

# Homework 1: Supervised Learning: Classification: Navie Bayes

- Explain the problem
- Explain the AI Model & Data
- Explain/Analysis the results

## Explain the problem:

Using of Naive Bayes is to try and detect [spam emails](#). Spam email is annoying and can lead to open backdoor, fishing and other harmful to the users. Detecting ontime can save user to protect their data, freedom, privacy and financial.

## Data Source:

I will be using dataset that of emails from the [Enron Corporation](#), an accounting firm that [went bankrupt in 2001 due to an accounting scandal](#).

This is one project I build when studied at AI Academy (NCSU.edu)

```
In [1]:  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn import datasets  
  
# set a seed for reproducibility  
random_seed = 25  
np.random.seed(random_seed)
```

## 1) Exploratory data analyst:

```
In [2]: df = pd.read_csv("./enron_emails.csv")
```

```
In [4]: df
```

	label	label_num	text
0	ham	0	Subject: enron methanol ; meter # : 988291\nth...
1	ham	0	Subject: hpl nom for january 9 , 2001\n( see a...
2	ham	0	Subject: neon retreat\nho ho ho , we ' re arou...
3	spam	1	Subject: photoshop , windows , office . cheap ...
4	ham	0	Subject: re : indian springs\nthis deal is to ...
...	...	...	...

label	label_num	text
5166	ham	0 Subject: put the 10 on the ft\nthe transport v...
5167	ham	0 Subject: 3 / 4 / 2000 and following noms\nhpl ...
5168	ham	0 Subject: calpine daily gas nomination\n>\n>\n...j...
5169	ham	0 Subject: industrial worksheets for august 2000...
5170	spam	1 Subject: important online banking alert\near ...

5171 rows × 3 columns

### Explain:

A ham email is a legitimate email, while a spam email is unwanted (in label columns).

```
In [5]: df.label.value_counts()
# There are 3672 legit email and 1499 spam email
```

```
Out[5]: ham    3672
spam    1499
Name: label, dtype: int64
```

### Explain:

Since the data is label. we are going to use supervised learning, classification naives bayes to solve the problem. and our model will be Bag-of-words model

```
In [6]: # Let's explore some of the ham emails...
print(df[df["label"]=="ham"].text.iloc[3])
```

Subject: re : indian springs  
 this deal is to book the teco pvr revenue . it is my understanding that teco just sends us a check , i haven ' t received an answer as to whether there is a predetermined price associated with this deal or if teco just lets us know what we are giving . i can continue to chase this deal down if you need .

```
In [7]: # And now the spam emails...
print(df[df["label"]=="spam"].text.iloc[18])
```

Subject: back  
 emile (
 the cablefilterz will allow you to receive  
 all the channels that you order with your remote control ,  
 payperviews , axxxmovies , sport events , special - events !  
 http : / / www . 8006 hosting . com / cable /  
 avocation , despoil .

## 2) AI Model: Bag-of-words:

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier.[2]

Retrieved from [https://en.wikipedia.org/wiki/Bag-of-words\\_model](https://en.wikipedia.org/wiki/Bag-of-words_model)

## CountVectorizer from sklearn:

We also use The CountVectorizer. it's fit\_transform method returns a NxM matrix. N is the number of documents (sentences) you have in your corpus, and M is the number of unique words in your corpus. Item n,m is how many times word m appears in document n.

## Build the model:

Build a Naive Bayes Classifier and evaluate it on a train and test set. In this instance, Multinomial Naive Bayes classifier, which is most useful for discrete features that use frequency counts (e.g. a bag of words vector).

```
In [12]: from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
In [25]: # Drop the Label so we not have any biased
df = df.drop(['label'],axis = 1)
```

```
In [26]: df.head()
```

	label_num	text
0	0	Subject: enron methanol ; meter # : 988291\nth...
1	0	Subject: hpl nom for january 9 , 2001\n( see a...
2	0	Subject: neon retreat\nho ho ho , we ' re arou...
3	1	Subject: photoshop , windows , office . cheap ...
4	0	Subject: re : indian springs\nthis deal is to ...

```
In [27]: # Create training and test splits
train, test = train_test_split(df, test_size=0.2, random_state=random_seed)

# Since this is supervised learning, we need to split the data into train and test data
# train data set will contain Label (ham/spam or 0/1)
# test data will not contain Label
```

```
In [28]: # Vectorize on your training data using Bow
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(train.text)
```

```
In [29]: # Fit the classifier below
clf = MultinomialNB()
clf.fit(X,train.label_num)
```

```
Out[29]: MultinomialNB()
```

```
In [30]: # Vectorize your test data using transform and then predict the test data
test_vecs = vectorizer.transform(test.text)
predictions = clf.predict(test_vecs)
```

```
In [31]: # Print a confusion matrix
confusion_matrix(test.label_num,predictions)
```

```
Out[31]: array([[728,    6],
   [ 11, 290]], dtype=int64)
```

```
In [32]: # Print a classification report
print(classification_report(test.label_num,predictions))
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	734
1	0.98	0.96	0.97	301
accuracy			0.98	1035
macro avg	0.98	0.98	0.98	1035
weighted avg	0.98	0.98	0.98	1035

### Result:

- We just created a model with the precision of 98% on predicting spam email (precision score)
- I will explain further below on how we can do that

### Technical Note: Log Probabilities:

When using probabilistic methods with large datasets, sometimes you get features with extremely small probabilities (e.g.  $10^{-10}$ ).

This becomes a problem, because computers aren't really good at doing operations with numbers at this scale. Therefore, in most systems, operations are done on the *log* of the probabilities.

This makes calculations much more manageable (e.g.  $\log(10^{-10}) = -10$ ). As an added bonus, due to log rules ( $\log(ab) = \log(a) + \log(b)$ ), all multiplications turn into additions, which are easier for the computer.

Some general rules of thumb: **the closer to zero a log prob is, the more probable it is**, and **each time a log prob decreases by one, it's an order of magnitude less probable**.

## Exploring Important Words:

```
In [33]: # Given that a message is ham, how probable is it for the words to show up?  
clf.feature_log_prob_[0]
```

```
Out[33]: array([-5.75023447, -5.71398455, -11.28296498, ..., -13.07472445,  
-13.07472445, -13.07472445])
```

```
In [34]: # Given that a message is SPAM, how probable is it for the words to show up?  
clf.feature_log_prob_[1]
```

```
Out[34]: array([-6.23363088, -7.07284789, -9.8691907, ..., -11.74099287,  
-11.74099287, -11.74099287])
```

```
In [36]: spam_args = np.argsort(clf.feature_log_prob_[1])  
spam_words = np.array(vectorizer.get_feature_names())[spam_args]  
spam_words = np.flip(spam_words)  
  
#Is this flipped or is this wrong  
ham_args = np.argsort(clf.feature_log_prob_[0])  
ham_words = np.array(vectorizer.get_feature_names())[ham_args]  
ham_words = np.flip(ham_words)
```

```
In [39]: # Since we're operating on logs, division turns into subtraction  
log_odds = clf.feature_log_prob_[1] - clf.feature_log_prob_[0]  
spam_ham_args = np.argsort(log_odds)  
spam_ham_words = np.array(vectorizer.get_feature_names())[spam_ham_args]  
spam_ham_words = np.flip(spam_ham_words)
```

```
In [40]: top_x=200  
spam_ham_words[0:top_x]
```

```
Out[40]: array(['td', 'nbsp', 'pills', 'width', 'computron', 'br', 'font', 'href',  
'viagra', 'height', 'xp', 'src', '2004', 'cialis', 'soft', 'meds',  
'paliourg', 'php', 'voip', 'drugs', 'oo', 'valign', 'bgcolor',  
'biz', 'hotlist', 'moopid', 'div', 'photoshop', 'mx', 'img',  
'knle', 'pharmacy', 'gr', 'intel', 'corel', 'prescription', 'iit',  
'demokritos', 'rolex', 'xanax', 'macromedia', 'dealer',  
'uncertainties', 'valium', 'htmlimg', 'darial', '000000',  
'0310041', 'lots', 'projections', 'jebel', 'adobe', 'rnd', 'color',  
'alt', '161', 'colspan', 'pain', 'readers', 'rx', 'canon',  
'export', 'draw', 'fontfont', 'gra', 'speculative', '1226030',  
'gold', 'pro', 'logos', 'wi', 'toshiba', 'china', '1933', 'spam',  
'vicodin', 'itoy', 'viewsonic', 'ooking', '1618', 'cellpadding',  
'weight', 'hewlett', '4176', 'pill', 'robotics', 'soma',  
'resellers', '8834464', '8834454', 'apc', 'intellinet', 'aopen',  
'iomega', 'enquiries', 'customerservice', 'targus', 'packard',  
'tr', 'uae', 'dealers', 'spain', 'nomad', '1934', 'drug', 'muscle',  
'abdv', 'zonedubai', 'eogi', 'aeor', 'doctors', 'inherent',  
'wysak', 'emirates', 'cheap', 'health', 'border', 'illustrator',  
'hottlist', 'oem', 'apple', 'fffffff', 'ce', 'verdana', 'sex',  
'gif', 'resuits', 'graphics', 'mining', 'studio', 'differ',
```

```
'materia', 'predictions', 'arial', 'waste', 'cellspacing', 'yap',
'male', 'phentermine', 'tirr', 'cf', 'wiil', 'construed', 'otcbb',
'atleast', 'materially', 'kin', '2005', 'vi', 'anticipates',
'erections', 'artprice', 'deciding', 'featured', 'prescriptions',
'sofftwaares', 'ali', 'ur', 'sir', 'discreet', 'gains', 'dose',
'cia', 'assurance', 'distributorjebel', 'nigeria', 'spur',
'serial', 'ambien', 'creative', 'align', 'stocks', 'aerofoam',
'der', 'penis', 'emerson', 'bingo', 'fffffffstrongfont', 'mai',
'style', 'anxiety', 'brbr', 'prozac', 'undervalued', 'epson',
'fontbr', 'notebook', 'levitra', 'es', 'iso', 'risks', 'alcohol',
'xm', 'erection', 'lasts', 'effects', 'vlagra', 'technoiogies',
'124', 'couid'], dtype='<U24')
```

In [ ]: