Probabilistic Learning

Naive Bayes Classifiers

Probability-based predictions

- Weather forecast says things like “70% chance of rain”
- They express probability of precipitation
- How are these probability calculated?

Use of past data

- Probability-based prediction use past data to produce predictions for future events
- E.g. in the past, under similar conditions, in 7 out of 10 cases it rained

Naive Bayes algorithm

- Originates from the work of the mathematician Thomas Bayes (18th century)
- Classifiers based on Bayesian methods use training data to compute the probability of each outcome based on evidence from feature values

Applications

- Text classification
- Intrusion/anomaly detection in computer networks
- Diagnosis of medical conditions from symptoms

Where should you apply it?

- Many features
- Even if some features alone have little effect, their combination can have a quite large effect on the model

Event, trials

Bayesian probability theory estimates the likelihood of an **event** based on the evidence of multiple **trials**

Event	Trial
Head result	Coin flip
Rainy weather	A single day’s weather
Message is spam	Incoming email message
Candidate becomes president	Presidential election
Win the lottery	Lottery ticket

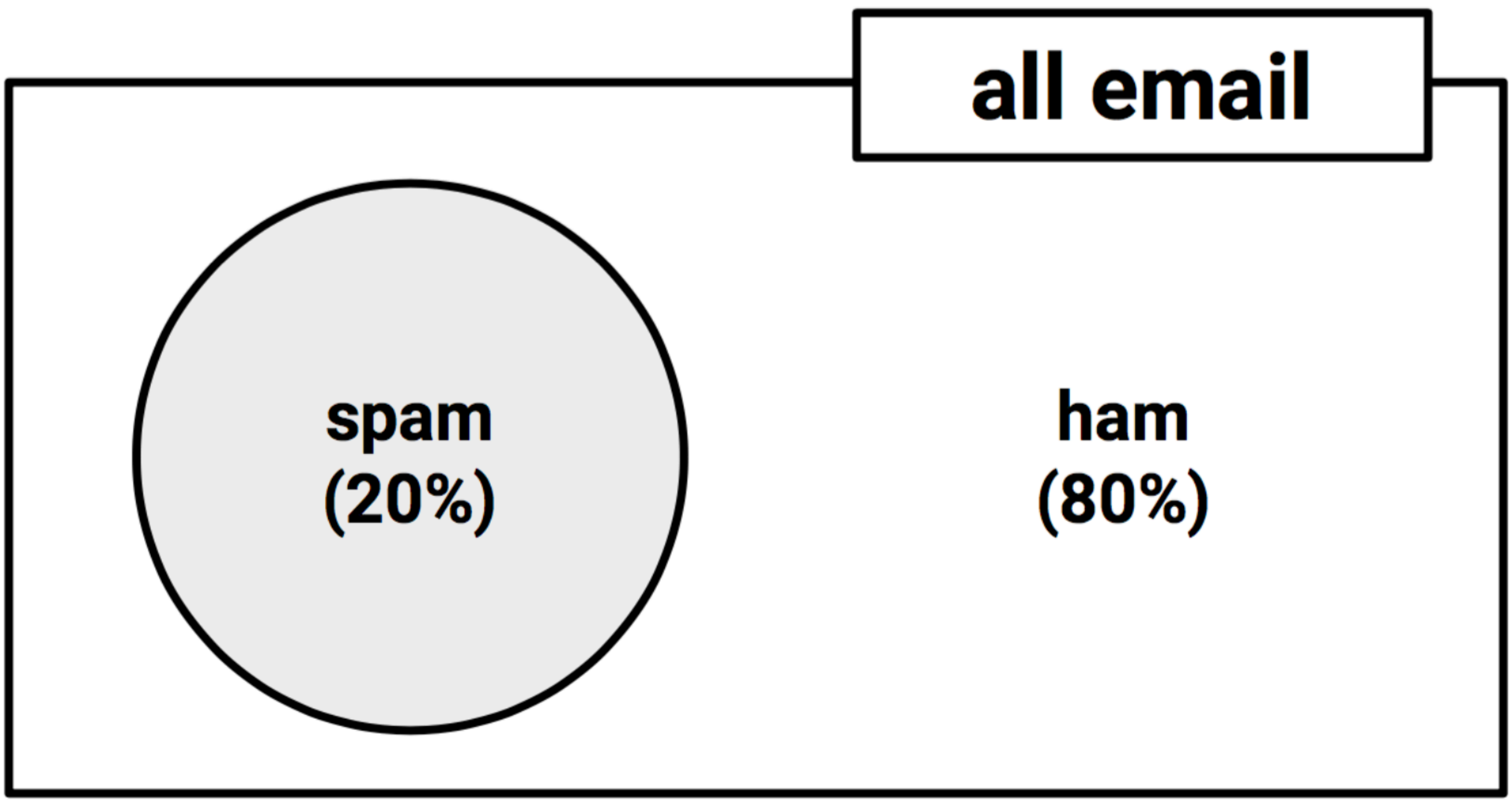
From trials to probability

- Probability is estimated by dividing the number of trials in which the event occurred by the total number of trials
- If 10 out of 50 similar messages contained spam, the the probability of spam = 10/50 = 0.20 or 20%
- If it rained 3 out of 10 days under similar conditions, the raining probability is 3/10=0.30 or 30%

Complementing probability

- The cumulative probability of all outcomes is 1
- if P(spam)=0.20, then P(ham)=1-0.20=0.80
- Spam and Ham are **mutually exclusive** events (i.e., they cannot occur at the same time)
- Spam and Ham are **exhaustive**, i.e., together they represent all possible outcomes
- Notation for complement: A’ or A^c
- Probability notation: P(A’) or $P(A^c)$

Spam and its complement

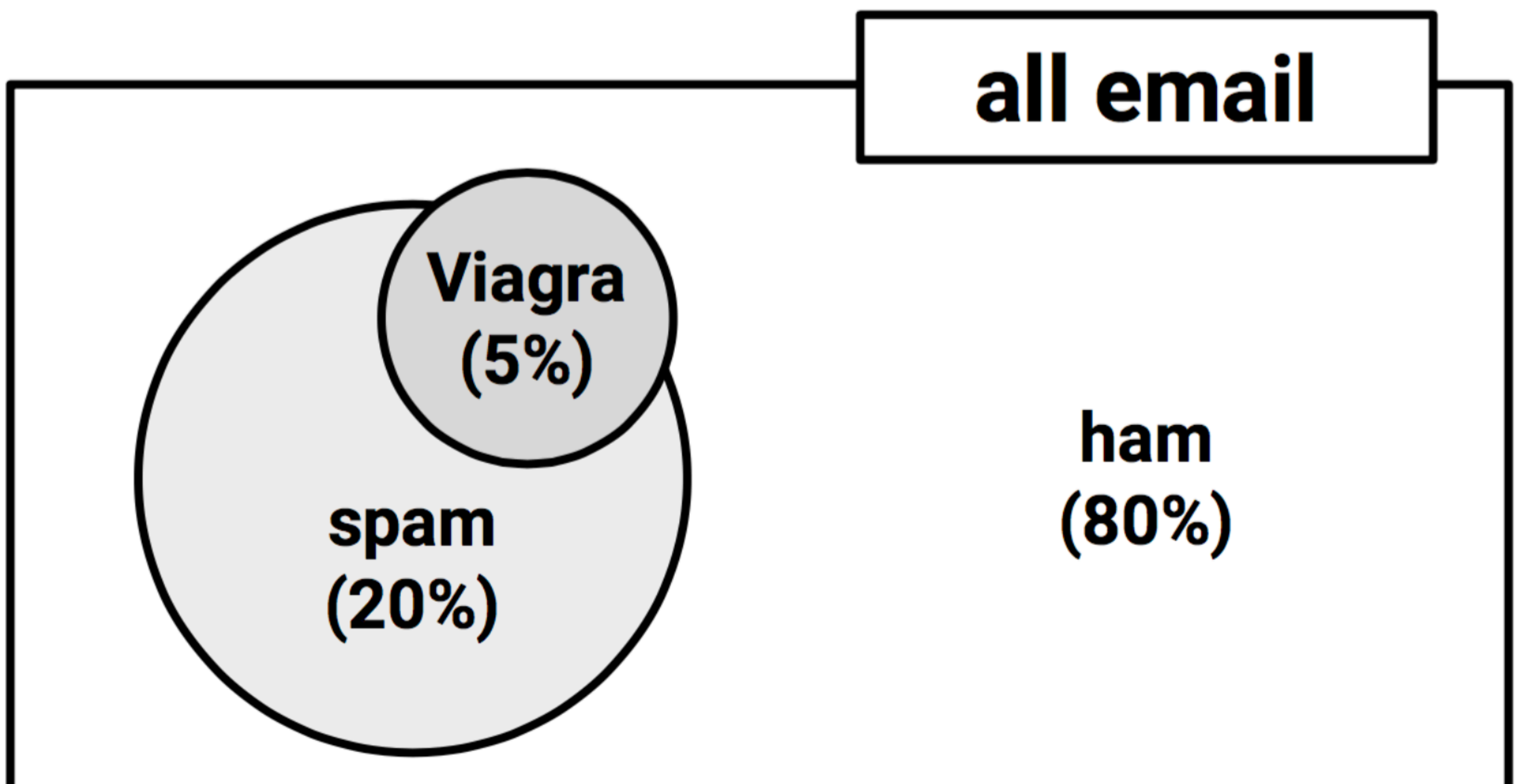


Joint probability

- Often we are interested to monitor several non-mutually exclusive events for the same trial
- If they occur concurrently, we may exploit them to make predictions

Example

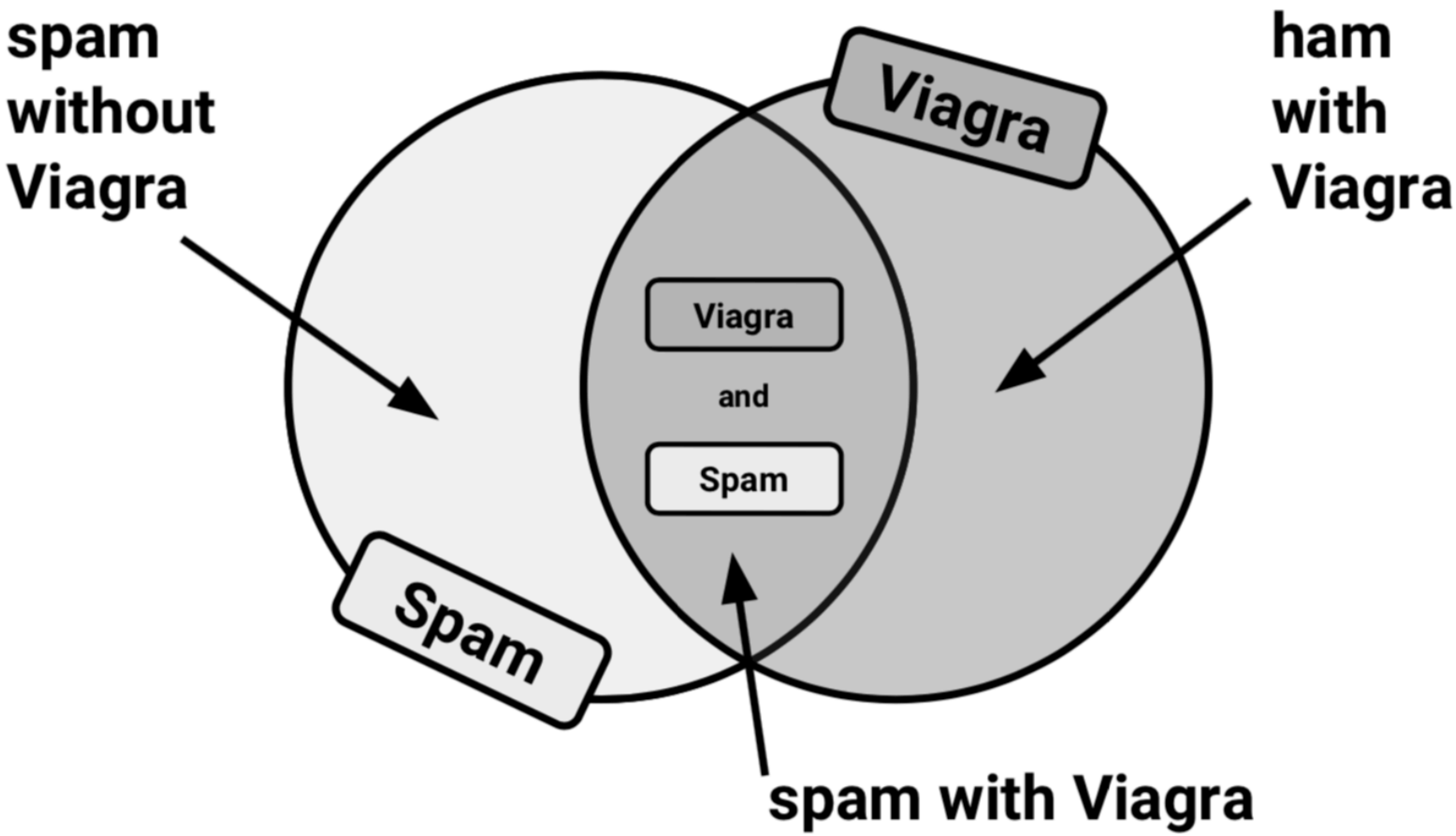
- Let's consider all email messages that contain the word "Viagra"
- Are they spam?



Example - discussion

- Not all spam contain the word “Viagra”
- Not all messages with the word “Viagra” are spam
- However, the diagram shows that in “most of the cases” the word “Viagra” appears for spam
- Can we quantify that?

Venn diagram



Facts

- 20% of all messages are spam
- 5% of all message contain the word “Viagra”
- What’s the overlap?

Joint probability

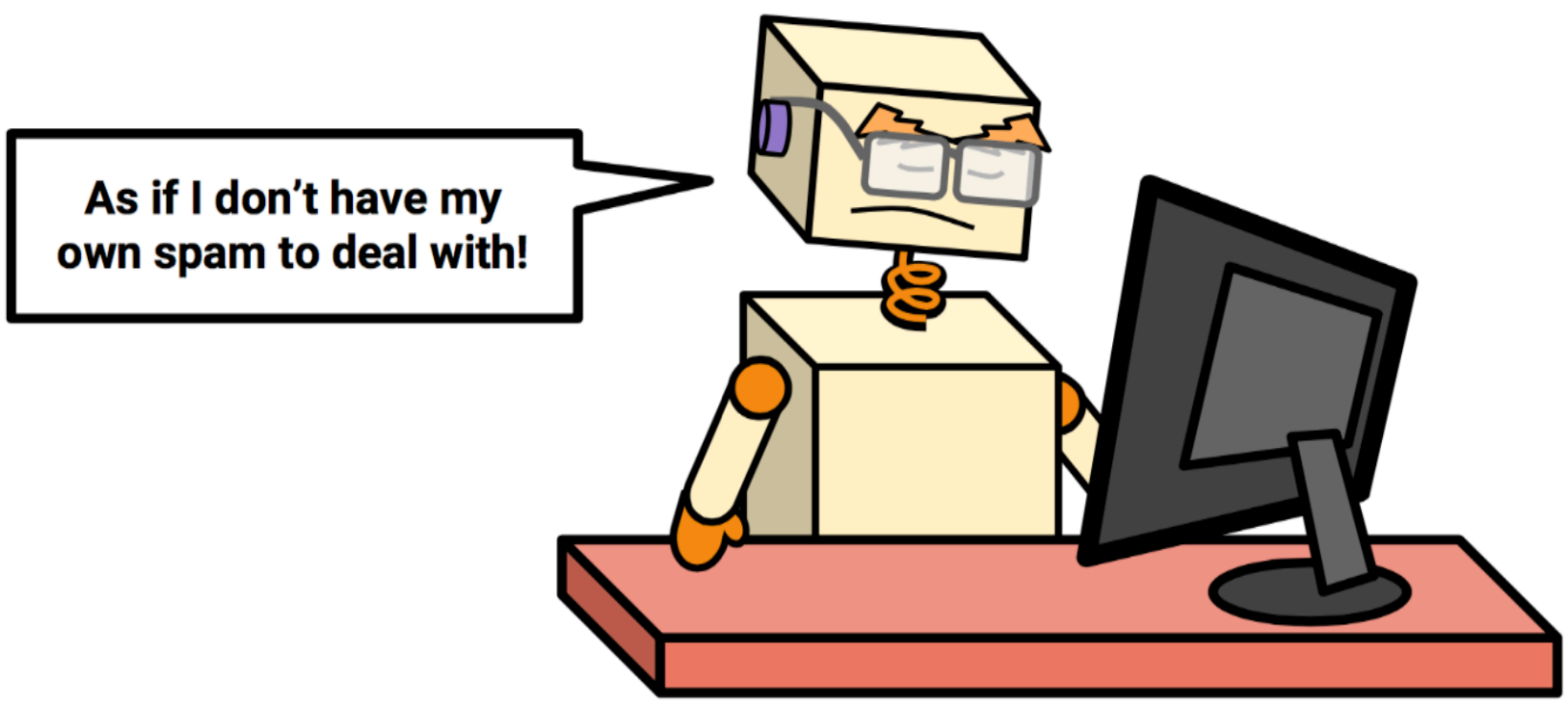
- Probability that both spam and Viagra occur
- Written as P(spam ∩ Viagra)
- Calculating the joint probability depends on whether events are **independent** or **related**

Independent events

- **Independence** means that knowing the outcome of one event does not provide any information about the other
- For example:
    - the outcome of a coin flip is independent from whether it is rainy or sunny on any given day
    - the outcome of a coin flip is independent from previous coin flips
- If all events are independent, it would be impossible to predict one event by observing another
- Therefore, we exploit **dependent** events for predictive modeling

Dependent events

- The presence of cloud is predictive of a rainy day
- The word Viagra may be predictive of a spam email



Probability of dependent events

- If P(spam) and P(Viagra) were independent, P(spam ∩ Viagra) would have been the probability of both events happening at the same time, i.e.

