CS 191 Naïve Bayes Classifier Exercise Report

Submitted by
Femo Bayani and Mikaela Ramos

INTRODUCTION

Spam Emails

Spam emails are “unsolicited bulk e-mail” that are posted blindly to thousands of recipients. These spam emails can waste time and bandwidth, and “may expose under-aged recipients to unsuitable (e.g. pornographic) content” [1].

Naive Bayes Classifier

The Naive Bayes classifier uses the Bayes’ Theorem with a “naive” assumption regarding the independence of a feature. It is “naively” assumed that each feature is independent from other features.

The Bayes’ Theorem is used during prediction, like this:

$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$, where:

- $P(c \mid x)$ is the posterior probability
- $P(x \mid c)$ is the likelihood
- $P(c)$ is the class prior probability
- $P(x)$ is the predictor prior probability

This **posterior probability** means that given the test document x, what is the probability of it belonging to class c?

The **likelihood** or conditional probability: given class c, what is the probability of the test document x belonging to it? Furthermore, the likelihood can be expanded:

$P(x \mid c_j) = P(x_1 \mid c_j) * P(x_2 \mid c_j) * ... * P(x_k \mid c_j)$, where x_i is a feature in the document, and c_j is a class

$P(x \mid c_j) = \Pi\, P(x_i \mid c_j),\ for\ i\ = 1\ to\ k$

Laplace Smoothing

Laplace smoothing is a technique applied for the smoothing of categorical data [2]. Laplace smoothing is a common component of Naive Bayes classifiers and is due to the problem when a feature was not in the training data, because this would mean that it’s probability of occuring would be 0, resulting in a total 0 probability for that specific word [3].

$$P(x_i \mid c_j) = \frac{count_{x_i,c_j}}{totalCount_{c_j}} \rightarrow \frac{count_{x_i,c_j} + 1}{totalCount_{c_j} + |V|}$$

IMPLEMENTATION

Data Preparation

The data set used is the TREC 2007 Public Spam Corpus, a dataset containing 75419 emails, 50199 (67%) of which are spam, and the remaining 25220 (33%) are ham.

The goal for data preparation of the researchers were to create a single `.csv` file with all the relevant preprocessed information stored. Multiprocessing was also utilized to speed up the preprocessing of the huge data set.

The following tasks were performed on the data set for data preparation and preprocessing using `preprocess.py` (a Python script, also in the source code and Github repository), based on the full subcorpora (index found at `dataset\trec07p\full\index`):

1. Reading the Files

   Each entry in the index was read, also followed by the reading of the corresponding email indicated in the entry, which undergo the next steps.

2. Obtaining the Body of the Email

   The body of each email is obtained and is passed to the next step for the actual preprocessing.

3. Preprocessing the Text
   a. Applying a lowercase filter

      All text is converted to lowercase.

   b. Stripping all the HTML tags

      All HTML tags are stripped, retaining only the actual text contents of the email. There have been cases where emails only contained images (through IMG tags), and thus these emails end up having no text left. Emails

   c. Removing accents

      All accented characters are transformed to their non-accented versions.

   d. Expanding contractions

      Contractions and slang words are transformed to their expanded forms. For example:

      ```
      you're -> you are
      i'm    -> I am
      ```

```
ima    -> I am going to
yall   -> you all
gotta  -> got to
```

e. Removing all special characters

All HTML tags are stripped, retaining only the actual text contents of the email. There have been cases where emails only contained images (through IMG tags), and thus these emails end up having no text left. These emails get removed from the data set later on.

f. Stop word removal

Stop words refer to the most common words in a language. This project uses NLTK's list of english stopwords. Some examples include "i", "me", "myself", "we", and many more. These words are removed from the current text.

g. Lemmatization

Lemmatization is a form of word normalization, performed on each word to remove inflectional endings, obtaining the base form of a word, known as the lemma. For example:

```
dogs       -> dog
churches   -> church
aardwolves -> aardwolf
abaci      -> abacus
hardrock   -> hardrock
are        -> are
is         -> is
```

4. Cleaning

Entries with empty texts are removed from the data set. A total of 2383 (3.15%) emails were removed.

