3] E. Klarreich, "In Search of Bayesian Inference," ACM Communications, Volume, 58 Issue 1, 2015, pp. 21 - 24.

[4] F. Yang, "Eliciting an Overlooked Aspect of Bayesian Reasoning," ACM SIGCSE Bulletin, Volume 39, Issue 4, 2007, pp. 45-48.

[5] F. Yang, "Crafting a Lightweight Bayesian Inference Engine," Proceedings of The World Congress on Engineering (WCE 2016), Vol. II, London, UK, 2016, pp. 612-617.

[6] F. Yang, "A General Purpose Probabilistic Inference Engine," IAENG Transactions on Engineering Sciences - Special Issue for the International Association of Engineers Conferences 2016, Singapore: World Scientific, 2017, pp. 33-44.

[7] B. Coppin, "Artificial Intelligence Illuminated, MA: Jones and Bartlett Publishers Inc., USA, 2004, pp. 352 353.

APPENDIX A
THE IMPLEMENTATION OF THE NAIVE BAYES CLASSIFIER

```
"""
File Name: classifier.py
Creator: Feng-Jen Yang
Purpose: Implementing a Naive Bayesian classifier
"""

class PrioProb(object):
    """Representing a prior probability"""

    def __init__(self, initClas, initProb):
        """Initializing the prior probability for each classification"""
        self.clas = initClas
        self.prob = initProb

    def __str__(self):
        """Returning a string representation of the prior probability"""
        return "P(" + self.clas + ") = " + str(self.prob)

class CondProb(object):
    """Representing a conditional probability"""

    def __init__(self, initAttr, initClas, initProb):
        """Initializing the conditional probability"""
        self.attr = initAttr
        self.clas = initClas
        self.prob = initProb

    def __str__(self):
        """Returning a string representation of the conditional probability"""
        return "P(" + self.attr + "|" + self.clas + ") = " + str(self.prob)

class DomainKnowledge(object):
    """Representing a knowledge domain"""

    def __init__(self, initPrioProbs, initCondProbs):
        """Initializing the prior probabilities and conditional probabilities"""
        self.listOfPrioProbs = initPrioProbs
        self.listOfCondProbs = initCondProbs

    def __str__(self):
        """Returning a string representation of the knowledge base"""
        result = "The Domain Knowledge:"
        for p in self.listOfPrioProbs:
            result += "\n" + str(p)
        for p in self.listOfCondProbs:
            result += "\n" + str(p)
        return result

class Classification(object):
    """Representing Bayesian classification"""
    def __init__(self, initDK, initClas, initAttrs):
        """Initializing the knowledge base"""
        self.dk = initDK
        self.clas = initClas
        self.listOfAttrs = initAttrs

    def classify(self):
        """Computing the probability of the given classification"""
        num = 1
        for cp in self.dk.listOfCondProbs:
            if (cp.clas == self.clas) and (cp.attr in self.listOfAttrs):
                num *=cp.prob
        for pp in self.dk.listOfPrioProbs:
            if pp.clas == self.clas:
                num *= pp.prob
                break

        return round(num, 4)

    def __str__(self):
        """Returning a string representation of the inference reault"""
        result = "When ("
        for a in self.listOfAttrs:
            result += a

            if a != self.listOfAttrs[-1]:
                result += ", "
            else:
                result += "), the validity to be in class " + self.clas + " is " + str(self.classify())

        return result
```

APPENDIX B
THE TEST CASE

```
"""
File Name: demo.py
Creator: Feng-Jen Yang
Purpose: An Example of Using the Naive Bayes Classifier
"""

from classfier import *

def main():
    #instantiate the classifications
    listOfClassifications = ["A", "B", "C"]

    #Instantiate the prior probabilities
    listOfPrioProbs = [PrioProb("A", 8/15), PrioProb("B", 4/15), PrioProb("C", 3/15)]

    #Instantiate conditional probabilities
    listOfCondProbs = [CondProb("X=2", "A", 5/8), CondProb("Y=3", "A", 2/8),\
                       CondProb("Z=4", "A", 4/8), CondProb("X=2", "B", 1/4),\
                       CondProb("Y=3", "B", 1/4), CondProb("Z=4", "B", 2/4),\
                       CondProb("X=2", "C", 1/3), CondProb("Y=3", "C", 2/3),\
                       CondProb("Z=4", "C", 1/3)]

    #Instantiate the domain knowledge
    dk = DomainKnowledge(listOfPrioProbs, listOfCondProbs)

    #Display the domain knowledge summerized from the data set
    print(str(dk), "\n")

    #Display the validities in each classification
    print("The Classification:")

    for c in listOfClassifications:
        validity = Classification(dk, c, ["X=2", "Y=3", "Z=4"])

        #display the classification result
        print(validity)

#The entry point of program execution
main()
```

APPENDIX C
THE OUTPUTS FROM THE TEST CASE

```
The Domain Knowledge:
P(A) = 0.5333333333333333
P(B) = 0.26666666666666666
P(C) = 0.2
P(X=2|A) = 0.625
P(Y=3|A) = 0.25
P(Z=4|A) = 0.5
P(X=2|B) = 0.25
P(Y=3|B) = 0.25
P(Z=4|B) = 0.5
P(X=2|C) = 0.3333333333333333
P(Y=3|C) = 0.6666666666666666
P(Z=4|C) = 0.3333333333333333

The Classification:
When (X=2, Y=3, Z=4), the validity to be in class A is 0.0417
When (X=2, Y=3, Z=4), the validity to be in class B is 0.0083
When (X=2, Y=3, Z=4), the validity to be in class C is 0.0148
```