Email Classification of Enron Data Using Nieve-Bayes

|  |
| --- |
| Guillermo Rodriguez, Ph.D.  Department of Computer Science Texas Tech UniversityLubbock, United States [gui\_rod\_dallas@hotmail.com](mailto:gui_rod_dallas@hotmail.com) |

*The utilization of Nieve-Bayes is a practical mechanism by which relevance may be determined in a machine learning endeavor. In this investigative study this specific domain is applied to emails to determine its benefits and its merits for classification. The study highlights the practical importance of supervised learning.*

Nieve-Bayes, email, python, information retrieval, classification, machine learning, supervised learning

# Introduction

Nieve-Bayes is a supervised machine learning doctrine that seeks to place order to the unknown. Nieve-Bayes has its roots in Bayes’ theorem of probability. Bayes’ theorem has a central tenant at its core and this being that there is strong independence between the features in the data set. This is the reason why the theorem is referred to as nieve as in reality there could exist a correlation between the feature sets of the data.

The theorem of Nieve-Bayes seeks to determine the posterior probability of some feature; the formula of which is defined below. The probability statement defines the posterior probability as the ratio of the probability of the class multiplied by the likelihood measure divided by the probability of the event.

|  |  |
| --- | --- |
|  | (1) |

While simple in nature and relatively simple to implement in a programming structure such as Python it does have limitations and these being susceptibility to skewness in the data set, data scarcity, and the erroneous assumption of strong independence between features. Nieve-Bayes is a widely adopted mechanism in machine learning endeavors despite its drawbacks.

The complete source code for this body of work inclusive of the data files used may be obtained from an open source code repository that may be accessed here: <https://github.com/guillermorodriguez/nieve-bayes>.

# Data set

## Enron Emails

The data was provided by a freely available data store that may be found here: <http://www2.aueb.gr/users/ion/data/enron-spam/>. While the data set does provide the pool of documents categorized by domain they are also available in raw form and may be downloaded from the link provided above. You may also view the data set in the Github.com code repository link provided previously in this paper.

The data set is in free text format in the English language, which allows for the processing using freely available open source language libraries that have a footprint in Python through external components.

Given the requirements for this body of work and that the class component be made up of at minimum three labels of which only two were provided by the data repository utilized it became essential to take the non-spam emails and create a subsequent repository. This data repository has been labelled as other in the data folder all of which you may find via the code repository link provided previously.

The data set was broken down into learning and non-learning data by dividing the available data pool by classification index at a cutoff of 10%. 10% of the data was used for training and the remaining data was used for classification. The formula used to divide the data pool for spam data in Python code is given as formula two (2) below. The cutoff variable is the 10% cutoff mark in the dictionary values.

|  |  |
| --- | --- |
| \_spam\_token\_test = \_statistics.tokenization(' '.join(spam[:cutoff])) | (2) |

For the data set it was essential to determine the probability of the document type. This calculation was derived for each of the class types using the formulas given below.

|  |  |
| --- | --- |
| P(spam) = spam documents / total documents = 0.290698 | (3) |
| P(not spam) = not spam documents / total documents = 0.267442 | (4) |
| P(other) = other documents / total documents = 0.441860 | (5) |

The calculations given above was utilized as part of the overall posterior probability calculation as given in equation 1 detailed above.

## Processing

The data set was processed with the Python programming language, which may be obtained through the Python website <http://python.org>. The version of the programming language that was used to compile and execute the code base was 3.4.3 and the interactive development environment (IDE) that was used to create the Python code was Atom, which may be obtained here: <https://github.com/atom/atom/releases/tag/v1.32.2>.

The application can be executed from the command line by executing a command sequence of the form

python start.py [parameter set]

The algorithm utilizes class structures and divides the bulk of the work effort between two main classes a statistics class and a graph class. The statistics class has the burden of performing all the needed calculations and the graph class as its namesake specifies has the burden of creating the graphical elements of the output.

The algorithm provides a series of outputs to the standard console including tokenization of the data files, class probabilities, and conditional probabilities.

The plot is generated as an image using a python library called plotly that may be obtained here: <http://plotly.ly>.

# calculations

The calculation of the posterior probability was broken down into components that when combined together allowed for derivation of the value needed. In this process the task was broken down into three (3) base modules of functions and they were tokenization of the data, the derivation of the class probability, and the determination of the final probability.

## Tokenizaiton

The processing of determining worth by way of the Naïve-Bayes method first entails that the data set must be tokenized. This was achieved by way of the tokenization function in the statistics class file. The function definition is given below.

def tokenization(self, data):  
 result = {}

for \_word in data.split(' '):

\_word = \_word.strip()

if \_word.encode('utf-8') in result:

result[\_word.replace('\n', '').encode('utf-8')] += 1

else:

result[\_word.replace('\n', '').encode('utf-8')] = 1

if \_word.encode('utf-8') not in self.vocabulary:

self.vocabulary.append(\_word.replace('\n', '').encode('utf-8'))

return result

## Probability of Value in Class

The determination of the class probability followed, which is a central component of the posterior probability. The algorithm utilized to derive this value is given below.

def probabilityOfValueInClass(self, data):

result = {}

class\_count\_total = 0

for value in data.values():

class\_count\_total += value

for key, value in data.items():

result[key] = ( value + 1 ) / ( class\_count\_total + len(self.vocabulary))

return result

## Posterior Probability

The final step in the process was putting all of the elements together to determine posterior probability. This metric was calculated using the function definition given below.

def posteriorProbability(self, data, class\_probability, spam, not\_spam, other):

result = {}

for key, value in data.items():

denominator = 0

if key in spam:

denominator += spam[key]

if key in not\_spam:

denominator += not\_spam[key]

if key in other:

denominator += other[key]

denominator /= ( sum(spam.values()) + sum(not\_spam.values()) + sum(other.values()) )

if denominator == 0:

denominator = 1

result[key] = ( value \* class\_probability ) / denominator

return result

## Performance

The overall system performance calculation was given to one function definition and it is given below. You will notice that this function returns a triplet, precision, recall, and f measure.

def performance(self, true\_positive, false\_positive, false\_negative):

precision = true\_positive / ( true\_positive + false\_positive )

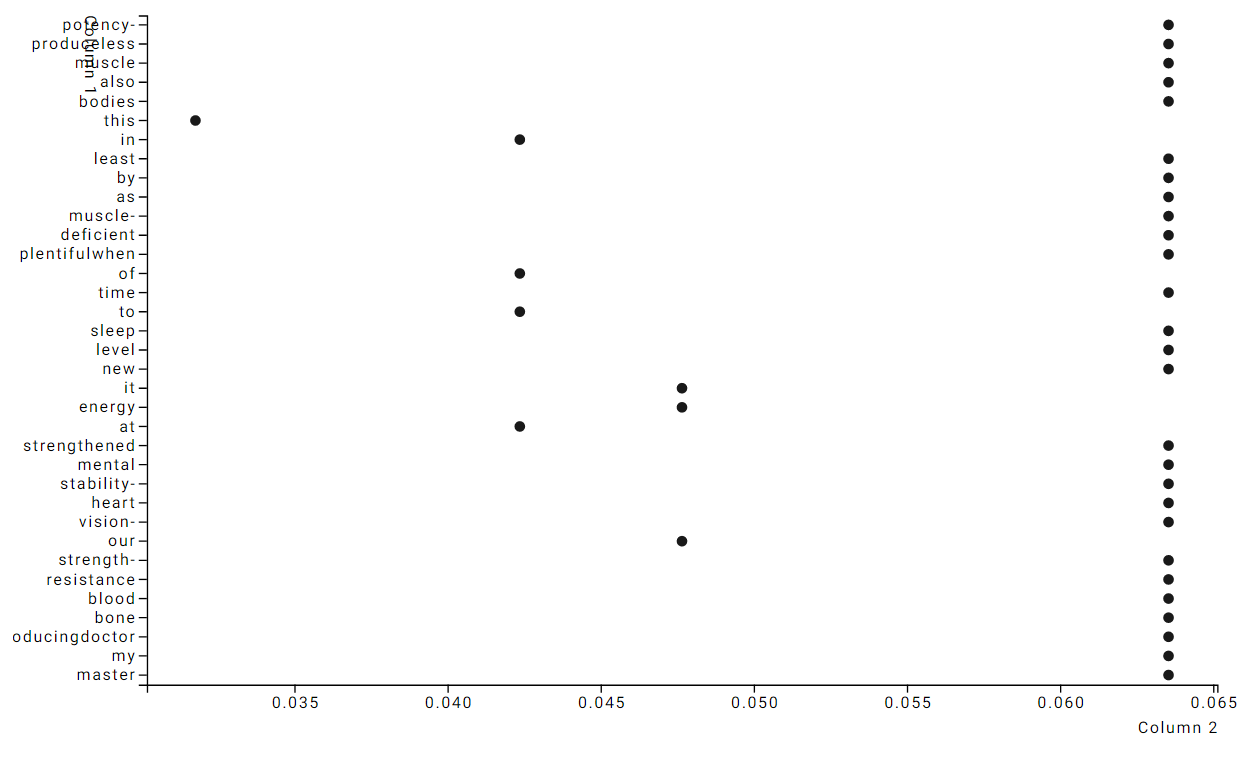
recall = true\_positive / ( true\_positive + false\_negative )

f\_measure = 2 \* precision \* recall / ( precision + recall )

return precision, recall, f\_measure

# Conclusion

A plot of the posterior probabilities for a query set is given below. Please do note that this is simply a sample data set for display purposes given the space constraint.



What is highlighted here as an end product is the resourcefulness that a development context such as Python affords to a statistical theorem such as Nieve-Bayes and the shear benefit that can result from simply applying a coding effort to a machine learning construct. Nieve-Bayes is not a difficult metric to calculate, but it can be challenging given a large data volume. This constraint is significantly moderated through Python and its rich libraries.

##### References

1. C. Manning, P. Raghavan, H. Schutze, Introduction to Information Retrieval. Cambrige: University Printing House, 2008.
2. V. Metsis, I. Androutsopoulos and G. Paliouras, Spam Filtering with Naive Bayes - Which Naive Bayes, Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006