5. Saving

A `.csv` file with the 4 columns (index, is_spam, email_path, text) is stored as `processed.csv`, for later use, during model training. The first few lines of the file looks like this:

```
,is_spam,email_path,text
0,1,../data/inmail.1,feel pressure perform rising occasion try viagra
anxiety thing past back old self
1,0,../data/inmail.2,hi ive updated gulu check mirror seems little typo
debianreadme file example httpgulususherbrookecadebianreadme
ftpftpfrdebianorgdebianreadme testing lenny access release diststesting
```

```
current tested development snapshot named etch package tested unstable
passed automated test propogate release etch replace lenny like readmehtml
yan morin consultant en logiciel libre yanmorinsavoirfairelinuxcom
unsubscribe email debianmirrorsrequestlistsdebianorg subject unsubscribe
trouble contact listmasterlistsdebianorg
2,1,../data/inmail.3,mega authenticv g discount pricec l discount pricedo
miss click httpwwwmoujsjkhchumcom mailboundary authentic viagra mega
authenticv g discount pricec l discount pricedo miss click
3,1,../data/inmail.4,hey billy really fun going night talking said felt
insecure manhood noticed toilet quite small area worry website telling
secret weapon extra inch trust girl love bigger one ive time many chick
since used pill year ago package used month supply one worth every cent
website httpctmaycom ring weekend go drink let know secret later dude brad
```

6. Summary

	Original	**Cleaned**	**Remaining**
Total	75419	2383 (3%)	73036 (97%)
Spam	50199 (67%)	1854	24691
Ham	25220 (33%)	529	48345

7. Training and Testing Sets

The training and test sets are obtained at random when running train.py, using the `train_test_split` method of `scikit-learn`, a Python-based machine learning package. The method provides a parameter for the random seed called `random_state`, which allows for the same split, provided the same parameters. The `random_state` used was also 191.

The data set was split into the training set (80%) and the test set (20%), as such (with `random_state=191`):

	Train Set (80%)	**Test Set (20%)**
Total	58428	14608
Spam	38640 (66%)	9705 (66%)
Ham	19788 (34%)	4903 (34%)

Bayesian Classifier Construction

A NaiveBayes class is provided in naivebayes.py, and it can be initialized with the following parameters:

Initialization Parameters

Parameter	Type	Description
`name`	`string`	Name of the model
`X_train`	`list`	List of X values containing strings
`y_train`	`list`	List of y values corresponding to the class or target of the X value in X_train at the same index. Values must be contained in targets
`targets`	`list`	List of targets or classes
`target_names`	`list`	List of strings corresponding to the name of the target with the same list index
`max_features`	`None or int, defaults to None`	If None, all words are used, but if set, it will be used to reduce the vocabulary to the set value

Methods

Method	Description
`train()`	Trains the model, uses the provided `X_train` and `y_train` during model initialization
`predict(X_test)`	Returns two lists of the same length as X_test, corresponding to the prediction result without laplace smoothing, and with laplace smoothing (hence, two lists)
`evaluate(X_test, y_test, show_top_features=False)`	A more comprehensive version of `predict(X_test)`, which prints additional statistics, such as accuracy, precision, and recall. There is also the optional parameter, `show_top_features`, which accepts a boolean or an integer. If set to an integer, it will print the top n words provided the integer, based on number of counts. If set to True, it's the same as setting it to 10. If set to False, same as setting it to 0.

Training

The training algorithm is based on [3]'s algorithm.

Note that only the training set is read and processed here.

1. Obtaining the Bag of Words, Prior Probability, and other data of each Class

   Per class, all documents class as the label are processed, and the following are obtained:

   1. Bag of Words (BoW)

      Bag of Words is a structure which has the features (each unique word) and the total count of that feature across all the documents for the class.

      The structure uses a dictionary, with the word as the key, and the count as the value, as such:

      $$word : count_{word}$$

   2. Total count of all features in the BoW in the class (totalCount)
   3. Total number of features in the class (featureCount)
   4. Total number of documents in the class (docCount)
   5. Logarithmic Prior Probability (prior)

      Equal to the number of documents for the class divided by the total number of documents across all classes, then logarithmized. In other words:

      $$log(docCount_{class} / docCount_{total})$$

   For the reduced vocabulary models, all words are first counted, and the top 200 words are the 200 words with the highest word counts.

2. Obtaining the Likelihood Probabilities of each Feature

   Per feature per class, all documents class as the label are processed, and the following are obtained:

   1. Logarithmic Likelihood probability without Laplace smoothing

      $$likelihood = log(\frac{count_{word}}{totalCount_{class}})$$

   2. Logarithmic Likelihood probability with Laplace smoothing ($\alpha = 1$)

$$likelihood_{laplace} = log(\frac{count_{word} + 1}{totalCount_{class} + featureCount_{class}})$$

Note that these two likelihood probabilities are stored as two dictionaries, with a structure similar to BoW, where the key is the word, and the value is its corresponding likelihood probability (with and without Laplace smoothing).

Testing

For testing a document, two "total scores" are maintained:

- The total score of each class (without Laplace smoothing)
- The total laplace score of each class (with Laplace smoothing)

Note that both are each initially set to the prior probability of the corresponding class

For testing a document, each word in the document preprocessed, tokenized, then a score and a laplace score are computed for that specific word, whereas these score will be added to the corresponding total score for that class for the document.

