Email Classification with Naive Bayes

Abdullatif Dalab

27880960 abdulatif.saleh@hotmail.com

1. Introduction

1.1. Naive Bayes

Naive Bayes is a generative machine learning algorithm that uses Bayes' rule to model the posterior probability P(H|E), where E represents evidence/features and H represents a Hypothesis/class. Given the chain rule in probability theory, computing the joint and conditional probabilities becomes unfeasible as the number of features grow. The way Naive Bayes gets around that is by applying the conditional independence assumption on all the features so that they become independent from one another. In essence, Naive Bayes is a classification algorithm.

1.2. Naive Bayes as a Probabilistic Graphical Model (PGM)

PGM's use a graph to represent conditional dependencies & independencies between random variables. Naive Bayes can be modelled using a PGM. As a PGM, Naive Bayes graphical structure is that of a directed acyclic graph (DAG), so it can be seen as a bayesian network where the model's hypothesis is a random variable with no parent nodes, and the evidence/features are all the children nodes of hypothesis(**Figure 1**). Using a graphical model, we can reach conditional independence between all the random variables in the bayesian network by applying the local Markov assumption. The visual intuition of the notion of conditional independence and the graphs structure can be seen in the appendix(**Figure 1**).

1.3. Papers Content

In this report, I will be discussing how the code was structured, the analysis & comparison of 3 experiments on the task of email classification using Naive Bayes, the difficulties encountered, and the topics of interest that were influenced by the project.

1.4. Technical Details

A class named EmailNaiveBayesClassifier encompasses the functions required to successfully preprocess a corpus, train a model, test a model, and compute metrics. After setting the correct directories, all that needs to be done is to instantiate an object of the class and to call the parse, train, test, and metrics functions sequentially.

The main data structures used in the attributes of the class are lists & dictionaries. Lists were used to store labels, predictions, and the vocabulary, while dictionaries stored word frequencies and conditional probabilities. Time complexity of the program is linear O(n).

2. Analysis & Comparison of Experiments

2.1. Baseline experiment

In the case of class Ham, its recall is high (98.5%), meaning that it has a low rate of false negatives. It's precision (84.18%) is lower than recall, meaning it has a high rate of false positives.

In the case of class Spam, its recall is low(81.5%), meaning that it has a high rate of false positives It's precision (98.19%) is higher than recall, meaning it has a low rate of false negatives.

Hams F1-score (90.78%) is greater than Spams (89.07%), meaning that its precision & recall are more balanced. Ham's accuracy is greater than Spams by 17%. The total accuracy, precision, recall, and F1-score is 90%, 84.18%, 98.5% and 90.71% respectively. Please refer to the (**Figure 2**) in the appendix for the confusion matrix and the exact details of the results.

2.2. Stop-words filtering experiment

In the case of class Ham, its recall is high (98.5%), meaning that it has a low rate of false negatives. It's precision (84.73%) is lower than recall, meaning it has a high rate of false positives.

In the case of class Spam, its recall is low(82.25%), meaning that it has a high rate of false positives It's precision (98.2089%) is higher than recall, meaning it has a low rate of false negatives.

Hams F1-score (91.098%) is greater than Spams (89.523%), meaning that its precision & recall are more balanced. Ham's accuracy is greater than Spams by 16.25% The total accuracy, precision, recall, and F1-score is 90.375%, 84.73%, 98.5% and 91.098 % respectively. Please refer to the (**Figure 3)** in the appendix for the confusion matrix and the exact details of the results.

2.2. Word-length filtering experiment

In the case of class Ham, its recall is high (98%), meaning that it has a low rate of false negatives. It's precision (86.72%) is lower than recall, meaning it has a high rate of false positives.

In the case of class Spam, its recall is low (85%), meaning that it has a high rate of false positives. It's precision (97.7011%) is higher than recall, meaning it has a low rate of false negatives.

Hams F1-score (92.018%) is greater than Spams (90.909%), meaning that its precision & recall are more balanced. Ham's accuracy is greater than Spams by 13% The total accuracy, precision, recall, and F1-score is 91.5%, 86.725%, 98% and 92.018 % respectively. Please refer to the (**Figure 4**) in the appendix for the confusion matrix and the exact details of the results.

2.3. Comparison of the 3 experiments

Interestingly, in all the experiments, the Ham class had a higher precision and a lower recall while the Spam class had a higher recall and a lower precision. Ham's accuracy was also significantly higher in all cases. The number of training/testing samples per class are equal, so an imbalanced dataset doesn't seem to be introducing any algorithmic bias.

Nonetheless, the filters applied in the experiments seem to reduce the discrepancy between each class's precision and recall. That is realized by the increase in the total F1-score from (90.78%) to (92%). The F1 score was not the only metric that the filters improved.The accuracy of the baseline model increased from (90%) to (90.375%) after applying the stop words filter and to (91.5%) after applying the world length filter. To confirm that these filters are indeed improving the model, a testing experiment using both filters showed even higher accuracy, precision, recall, and F1 score. The results of the 4th experiments were: 92.125%, 87.69%, 98%, and 92.56% respectively. You can refer to (**Figure 5)** for more details about the results per class and the confusion matrix.

In all the experiments, the total number (Ham & Spam) of false positives were greater than false negatives. That explains overall low precision. After comparing all the results of class Ham and Spam separately in these experiments, class Spam had the lowest accuracy in every experiment (lower by 15.416% on average). A logical conclusion to make is that by figuring out a way to improve the predictions across Spam, the overall model will be able to perform better. Now, the question is: why would there be such discrepancy between Ham and Spam's metrics given that the dataset is perfectly balanced? One way to try to explain Spam's results is by assuming that the frequency of words in the Spam class vary a lot (spam content varies more than normal content). In other words, Spam's probability distribution has higher variance. Given that assumption, when calculating the posterior probability P(Spam|Words), a lot of the words will have low conditional probabilities, giving a window for false positives to occur (classify an email as Ham). To support this claim, the standard deviation for the frequencies of words in each class was calculated. In the baseline experiment, the Ham class had a standard deviation of 468.106 while the Spam class had a standard deviation of 710. You can refer to (**Figure 6)** in the appendix for details to see how the results were calculated. In the stop-words filtering experiment, the standard deviation for Ham & Spam were 342, 507 respectively. In the word-length filtering experiment, the standard deviation for Ham & Spam were 355.7, 482.8 respectively. In every experiment, the standard deviation of each class decreased be-

cause of the filters applied. The steady decrease in variance/standard deviation seems to be directly correlated to the steady increase in accuracy throughout the experiments. Hence, It is safe to conclude that by reducing the variance in the Spam class data distribution, a better overall accuracy can be attained.

3. Difficulties Encountered

3.1. Designing the class

I wanted to make class object calls as minimal and intuitive as possible so that it would be easy to test the classifier using different modifications, filters, and states. In the beginning, I made a Naive Bayes class that only encompassed methods relevant to the algorithm. All the other parsing and writing functions where outside the class. Then I realized that a lot of code repetition occurs in every test case. I decided to change the class so that it encompassed parsing, writings, and other methods. That way, only 3 to 4 function calls are required in every test case with no redundancies. To generalize it further, I prepared a block of code that takes 4 directories: Training, testing, filters such as stop-words, and output destination. That way, the user of the program only needs to set the correct paths at the very beginning. The rest can be done using only 4 functions: parsing, training, testing, and metrics.

4. Topics of Interest

4.1. Naive Bayes implementation as a graphical model

Lately, my interest in generative models lead me to study probabilistic graphical models. I wish to build a strong foundation in probabilistic thinking, so that I can better understand unsupervised learning algorithms. When I stumbled upon bayesian networks, I realized that their was graphical interpretation of Naive Bayes that was even more intuitive. If I were to continue working on this project, I would like to implement the classifier as a probabilistic graphical model. It would be nice to compare the results and see if there is any difference. Despite how powerful the supervised approach is, it feels like the right move to invest more time in the other side.

