Naive Bayes Classifier

Kohei Tateyama, S6427214, Università di Genova

***Abstract*—Naive bayes classifier is a simple algorithm which solves classification tasks. It uses the probability from the given dataset to predict the unknown factor of a new dataset. In this assignment we built a code for the naive bayes classifier, and implemented laplace smoothing to make the classifier robust to missing data. The goal of the assignment was to understand the fundamentals of the Naive bayes classifier through the process.**

***Index Terms*—Naive Bayes Classifier, Machine Learning, AI, Probability, Bayesian Classifier, Bayes' theorem, Classification**

I. Introduction

In a field of machine learning, several methods have been developed to solve classification problems. With each having advantages and disadvantages. But in particular, naive bayes classifier is known for its simplicity and efficiency. In this report, I will discuss the algorithm of the naive bayes classifier that I implemented in the assignment, show results of the classification, and conclude the report by stating the strengths and weaknesses of the classifier.

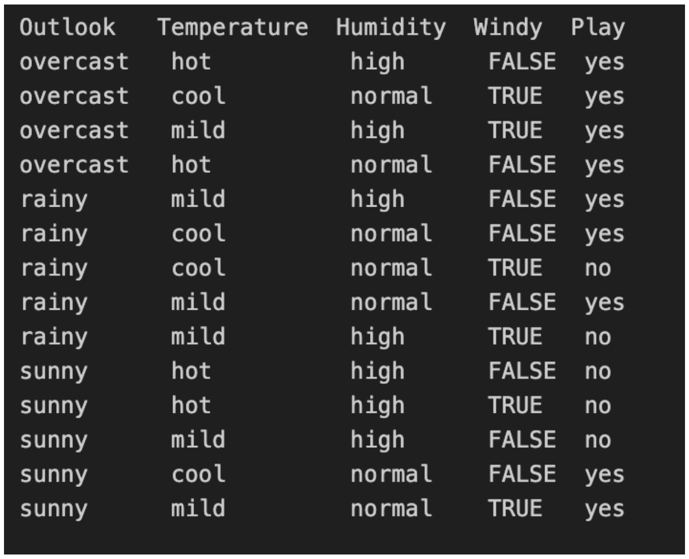
II. Lab Work

The code for this assignment is written in Python version 2.7.18. The assignment is done in three steps. First is data preprocessing, second is naive bayes classifier, and third is applying laplace smoothing to the make the classifier more robust to missing data.

A. Data Preprocessing

The data that I used in this assignment is data.txt file shown in figure 1. There are 14 data of which 10 are used as the training set, and 4 are used as test set. The 'Play' column is the target class, which we tried to predict for the test set.

In the data preprocessing process, the data.txt file is turned into a dataframe using pandas. Then, the data is split into training set and test set randomly with a random state of 42.



```
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
```

Fig. 1. Data

B. Naive Bayes classifier

Naive bayes classifier is divided into 2 sections. First is the training part where all the probabilities are calculated and stored in the dictionary. Second is the test part, where the test data are classified using the probabilities.

1) Train: The pseudo code for the training is shown in Algorithm 1. This training part is to find all conditional probabilities. As written in the pseudo code, the conditional probability of every target, in every element, of every factor is calculated and stored in the dictionary. Which we will then be used in the test part. A probability of 'Outlook' being 'Sunny' and the target being both 'yes' and 'no' is stored in a dictionary like {'Outlook' {'Sunny' {Probability of 'Yes', Probability of 'No'}}}.

Algorithm 1 Training algorithm for Naive bayes classifier

```
1: for every factor 'Outlook', 'Temperature', 'Humidity', 'Windy' do
2:     Make a dictionary to store probabilities
3:     for every element of a factor that are not the same do
4:         Make a sub-dictionary to store probabilities
5:         for every target that are not the same do
6:             Calculate and store the probability in dictionary
7:         end for
8:     end for
9: end for
```

2) Test: The pseudo code for the test algorithm is shown in Algorithm 2. As written in the pseudo code, for every test set, the conditional probability of every element is calculated for target being both 'yes' ad 'no'. After calculating the probability of each data, it will compare the probability of the target, and store that target class 'yes' or 'no' with the higher probability. This will be the prediction.

C. Laplace Smoothing

The laplace smoothing is a simple smoothing technique that solves 0 probabilities. This makes the classifier robust to missing data. To implement this code, I simply added 1 to the numerator, and 1 times the number of unique factors to the denominator of every conditional probabilities. The change in code is only in the line 6 of Algorithm 1.

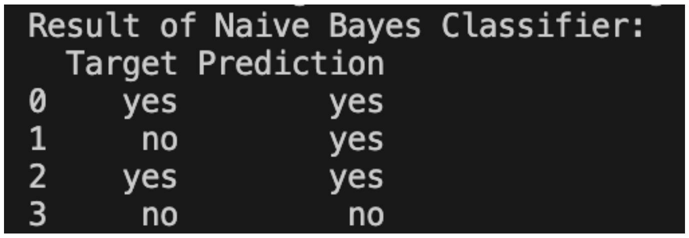
Algorithm 2 Test algorithm for Naive bayes classifier

```
Make an empty list to store predictions
for each test data do
    Make a dictionary to store new probabilities
    for each target class do
        Calculate the probability of target class
        for every factor 'Outlook', 'Temperature', 'Humidity', 'Windy' do
            Multiply all conditional probabilities with the target class probability to get new probabilities
        end for
        Store the new probabilities
    end for
    Compare each target probabilities
    Store the target class of the one with higher probability
end for
```

III. Results

A. Naive Bayes classifier

The result of the Naive Bayes Classifier is shown in figure 2. The left column is the actual target data, and the right column is the prediction of the naive bayes classifier. The error rate of the 4 test data was 25%.

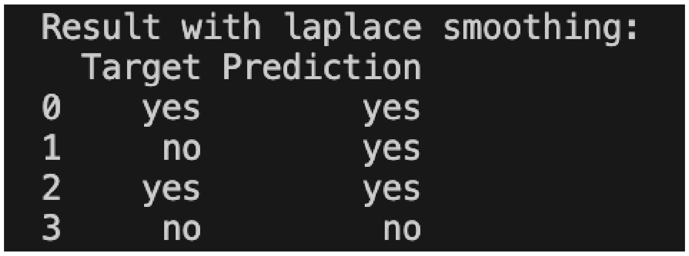


```
Result of Naive Bayes Classifier:
  Target Prediction
0    yes        yes
1     no        yes
2    yes        yes
3     no         no
```

Fig. 2. Result of Naive Bayes Classifier

B. Laplace Smoothing

The result after smoothing is shown in figure 3. The error rate was 25% and was the same without the smoothing.

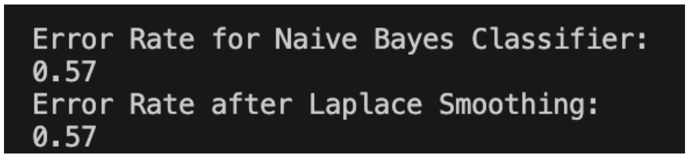


```
Result with laplace smoothing:
  Target Prediction
0    yes        yes
1     no        yes
2    yes        yes
3     no         no
```

Fig. 3. Result after Laplace Smoothing

C. Comparison

The error rate with and without laplace smoothing did not change. I tried implementing the cross validation method to get a more precise error rate of the two methods. The result of the cross validation is shown in figure 4. In contrary to my expectation, the error rate with and without the smoothing did not change. Laplace smoothing is a method to making the data robust to missing data, and it should not change the performance of the classifier itself.



```
Error Rate for Naive Bayes Classifier:
0.57
Error Rate after Laplace Smoothing:
0.57
```

Fig. 4. Error rate comparison

IV. How to run the code

The source code(NaiveBayesClassifier.py) and the dataset(data.txt) for this assignment is located in the same folder as this PDF file. The dataset has to be in the same folder as the source code. numpy, pandas must be installed. I have also used scikit-learn to implement the cross validation. But if not installed, comment out line 3,4 which is importing scikit-learn, and also comment out all of task 4 and it should run perfectly.

The source code is also written so that it works with other datasets. So to try running it with different datasets, simply add a dataset as a text file in the same directory, change the name of the file in line 10 of NaiveBayesClassifier.py and it should work. The only constraint is that the target must be in the very last column of the data.

V. Conclusion

The error rate of the naive bayes classifier without laplace smoothing was 57%, and the error rate with the laplace smoothing was also 57%. Though the smoothing method has made the classifier robust to missing data, the performance of the classifier did not change.

The performance of the classifier can be improved by scaling up the dataset. More importantly, by having more factors that are not related to each other. In real life, the target 'Play' will change with more factors such as 'the amount of tasks you have to do', or 'Health' or 'Amount of friends that can play on the same day' and more on. If the data had more factors like this, and more data, the error rate would have been lower.

After implementing the code by my self, I can say that the simplicity is the biggest advantage of this classifier. But oth the other hand, the lack of understanding on some of the complex connections with each factor can be a downside. 'Outlook', 'Windy', 'Temperature', 'Humidity' is all related to each other, so in such datasets, other machine learning methods would be suitable.