Machine Learning Course

Lab assignment 2: Naive Bayes classifier

Augustin VIOT

Contents

Introduction

This document is a report of the lab assignment done in the course of Machine learning at the University of Genova (UNIGE). It describes the theory and implementations of a Bayes Naïve classifier.

Theory explanation:

Framework:

In our exercise, we want to determine if given a description of the weather today, we decide whether to go play outdoor.

In this case, the features correspond to the different parameters describing the weather (Outlook, Temperature, Humidity, Windy). These parameters can take 2 or 3 different values. The target is the answer to the question "should we play outdoor?", it can take two values (YES or NOT).

The objective of a classifier is to determine which class we should choose given a set of feature. For example, in our case, the goal would be to determine if we should play outdoor given a weather description (for example: outlook=Sunny, Temperature=hot, Humidity=High, Windy=TRUE).

How to determine the appropriate class?

What we actually want is: "find which class is the most appropriate for these feature", or in other terms "which class has the highest probability given these feature". In a math point of view, this correspond to finding which has the high probability between:

$$P(\omega = YES \mid X) \text{ and } P(\omega = NO \mid X) \text{ where}$$

- ω is the target and
- $X = (outlook = Sunny,\ Temperature = hot,\ Humidity = high,\ Windy = TRUE)$ is the vector of features describing the weather.

Comment: for convenience, we will write $X = (outlook = Sunny,\ Temperature = hot,\ Humidity = high,\ Windy = TRUE)$ as $X = (X_1 = x_1, X_2 = x_2, X_3 = x_3, X_4 = x_4)$

To do so, we calculate both of the probabilities.

How to calculate $P(\omega = YES \mid X)$?

Thanks to the Bayes theorem (refer to annex 1 for a recall) we can write the following equality:

$$P(\omega = YES \mid (X_1 = x_1, X_2 = x_2, X_3 = x_3, X_4 = x_4)) = \frac{P(\omega = YES) \times P(X_1 = x_1, X_2 = x_2, X_3 = x_3, X_4 = x_4 \mid \omega = YES)}{P(X_1 = x_1, X_2 = x_2, X_3 = x_3, X_4 = x_4)}$$

Or:

$$P(\omega = YES \mid X) = \frac{P(\omega = YES) \times P(X \mid \omega = YES)}{P(X)}$$

Thanks to the Conditional probability rules, we can write $P(X|\omega = YES)$ as followed:

$$P(X|\omega = YES) = P(X_1 = x_1, X_2 = x_2, X_3 = x_3, X_4 = x_4|\omega = YES)$$

$$\begin{aligned} P(X|\omega = YES) = \ & P(X_1 = x_1|\omega = YES) \times P(X_2 = x_2|\omega = YES, X_1 = x_1) \\ & \times P(X_3 = x_3|\omega = YES, X_1 = x_1, X_2 = x_2) \\ & \times P(X_4 = x_4|\omega = YES, X_1 = x_1, X_2 = x_2, X_3 = x_3) \end{aligned}$$

But the Naïve Bayes Classifier use the Naïve assumption: "The effect of the value of a predictor x on a given class c is independent of the values of the other predictors". In other terms, a naïve Bayes classifier assumes that the presence of a particular feature in a class in unrelated to the presence of any other feature. The assumption, also called **class conditional independence** can be expressed with a math formula:

$$P(x_i|\ \omega, x_j) = P(x_i|\ \omega)$$

Or:

$$P(X_i = x_i|\ \omega, X_j = x_j) = P(X_i = x_i|\ \omega)\ \ , \forall\ i \neq j$$

So now, we can simplify $P(X|\omega = YES)$

$$\begin{aligned} P(X|\omega = YES) = \ & P(X_1 = x_1|\omega = YES) \times P(X_2 = x_2|\omega = YES) \times P(X_3 = x_3|\omega = YES) \\ & \times P(X_4 = x_4|\omega = YES) \end{aligned}$$

$$P(X|\omega = YES) = \prod_{i=1}^{4} P(X_i = x_i|\omega = YES)$$

Now we have:

$$P(\omega = YES\ |X\ ) = \ P(\omega = YES)\ \times \prod_{i=1}^{4} P(X_i = x_i|\omega = YES)\ \times \frac{1}{P(X)}$$