5. References

1. Wikipedia contributors. (2019, March 16). Graphical model. In Wikipedia, The Free Encyclopedia. Retrieved 17:04, April 10, 2019, from https://en.wikipedia.org/w/index.php?title=Graphical_model&oldid=888011275

6. Appendix

6.1. Figure 1

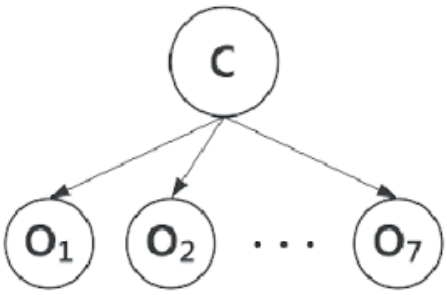

Fig. 1. This is the graphical representation of a Naive Bayes model.

6.2. Figure 2

```
CONFUSION-MATRIX:
[['TP:394' 'FP:74']
 ['FN:6' 'TN:326']]
-----------------------------------------
METRICS:

HAM:
Accuracy: 98.5%
Precision: 84.1880341880342%
Recall: 98.5%
F1-Score: 90.78341013824884%

SPAM:
Accuracy: 81.5%
Precision: 98.19277108433735%
Recall: 81.5%
F1-Score: 89.07103825136612%
-----------------------------------------
TOTAL RESULTS
ACCURACY:90.0%
PERCISION:84.1880341880342%
RECALL:98.5%
F1-Score:90.78341013824884%
-----------------------------------------
```

Fig. 2. Baseline experiment results.

6.3. Figure 3

```
CONFUSION-MATRIX:
[['TP:394' 'FP:71']
 ['FN:6' 'TN:329']]
-----------------------------------------
METRICS:

HAM:
Accuracy: 98.5%
Percision: 84.73118279569893%
Recall: 98.5%
F1-Score: 91.09826589595376%

SPAM:
Accuracy: 82.25%
Percision: 98.2089552238806%
Recall: 82.25%
F1-Score: 89.52380952380953%
-----------------------------------------
AVERAGED RESULTS:
ACCURACY:90.375%
PERCISION:84.73118279569893%
RECALL:98.5%
F1-Score:91.09826589595376%
```

Fig. 3. Stop-words experiment results.

6.4. Figure 4

```
CONFUSION-MATRIX:
[['TP:392' 'FP:60']
 ['FN:8' 'TN:340']]
-----------------------------------------
METRICS:
Class Ham:
Accuracy: 98.0%
Percision: 86.72566371681415%
Recall: 98.0%
F1-Score: 92.01877934272301%
Class Spam:
Accuracy: 85.0%
Percision: 97.70114942528735%
Recall: 85.0%
F1-Score: 90.90909090909092%
-----------------------------------------
AVERAGED RESULTS:
ACCURACY:91.5%
PERCISION:86.72566371681415%
RECALL:98.0%
F1-Score:92.01877934272301%
```

Fig. 4 Word-length filtering experiment results.

6.5. Figure 5

```
CONFUSION-MATRIX:
[['TP:392' 'FP:55']
 ['FN:8' 'TN:345']]
-------------------------------------------
METRICS:
Class Ham:
Accuracy: 98.0%
Percision: 87.69574944071589%
Recall: 98.0%
F1-Score: 92.56198347107438%
Class Spam:
Accuracy: 86.25%
Percision: 97.73371104815864%
Recall: 86.25%
F1-Score: 91.63346613545818%
-------------------------------------------
AVERAGED RESULTS:
ACCURACY:92.125%
PERCISION:87.69574944071589%
RECALL:98.0%
F1-Score:92.56198347107438%
```

Fig. 5 Word-length filtering + Stop-words filtering experiment results.

6.6. Figure 6

```
In [11]: w_g_h = [] # Frequency of words in Ham
         w_g_s = [] # Frequency of words in Spam
         "Filing the lists with data from naive bayes class object"
         for word in nbo.word_given_ham_freq:
             w_g_h.append(nbo.word_given_ham_freq[word])
             w_g_s.append(nbo.word_given_spam_freq[word])

In [16]: # Standard deviation of Ham class
         np.std(w_g_h)

Out[16]: 468.10622815381362

In [17]: # Standard deviation of Spam class
         np.std(w_g_s)

Out[17]: 710.24826841853894
```

Fig. 6 Standard deviation of Ham and Spam class in the baseline experiment.