Algorithms for Data Mining

Hidden Naive Bayes

applied to Vehicle accidents

Second delivery

C., Laura
G., Carles

June 30, 2018

Contents

1 Introduction

In our first delivery, we focused on selecting an interesting data set: the recollection of vehicles' accidents occurred during 2016 in the United Kingdom, published by their Department for Transport.

We cleaned the data, selected the most important and interesting features for us and joined the three files in order to be able to use the whole data set comfortably.

The aim of this new study is to make a classifier that is able to predict an accident severity by taking into account all the other variables, such as how the weather or light were at the moment of the accident.

To do so, we have chosen to study ***Hidden Naive Bayes*** (also known as and referred to as *HNB*) and afterwards implement it based on the explanation of its behaviour in the paper that proposed it.

This algorithm is an extension of Naive Bayes algorithm (also referred to as *NB*), which is well-known in the Machine Learning community, and promises to give better results than the algorithm that it extends, Naive Bayes.

After implementing the *HNB* algorithm, our goal is to check if it has indeed better accuracy than *NB*.

So, to summarize, **this project is threefold**:

1. **Study Hidden Naive Bayes** in order to see how it works. Compare it with the traditional Naive Bayes, showing the differences of its behaviour.

2. **Implement** it in *python*.

3. **Apply it to our data set**, which we preprocessed in our previous report. Compare the results vs using the traditional Naive Bayes.

2 Study

First we want to study the Hidden Naive Bayes classifier.

To do so, first a brief study is made on how Naive Bayes, the algorithm in which HNB is based on, works.

Afterwards, a specification on how Hidden Naive Bayes works will be presented followed by a comparison between both of them, stating their main differences.

2.1 Naive Bayes

Naive Bayes (also known as *Naive Bayesian Classifier*) is a well-known classifier that computes probabilities of an element pertaining to a class or another based on the occurrence of the same values in all the attributes.

It computes the classifier model with a training set composed by different individuals, thus counting each combination occurrence of the attributes' values.

Mathematically, this explanation roughly translates to the following:

1. Let D be a set of elements $D = e_1, e_2, ..., e_m$ and let A be a set of **independent** attributes $A = A_1, A_2, ..., A_n$, where m is the total number of elements and n is the total number of attributes.

2. An individual e is an element of the set that contains a value for **each** attribute such that $e_i = (a_1, a_2, ..., a_n)$, $0 \leq i \leq m$. Each a_x corresponds to the value of the attribute A_x.

3. Let C be the class variable, which varies depending on the values of the attributes. It can take up to k different classes such that: $C = c_1, c_2, .., c_k$.

4. Now let e be a new element without class defined and with a set of attributes $e = (a_1, a_2, ..., a_n)$ defined. That is, with values.

   To classify it, Naive Bayes would compute the probability of that element pertaining to each class with the attributes given and the training model already defined. The class it selects is the one with most probability. Thus:

$$c(e) = max_{c \in C} P(c) \prod P(a_i|c) \tag{1}$$

We want to note that in this classifier, which its approach is basic statistics, **the assumption of Independence between attributes is made**. That is, that no attribute is derived from another and so they have no influence between them whatsoever.

However, as its name indicates, this is a very naive assumption and this is rarely the case. Nevertheless, the classifier works generally well despite this assumption and it is widely used.

2.2 Hidden Naive Bayes

Hidden Naive Bayes is a new classifier proposed and published in 2009 by a research group formed by Liangxiao Jiang, Harry Zhang, and Zhihua Cai.

It is based on Naive Bayes but instead of assuming that the attributes are independent, this classifier considers them to be dependant on each other. In fact, the structural model on which Naive Bayes is based on accepts dependencies between attributes. This structural model is is called *Bayesian Networks.*

However, generating the optimal Bayesian Network from the training data by taking into account the dependencies between variables is not time efficient. This problem by itself is NP-Hard.

Thus **Hidden Naive Bayes discards a brute force discovery of the optimal dependencies between attributes** using Bayesian Networks and instead, its proposal is that a parent is virtually created for each attribute as a combination of other attributes. This parent is *hidden*, underlying in the data, and is what *Hidden* from the algorithm's name refers to.

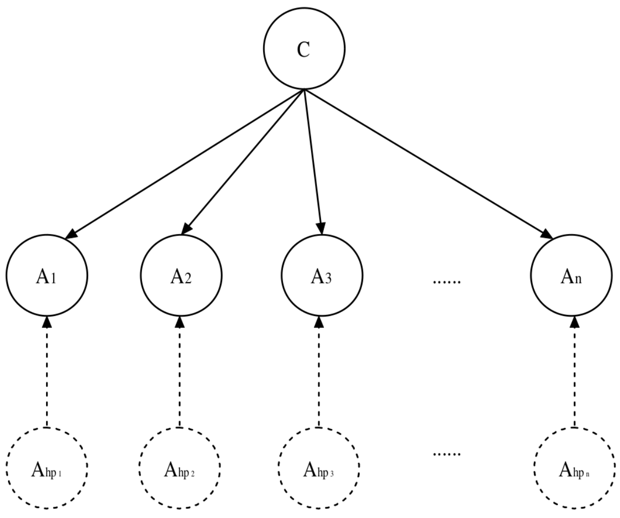

Figure 1: Structure of Hidden Naive Bayes.

To compute the probability of an attribute depending on its parent and the classifier, the following formula is applied:

$$P(A_i|A_{hpi}, C) = \sum_{j=1, j\neq i}^{n} W_{ij} * P(A_i|A_j, C) \tag{2}$$

The sum of all weights must be one:

$$\sum_{j=1, j\neq i}^{n} W_{ij} = 1 \tag{3}$$

The conditional mutual information (I_p) is the expected value of the mutual information contained in two random variables given a third:

$$I_p(X;Y|Z) = \sum_{x,y,x} P(x,y,z) \log \frac{P(x,y|z)}{P(x|z)P(y|z)} \tag{4}$$

This conditional mutual information is used to compute the weights in the edges between two attributes: a son and its parent. These weights specify the influence of on attribute to another, so the greater the weight, the more influence one attribute has to another.

$$W_{ij} = \frac{I_p(A_i; A_j|C)}{\sum_{j=1, j\neq i}^{n} I_p(A_i; A_j|C)} \tag{5}$$

So now, to classify a new element, the probability of an element pertaining to a class also takes into account the influence of the hidden parent for each attribute:

$$c(e) = max_{c\in C} P(c) \prod P(a_i|a_{hp_i}, c) \tag{6}$$

where a_{hp_i} is the attribute's parent influence.

2.3 Comparison

Having explained briefly how these two classifiers work, we can now state that they differ in the following points:

1. **Dependency.** Naive Bayes assumes that attributes A are independent between them, which is usually not the case. Hidden Naive Bayes accepts the possibility of their dependency.
2. **Existence of parents.** Which is derived from the previous point. Since HNB assumes dependency, it also assumes a hidden parent for each attribute and computes the weight that each attribute has for all the other ones.

3 Implementation

To implement Hidden Naive Bayes we tackled it with two steps.

Firstly, an implementation of Naive Bayes has been done to fully understand its mechanics.

Secondly, our own version of the implementation of Hidden Naive Bayes classifier has also been done, following the HNB paper [1].

Both implementations can be found attached to this report and have been done in ***Python* 3.6**, and **so their executions should be made with this version**. Otherwise the results may differ with our own due to incompatibilities in the Python language.

3.1 Pseudo-code with explanation

We are only going to focus on the implementation of Hidden Naive Bayes, for plenty of examples and implementations of Naive Bayes can be found in varying languages and it is not important for our problem at hand.

Thus, the following pseudo-code reflects the behaviour of our implementation of Hidden Naive Bayes.

```
Algorithm 1: HNB
  Result: Creates a HNB model and predicts with a test set subset of the given
          dataset.
  Input : D, // Dataset to work with
          splitRatio,  // Split ratio to split the dataset into a trainingSet and testSet
          predictColumnIdx // Class (predict) column index
1 trainingSet, testSet ← split(D)
2 classValues, classProbs, classCount ← computeClassProbabilites(D,
   predictColumnIdx)
3 attrValueIdx, Ip, attrCount ← countAttrComputeMutualInfo(D, classValues,
   predictColumnIdx, classProbs)
4 predict(D, Ip, testSet, attrCount, attrValueIdx, classCount, classValues,
   predictColumnIdx)
```

After reading the CSV file containing the data set, the *HNB* method handles:

1. **The data partition**. It splits the data in a training and testing set, according to a a *splitRatio* value.

2. Invokes the *computeClassProbabilites* that **computes the $P(C)$ class probability**.

3. Invokes the *countAttrComputeMutualInfo* as a second step that computes, for each possible class value and for pair of attribute values, **the total number of instances that match both the class value and the attributes value**. This method also returns the **conditional mutual information Ip**.

4. After computing the information, it **predicts** the test set *testSet* using the previous computed values.

Algorithm 2: countAttrComputeMutualInfo

```
Result: Returns the number of instances that match the given attributes values and
        the mutual information
Input  : D, classValues, predictColumnIdx, classProbs
Output: attrCount, Ip
attrs = D[0]
numAttr = len(attrs)
attrCount = [numAttr][numAttr]
foreach instance in D do
    iClass = getClass(instance) foreach attr in instance do
        if attr != iClass then
            foreach attr2 in instance do
                if attr2 != iClass then
                    attrCount[iClass][attr][attr2] ++
                end
            end
        end
    end
end
foreach son in attrs do
    foreach parent in attrs do
        Ip[son][parent] ← computeMutualInfo(D, son, parent, attrCount, classProbs)
    end
end
return attrCount, Ip
```

This method method does a double job:

1. From one side, it stores the total number of instances for each class $c_i \subseteq C$ and for each a pair of attributes A_i, A_j with values a_i, a_j. The resulting matrix will be known as the attribute count.

2. Using the previous matrix, it computes the conditional mutual information between two attributes invoking the method *computeMutualInfo*.

Algorithm 3: computeMutualInfo

```
Result: Computes the
Input  : D, son, parent, attrValueIndex, attrCount, classProbs
Output: Ip
Ip ← 0
totalRows ← size(D)
foreach class in C do
    foreach attrVal in parent do
        foreach attrVal2 in son do
            P_Ai_C ← attrCount[class][attrVal][attrVal] / totalRows
            P_Aj_C ← attrCount[class][attrVal2][attrVal2] / totalRows
            P_Ai_Aj_C ← attrCount[class][attrVal][attrVal2] / totalRows
            Ip += P_Ai_Aj_C * $\log_2$ (P_Ai_Aj_C * classProbs[class] / (P_Ai_C
             * P_Aj_C ))
        end
    end
end
return Ip
```

- The method *computeMutualInfo* computes the probabilities of a given class and a given attribute (for both a parent attribute and a son attribute) as well as the joint probability. Later on, those values are going to be used to compute the conditional mutual info **following the approach of the equation** (4)**.**

Algorithm 4: predict

```
Result: Classifies each individual of the testSet using the trained model.
Input  : D, Ip, testSet, attrCount, attrValueIndex, classCount, classValues,
         predictColumn
correctPredictions ← 0
foreach instance in testSet do
    classProbs ← computeProbabilities()
    predictClass ← max(classProbs)
    if predictClass == instanceClass then
        correctPrections++
    end
end
accuracy ← correctPredictions / size(testSet)
printStatistics(correctPredictions, testSet, D)
```

- Given a set of instances that are not used in the training stage (test set), the predict method computes the probabilities of each class for each instance. Those probabilities are calculated using the approach in equation (6). As the paper pointed out, in order to avoid having zeroes while computing the probabilities, we have used the Laplace correction:

$$P(c) = \frac{n_c + 1}{t + k} \tag{7}$$

$$P(a_i|c) = \frac{n_{ic} + 1}{n_c + v_i} \quad (8)$$

$$P(a_i|a_j, c) = \frac{n_{ijc} + 1}{n_{jc} + v_i} \quad (9)$$

After that, it predicts the instance class taking the maximum probability and it compares to the instance true class. After all the instances are made, statistics (regarding the quality of the predictions) are shown.

3.2 Implementation comparison

Now that we have both Naive Bayes and Hidden Naive Bayes implemented, it is time to test how well the classifier does.

The important metrics that are used in order to assess how well HNB classifies are the total prediction accuracy as well as the proportion between false positives and false negatives for each class.

We are going to compare them with the Titanic data set provided to us in this course.

This dataset has been chosen because it is simple enough to do a full comparison. It only has four attributes, so choosing one of them as the class leaves the algorithm with only three attributes to work with.

They are:

1. **Class**. Modality in which the passenger traveled with.
   (a) Crew.
   (b) 1st.
   (c) 2nd.
   (d) 3rd.
2. **Sex** of passenger.
   (a) Male.
   (b) Female.
3. **Age** of passenger.
   (a) Adult.
   (b) Child.
4. **Survived**.
   (a) Yes.
   (b) No.

We have executed both our implementations several times in Naive Bayes (NB) and Hidden Naive Bayes (HNB) using each time a different attribute as the class and as the **split ratio 0.8**. That is, 80% of the input data is randomly taken and used to train our model and the other 20% is used as to test it.

Our results in general have been, in average accuracy, the following:

Attribute as class	**% accuracy NB**	**% accuracy HNB**
Class	41	46
Sex	73	78
Age	60	**95**
Survived	73	76

So as we can see there is in average more accuracy using Hidden Naive Bayes.

In fact, in the case of age there is a clear advantage on applying Hidden Naive Bayes instead of Naive Bayes when using Age as the class to be predicted. This is due to attributes Class, Sex and survived having weights on their impact of the age attribut. In other words, they are dependant.

With these tests we can affirm at Hidden Naive Bayes works at least as well as Naive Bayes, and generally speaking slightly better than Naive Bayes when dependency between attributes is present.

More examples on executions on each attribute used as the class can be found at the appendix.

4 Vehicles Accidents classification with HNB

Now that we have verified that Hidden Naive Bayes works correctly and returns better results, we proceed to apply it to our preprocessed dataset from our previous delivery.

4.1 Preprocessing

Again, some small preprocessing had to be done in order to create classes correctly, for we had only computed binary classes as *Good/Bad* or *True/False* in multiple attributes. Due to this, these values were not unique (for they were reused in multiple attributes), so we renamed each possible value according to the attribute it described.

For example, *WEATHER* attribute could be either *Good* or *Bad*. So we changed its possible values to *Good_ Weather* or *Bad_ Weather*.

Naive Bayes and Hidden Naive Bayes also do not allow missing values. So instead of imputing values, we decided to just remove all the individuals with missing attributes, for we had plenty of individuals.

We started with 136.575 individuals and after cleaning we ended up with 68.787, which is still a lot.

And finally, we also removed some new categories as well as creating new ones.

More specifically, we created a new attribute containing whether the accident occurred on weekend or workday, as well as reducing the Time attribute to just the hour to decrease the number of attribute values to just 24.

The removed attributes have been Date, Accident Index, Latitude, Longitude and worstCasualtySeverity, for we do not deem the date, accident index or worstCasualtySeverity important (this last one had always the same value), and Latitude and Longitude are not categorical values, so they are not useful.

4.2 Applying HNB

After the data set has been ready to be applied Naive Bayes and Hidden Naive Bayes on, we proceed to do so.

One of the categories that we are more interested in predicting is the Severity of the accident.

We want to know if there is the possibility to be able to classify the accident severity according to the rest of attribute's variables.

A correct classification could help to identify what we call *dark variables.* That is, values of the attributes that in conjunction may lead statistically most cases to an accident with a high severity.

Naive Bayes could help identify variables if they were independent, but we know with a simple glance that a number of ours are dependant. For example, this could be the case of the *weekDay* attribute and whether the attribute occurred in a workday or weekend. And there may be many dependencies that we are not able to see at simple glance.

It is because of these dependencies, hidden or not, **that our hypothesis is that Hidden Naive Bayes will be useful for us**.

After applying HNB using the Accident Severity as the class to be predicted, as well as other columns that define the characteristics of the accident, we encounter the accuracies in the predictions shown in the table below.

In order to check how good (or bad) HNB is, we are going to compare it with the traditional Naive Bayes and see if our hypothesis that HNB is preferred in our case is true or not.

Attribute as class	**% accuracy NB**	**% accuracy HNB**
Accident Severity	67.05	**79.04**
Road Type	8.15	**82.89**
Speed Limit	11.64	**66.75**
Road conditions	78.31	**83.46**
Weather conditions	82.39	82.29

Having seen these results, summarized in the table above, we can conclude that our hypothesis was correct.

Even more, we were not expecting such good results in most cases. Take, for example, the case of Road type. The accuracy increased from an 8% to an 82%. Or in the case of the Speed Limit, the increase was from 11% to 66%.

We want to note that the improvement on the results prediction comes with quite higher execution times (44 sec. compared to 3). However, we deem this tradeoff **worth**.

So all in all, we are satisfied at how Hidden Naive Bayes performs and can conclude and it is preferred over Naive Bayes.

5 Conclusions

After having studied both Naive Bayes and Hidden Naive Bayes, implemented and tested them, our conclusions are multiple:

We have encountered several points in favor of Hidden Naive Bayes:

1. **Assuming the independence of attributes**, despite providing good results, is a **very naive** assumption which realistically speaking is, more often than not, **false**.

2. Computing possible dependencies from one attribute to another as weights (and taking into account those at prediction time) has demonstrated to provide solid results. In our tests, way better than those obtained at Naive Bayes.

3. Weights are useful to see the detected dependencies between attributes.

This observed increase in accuracy is, of course, dependant on the data set used.

In our case, it has been more evident in our own large vehicles' data set than on the Titanic one, although in both cases the obtained accuracy has been better than the one obtained in Naive Bayes.

Therefore, we can relate the improvement in prediction of the Hidden Naive Bayes classifier to the number of attributes that a data set has, as well as the number of individuals observed.

Although neither the complexity nor the execution time are taken as an important metric at the time when a prediction has to be made, the execution times for Naive Bayes are lower than those obtained in Hidden Naive Bayes.

To summarize, **Hidden Naive Bayes** have given us much more **better results** than Naive Bayes in all our tests and is **preferred as a classifier**, for the assumption of **independency of attributes is almost never true**.

6 Appendix

Examples of executions of Hidden Naive Bayes for each attribute are:

- **Class**

Value	Correctly guessed	Incorrectly guessed	% accuracy
Crew	180	225	44.44
1st	30	6	83.33
2nd	0	0	-
3rd	0	0	-

- **Sex**

Value	Correctly guessed	Incorrectly guessed	% accuracy
Male	352	87	80.18
Female	0	2	0.0

- **Age**

Value	Correctly guessed	Incorrectly guessed	% accuracy
Adult	415	26	94.1
Child	0	2	-

- **Survived**

Value	Correctly guessed	Incorrectly guessed	% accuracy
No	279	108	72.09
Yes	50	4	92.59

6.1 Execution examples of traces generated

6.1.1 Titanic dataset

#Initial rows: 2201 splitted.
#Train rows: 1760
#Test rows: 441
Accuracy: 65.75963718820861%

Split 2201 rows into train=1760 and test=441 rows
——-

```
Class -> Sex : 0.5
Class -> Age : 0.0
Class -> Survived : 0.5
Sex -> Class : 0.33
Sex -> Age : 0.0
Sex -> Survived : 0.67
Age -> Class : 0.2
Age -> Sex : 0.4
Age -> Survived : 0.4
Survived -> Class : 0.33
Survived -> Sex : 0.67
Survived -> Age : 0.0
——-
Accuracy: 94.33
——-
* Adult || OK: 416 || FAIL: 25 || acc: 94.33
* Child || OK: 0 || FAIL: 0 || acc: 100
```

6.1.2 Accidents dataset

```
#Initial rows: 68787 splitted.
#Train rows: 55029
#Test rows: 13758
Accuracy: 60.90274749236808%
```

```
    Split 68787 rows into train=55029 and test=13758 rows
——-
Accident_Severity -> Number_of_Vehicles : 0.01
Accident_Severity -> Number_of_Casualties : 0.01
Accident_Severity -> Time : 0.05
Accident_Severity -> Road_Type : 0.01
Accident_Severity -> Speed_limit : 0.01
Accident_Severity -> Junction_Detail : 0.01
Accident_Severity -> Light_Conditions : 0.24
Accident_Severity -> Weather_Conditions : 0.06
Accident_Severity -> Road_Surface_Conditions : 0.26
. . .
——-
Accuracy: 79.35
——-
* SLIGHT || OK: 10449 || FAIL: 2006 || acc: 83.89
* SERIOUS || OK: 455 || FAIL: 834 || acc: 35.3
* FATAL || OK: 13 || FAIL: 1 || acc: 92.86
```

References

[1] Liangxiao Jiang, Harry Zhang, and Zhihua Cai. “A novel bayes model: Hidden naive bayes”. In: *IEEE Transactions on knowledge and data engineering* 21.10 (2009), pp. 1361–1371.

[2] *Road Safety Data.* URL: `https://data.gov.uk/dataset/road-accidents-safety-data`.