Algorithm

```
def predict(X_test):
 predicted = []
 lap_predicted = []

 smooth_probability = []

 for c_idx, class in enumerate(classes):
   smooth_probability[c_idx] = math.log(1 / (class.doc_count + len(total_doc_count)))

 for document in X_test:
   total_score = []
   total_lap_score = []

   words = preprocess(document)

   for c_idx, class in enumerate(classes):
     total_score[c_idx] = class.prior
     total_lap_score[c_idx] = class.prior

     for word in words:
       if word in class.likelihood:
         total_score[c_idx] += class.likelihood[word]
         total_lap_score[c_idx] += class.lap_likelihood[word]
       else:
         for c2_idx, class2 in enumerate(class_data):
           if c2_idx != c_idx and word in class2.likelihood:
             total_score[c_idx] += smooth_probability[c_idx]
```

```
            total_lap_score[c_idx] += smooth_probability[c_idx]
            break

    max_score_idx = 0
    max_lap_score_idx = 0
    for i in range(1, len(total_score)):
      if total_score[i] > total_score[max_score_idx]:
        max_score_idx = i
      if total_lap_score[i] > total_lap_score[max_lap_score_idx]:
        max_lap_score_idx = i

    predicted.append(classes[max_score_idx])
    lap_predicted.append(classes[max_lap_score_idx])

  return predicted, lap_predicted
```

Note that this is a modified version from the source code for more readability, since some of the methods done use methods that aren't common in other languages (difficult to use as pseudocode). The original version can be viewed in the source code.

Evaluation

Using the previous predict() algorithm, the predicted and lap_predicted arrays are obtained. These two arrays are tested against the actual labels of the test set to compute for the accuracy.

$$accuracy = count_{correctly\ classified} / count_{all\ documents}$$

RESULTS

Training and Data Set

The data set was split into the training set (80%) and the test set (20%), as such (with `random_state=191`):

	Train Set (80%)	**Test Set (20%)**
Total	58428	14608
Spam	38640 (66%)	9705 (66%)
Ham	19788 (34%)	4903 (34%)

Training

Model	**Feature Count**	**Total Word Count**

General Vocab	327,697	9,798,466
Reduced Vocab	200	2,741,367

For vocabulary, here is the list of the top 10 words for both ham and spam:

Ham Words	**Ham Word Count**	**Spam Words**	**Spam Word Count**
list	17759	pill	49021
email	17520	per	39364
new	15160	desjardins	35020
would	14471	de	34998
please	13861	mg	28677
code	13483	price	28429
may	13227	item	25156
one	12306	le	24110
mailing	11877	save	21911
news	11779	product	20845

Testing (Prediction & Evaluation)

Model	**Accuracy**
General Vocabulary (Without Laplace)	89.95071%
General Vocabulary (With Laplace)	90.40252%
Reduced Vocabulary (Without Laplace)	87.06873%
Reduced Vocabulary (With Laplace)	87.06873%

For general vocabulary, the accuracy of the model with laplace smoothing is higher than the one without, but for reduced vocabulary, both are the same.

The researchers originally suspected that the reason laplace smoothing did not have much of an effect on the reduced vocabulary models is the reduced number of features. However, the researchers could not confirm their suspicious after training and testing additional reduced vocabulary models at 400, 600, 800, and 1000 words, which all had the same accuracies for both models with and without laplace smoothing.

Model	Accuracy
Reduced Vocabulary 200 (Without Laplace)	87.06873%
Reduced Vocabulary 200 (With Laplace)	87.06873%
Reduced Vocabulary 400 (Without Laplace)	86.67169%
Reduced Vocabulary 400 (With Laplace)	86.67169%
Reduced Vocabulary 600 (Without Laplace)	87.32202%
Reduced Vocabulary 600 (With Laplace)	87.32202%
Reduced Vocabulary 800 (Without Laplace)	89.06079%
Reduced Vocabulary 800 (With Laplace)	89.06079%

The researchers tried once more with a 1000 feature limit, and found that it resulted in a higher accuracy than the general vocabulary and also found a difference between the accuracies of the model with laplace smoothing and without laplace smoothing. It was also shocking to find that the model **without** laplace smoothing had a higher accuracy than the one with laplace smoothing.

Model	Accuracy
General Vocabulary (Without Laplace)	89.95071%
General Vocabulary (With Laplace)	90.40252%
Reduced Vocabulary 1000 (Without Laplace)	90.55312%
Reduced Vocabulary 1000 (With Laplace)	90.54628%

CONCLUSION

The Naive Bayes classifier with a reduced vocabulary of 1000 words without laplace smoothing had the highest accuracy of 90.55%, beating the classifier with all 327,697 words and laplace smoothing, which had 90.40% accuracy.

REFERENCES

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., Paliouras, G., & Spyropoulos, C. D. (2000). An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013*.

[2] C.D. Manning, P. Raghavan and M. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, p. 260.

[3] Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing* (Vol. 3). London: Pearson.