We proceed with the same method to get

$$P(\omega = NO\ |X\ ) = \ P(\omega = NO)\ \times \prod_{i=1}^{4} P(X_i = x_i|\omega = NO)\ \times \frac{1}{P(X)}$$

Now, we just need to calculate the two probabilities, and choose the highest one.

Application:

When we apply the classifier in real life, we actually do not need to calculate $\frac{1}{P(X)}$.

Actually, the Bayes classifier function that assigns a class label $\hat{y} = C_k$ for some k can also be written as follows:

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{argmax}\, p(\omega_k) \times \prod_{i=1}^{n} P(x_i|\omega_k)$$

With

- $K : number\ of\ possible\ target$
- $\omega_k : the\ possible\ target$
- $n : number\ of\ features$
- $x_i : the\ possible\ feature$

This simplified formula of the Bayes classifier can be written thanks to the following simplification:

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{argmax}\, P(\omega_k|X)$$

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{argmax}\, p(\omega_k) \times \frac{1}{P(X)} \times \prod_{i=1}^{n} P(x_i|\omega_k)$$

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{argmax}\, p(\omega_k) \times \prod_{i=1}^{n} P(x_i|\omega_k)$$

With:

- $X : vector\ of\ features$

Implementation

With the Bayes classifier formula, we can now see that we actually need two information to classify a point with the Bayes classifier:

- $p(\omega_k)$ for each class
- $P(x_i|\omega_k)$ for each feature value of our point and each class

To store these values, we will use two data structures in our program, an array for the $p(\omega_k)$, and a cell array for the $P(x_i|\omega_k)$ value.

General representation of our data structure

Here are two illustration of our data structure:

$p(\omega_1)$	$p(\omega_2)$	…	$p(\omega_K)$

With:

- $K : number\ of\ possible\ target$
- $\omega_k : the\ possible\ target$

To see the cell array, please go to annex number 2.

Specific representation of our data structure:

With our specific dataset, this is what our data structures will look like :

Array for the (ω_k) :

$p(Play = yes)$	$p(Play = no)$

To see what the cell array look like with our specific dataset, please look at annex number 3.

General structure of the program

Our program contains 3 code files and one data files. The code files are:

- **Exercice1.m** : contains the general script to charge the data, call the bayes function and print the results
- **naiveClassification.m** : contains the function *naiveClassification*. It takes two data set and returns the classification (and error rate if the test set contains a target), or errors is the parameters do not meet certain requirement (described in the next part). This function is only used to check the data dimension, then call the *BayesTraining* function is the data given in the parameters are correct.
- **BayesTraining.m** : contains the actual code used to build the classifier and classify the test set.

Here is a description of the function *BayesTraining* function:

This function takes two argument:

- *trainingDataSet* : a matrix of size $(d+1) \times numerOfTrainingObservation$, with d the dimension of our feature (number of feature)
- *testDataset* : a matrix of size $(d+1) \times numerOfTrainingObservation$ OR (d)*numerOfTrainingObservation. The size depends on if the test set includes the target column or not.

In normal use it returns two arguments:

- *targetTest* : a vector containing the class of the test observation determined by the Bayes classifier
- *errorRate* : this is only returned when the *testSet* has the column of target. In this case, it has the value of the error rate of the Bayes classification

The function can also return errors. When it returns an error, the value of *targetTest* = 0, and *errorRate* contains a message. Errors are returned in this cases :

- Training set and test set have different target values
- Test set have different feature value(s)

Structure of the code : the function contains 4 parts

- **CHECK THE DATASET** : it check if the test set as different feature/target values from the *trainingSet*
- **TRAINING OF THE CLASSIFIER** : it build a cell array **likehoodMatrix** and a vector **targetValues**. Those contain the information needed to make the Bayes classification
- **CLASSIFICATION OF THE TEST SET**
- **ERROR RATE** : it compute the error rate if the *testSet* contains the target column

Constraints given by the exercise

Here we describe how we handle the constraints asked in the exercise.

Task 1: Data preprocessing

We had to make a data preprocessing. Actually, we wanted the Classifier to work with integer positive values (more convenient because they can also be used as indexes for matrix). To do so, we just transformed the data using the following rules:

Outlook	Temperature	Humidity	Windy	Play
Overcast => 1	cool => 1	normal => 1	TRUE => 1	yes => 1
Rainy => 2	mild => 2	high => 2	FLASE => 2	no => 2
Sunny => 3	hot => 3			

Since the data set is really small, we did the transformation manually. Here is the result of this transformation.

Dataset before transformation	Dataset after transformation

```
#Outlook   Temperature  Humidity  Windy  Play
overcast   hot          high      FALSE  yes
overcast   cool         normal    TRUE   yes
overcast   mild         high      TRUE   yes
overcast   hot          normal    FALSE  yes
rainy      mild         high      FALSE  yes
rainy      cool         normal    FALSE  yes
rainy      cool         normal    TRUE   no
rainy      mild         normal    FALSE  yes
rainy      mild         high      TRUE   no
sunny      hot          high      FALSE  no
sunny      hot          high      TRUE   no
sunny      mild         high      FALSE  no
sunny      cool         normal    FALSE  yes
sunny      mild         normal    TRUE   yes
```

```
1   3       2   2   1
1   1       1   1   1
1   2       2   1   1
1   3       1   2   1
2   2       2   2   1
2   1       1   2   1
2   1       1   1   2
2   2       1   2   1
2   2       2   1   2
3   3       2   2   2
3   3       2   1   2
3   2       2   2   2
3   1       1   2   1
3   2       1   1   1
```

We saved this modified data in a file named **dataset1modified.txt**.

Task 2: Build a naive Bayes classifier (Check data set and dimension)

In the *naiveClassification* function, we make the following checks before calling the *BayesTraining* function:

- Check that the number c of columns of the second matrix is at least the number d of columns of the first matrix - 1
- Check that no entry in any of the two data sets is <1

Then we call the *BayesTraining* function that will perform the following test:

- Check if the test set contains the target column (then the two dataset would have the same number of columns). This information is used at the end of the program to see if error rate should be compute or not. If the target is included in test set, we also test that all target values exist in the training set.
- Then check for all features, if the test set contains no new feature values that the training set did not have (otherwise the classifier would not know how to compute these values).

Training of the classifier

As said before, to make the classifier, we have to build a cell array containing all the useful probabilities. However, for better understanding of our task, we decide to make a two steps building:

1. **Build a frequency cell array** :

   This cell array contains the correct number of occurrence of each events. We build it by a series of loop going throw the features, features values and target values of our training set. Please refer to the code comments for more details about the algorithm used.

   Comment: this frequency cell array may contains cell with an occurrence number equal to 0.

2. **Build a likehood cell array** :
   This likehood cell array contains all the values ready to be used to compute the classification. To fill up this array, we need to use a smoothing (explanation below) to calculate the estimation of the probabilities. . Please refer to the code comments for more details about the algorithm used.

Smoothing:

It may happen (and in our case it happens), that some specific cell of the frequency cell array have no instance at all. The computation without smoothing would give a 0 probability, which cannot be used for our Bayes classifier. We used Laplace's smoothing rule (or Laplace's "rule of succession") :"If an event occurs r times in n experiments, then its probability is (r+1)/(n+2) rather than r/n."

Also, we had to build an array containing the $p(\omega_k)$ values, but this was quite an easy task so please refer to the code comments for more details about the algorithm used.

Classification of the test set

To classify a points we just need to find the required values to apply the following formula:

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{argmax}\, p(\omega_k) \times \prod_{i=1}^{n} P(x_i|\omega_k)$$

The values can be found in the cell array and the vector we built during the classification, so we just need to find these values and make the computation to find the class given by the classifier. Then we can repeat these steps for the whole test set.

The results of the classification is a vector of all target for each data points and it is stored in the *targetTest* returned variable of the functions.

An example of a classification for one point is described in the code, please refer to it and its comments for more precision about the algorithm used.

Error rate computation

As said before, the error rate is only computed when the test set contains a target column. To compute it, we simply make a comparison between the target given in the test set, and the target found by the classifier. We sum up the correct values, the compute the error rate. When the test set contains no target column, the error rate variable to return is set to 2.

Description of the initial script

The initial script had many constraints asked by the exercise:

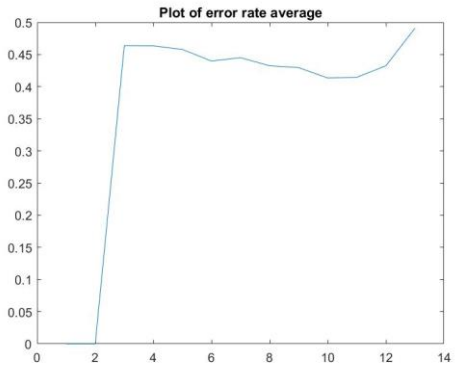
3. Loads the weather data set (already converted in numeric form)
4. Splits it into a training set with 10 randomly chosen patterns and a test set with the remaining 4 patterns
5. Calls the naive Bayes classifier and reads the results
6. Prints the results: classification for each pattern of the test set, corresponding target if present, error rate if computed.

We met all these constraints, but since the code is really easy to implement, please refer to the comments for details about the methods used.

Analysis of the results + possible improvement

Analysis of our program:

Our program works well, it always has an appropriate behavior. To analyze the performance of our classifier and the influence of the training/test sets sizes, we draw a graph representing the error rate average VS the number of elements in the training set. For each average, we used 1000 classifications to compute the average error rate. Here is the result:



Comment: we had to use a training set of a minimum size of 3 values, otherwise we cannot get enough feature values.

We can see here that the error rate decrease when the size of the training set increase until around 10 data points. With more than 10 data points in the training set, the error rate increases. This could be explained by the fact that a big training set imply a small test set. Moreover, we have to consider that we are working with a really small dataset, so it is really hard to analyze such results.

Possible improvements:

Due to the random selection of values in the data set during the initial script, the it may happen that the test set contains all the data with the a specific target/feature value (for example, the test set contains all the points where the target is “yes”). This mostly due to the really small size of our dataset. This generate the appropriate error message (‘Training set and test set have different target values’ or ‘Test set have different feature value(s)’).

This can be avoided by check if the random selection produce correct training and test sets, but this was not asked in our exercise.

What if the data are not categories?

In case the data contains numerical value (for example if the temperature was given using degrees instead of hot/cool/mild), we would have to use different methods. An easy way would be to transform the numerical data into categorical data to use the same methods. Another way could be to use the distribution of the numerical value to guess the frequency (for example, we can assume that the data follows a normal distribution, and guess the frequency).

References

Useful references:

- Machine learning course
- “Pattern classification”, second edition, Richard O. Duda, Peter E. Hart, David G. Stork

Other useful references:

- https://www.youtube.com/watch?v=XcwH9JGfZOU
- https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/#
- http://scikit-learn.org/stable/modules/naive_bayes.html

Annexes:

Annex 1: Bayes Theorem recall

Annex 2: Cell array general data structure

Annex 1: Bayes Theorem recall

Here are two ways to write the Bayes theorem:

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

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

Annex 2: Cell array general data structure

	ω_1	ω_2	...	ω_K
X_1	*possible values of the **First** feature* a_1 | b_1 | ... | d_1 $P(a_1\|\omega_1)$ | $P(b_1\|\omega_1)$ | ... | $P(d_1\|\omega_1)$	*possible values of the **First** feature* a_1 | b_1 | ... | d_1 $P(a_1\|\omega_2)$ | $P(b_1\|\omega_2)$ | ... | $P(d_1\|\omega_2)$	...	*possible values of the **First** feature* a_1 | b_1 | ... | d_1 $P(a_1\|\omega_K)$ | $P(b_1\|\omega_K)$ | ... | $P(d_1\|\omega_K)$
X_2	*possible values of the **Second** feature* a_2 | b_2 | ... | d_2 $P(a_2\|\omega_1)$ | $P(b_2\|\omega_1)$ | ... | $P(d_2\|\omega_1)$	*possible values of the **Second** feature* a_2 | b_2 | ... | d_2 $P(a_2\|\omega_2)$ | $P(b_2\|\omega_2)$ | ... | $P(d_2\|\omega_2)$	...	*possible values of the **Second** feature* a_2 | b_2 | ... | d_2 $P(a_2\|\omega_K)$ | $P(b_2\|\omega_K)$ | ... | $P(d_2\|\omega_K)$
...	...	...	...	...
X_d	*possible values of the **Last** feature* a_d | b_d | ... | d_d $P(a_d\|\omega_1)$ | $P(b_d\|\omega_1)$ | ... | $P(d_d\|\omega_1)$	*possible values of the **Last** feature* a_d | b_d | ... | d_d $P(a_d\|\omega_2)$ | $P(b_d\|\omega_2)$ | ... | $P(d_d\|\omega_2)$	...	*possible values of the **Last** feature* a_d | b_d | ... | d_d $P(a_d\|\omega_K)$ | $P(b_d\|\omega_K)$ | ... | $P(d_d\|\omega_K)$

With:

- d : *the number of features (or dimension)*
- $X_1, X_2, \dots, X_d$: *the features*
- $a_i, b_i, \dots, d_i$: *the possible value for the feature* i
- K : *the number of class (or target)*
- $\omega_1, \omega_2, \dots, \omega_K$: *the different class (or target)*

Annex 2: Cell array specific data structure

	$\omega = \omega_1 = 1 \ (play = yes)$	$\omega = \omega_2 = 1 \ (play = no)$
$Outlook$	*possible values of the feature Outlook* **Overcast** | **Rainy** | **Sunny** $P(Outlook = Overcast \mid \omega_1)$ | $P(Outlook = Rainy \mid \omega_1)$ | $P(Outlook = Sunny \mid \omega_1)$	*possible values of the feature Outlook* **Overcast** | **Rainy** | **Sunny** $P(Outlook = Overcast \mid \omega_2)$ | $P(Outlook = Rainy \mid \omega_2)$ | $P(Outlook = Sunny \mid \omega_2)$
$Temp$	*possible values of the feature Temp* **cool** | **mild** | **hot** $P(Temp = cool \mid \omega_1)$ | $P(Temp = mild \mid \omega_1)$ | $P(Temp = hot \mid \omega_1)$	*possible values of the feature Temp* **cool** | **mild** | **hot** $P(Temp = cool \mid \omega_2)$ | $P(Temp = mild \mid \omega_2)$ | $P(Temp = hot \mid \omega_2)$
Hum	*possible values of the feature Hum* **normal** | **high** $P(Hum = normal \mid \omega_1)$ | $P(Hum = high \mid \omega_1)$	*possible values of the feature Hum* **normal** | **high** $P(Hum = normal \mid \omega_2)$ | $P(Hum = high \mid \omega_2)$
$Windy$	*possible values of the feature Windy* **True** | **False** $P(Hum = True \mid \omega_1)$ | $P(Hum = False \mid \omega_1)$	*possible values of the feature Windy* **True** | **False** $P(Hum = True \mid \omega_2)$ | $P(Hum = False \mid \omega_2)$