  $P(A \cap B)= P(A) * P(B)$

- In our case:

  $P(\text{spam} \cap \text{Viagra})=P(\text{spam})*P(\text{Viagra})=0.20 * 0.05 = 0.01$

However…

In our case, the events are likely to be dependent, therefore this calculation is incorrect

The Bayes' theorem

- Estimates the probability of an event conditioned by the evidence provided by another:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

- P(A|B) is said **conditional probability**

Bayes' Theorem

- $P(A \cap B) = P(A \mid B) * P(B)$
- also, since $P(A \cap B) = P(B \cap A)$:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B)}$$

Prior probability, likelihood, marginal likelihood

- P(spam) is based on the evidence that previous messages are spam, and is known as **prior probability**
    - This is 20% in our case
- The probability that the word "Viagra" was used in previous spam messages, P(Viagra|spam) is called the **likelihood**
- The probability that "Viagra" appeared in any message at all, P(Viagra) is called **marginal likelihood**

Probability vs. Likelihood

- The likelihood means increasing the chances of a particular situation to occur by varying a certain distribution (i.e., how would the chances of a message to be spam increase if the word “Viagra” is there)?

- Probability simply gives you a view of the current distribution, indicating the chances of something to happen.

Posterior probability

- We can compute a posterior probability that a message is spam given that it contains the word Viagra:

$$\underbrace{P(spam|Viagra)}_{\text{Posterior Probability}} = \frac{\overbrace{P(Viagra|spam)}^{\text{Likelihood}}\overbrace{P(spam)}^{\text{Prior Probability}}}{\underbrace{P(Viagra)}_{\text{Marginal Likelihood}}}$$

Calculating the components of the Bayes theorem

We first construct a frequency table

	Viagra		
Frequency	**Yes**	**No**	**Total**
spam	4	16	20
ham	1	79	80
Total	5	95	100

Calculating the components of the Bayes theorem

Then, we derive a likelihood table from it, computing the conditional probabilities

	Viagra		
Frequency	Yes	No	Total
spam	4	16	20
ham	1	79	80
Total	5	95	100

Likelihood table

	Viagra		
Frequency	Yes	No	Total
spam	4	16	20
ham	1	79	80
Total	5	95	100

	Viagra		
Likelihood	Yes	No	Total
spam	4 / 20	16 / 20	20
ham	1 / 80	79 / 80	80
Total	5 / 100	95 / 100	100

Likelihood table: discussion

	Viagra		
Likelihood	Yes	No	Total
spam	4 / 20	16 / 20	20
ham	1 / 80	79 / 80	80
Total	5 / 100	95 / 100	100

- P(Viagra= Yes|spam)=4/20=0.20, i.e. 20% probability that a message contains "Viagra" when it is spam

- P(spam ∩ Viagra)= P(Viagra|spam) * P(spam) = (4/20)*(20/100)=0.04

  This means that 4 out of 100 spam messages were spam with the term viagra
  This is 4 times higher than 0.01, the previous combined probability we computed under the false assumption of independence

Computing the posterior probability

P(spam|Viagra)=P(Viagra|spam)*P(spam)/P(Viagra) = (4/20)*(20/100)/(5/100)=0.80

- Therefore there is 80% probability that a message is spam given that it contains the word “Viagra”
- Therefore, we can use such a word as a reliable predictor
- This is more or less like modern spam filters work…

The Naive Bayes Algorithm

Main assumption

- it based on the “naive” assumption that **all features in the data are equally important and independent**
- This is rarely true in real-world
- For example, the email sender may be a more important indicator of spam than any word in the message text

Strengths and Weaknesses

Strengths	Weaknesses
Simple, fast, yet effective	The underlying assumption is often faulty
Works well with noisy and missing data	Not ideal if you have many numeric features
Works well with a small training, but also with large ones	Estimated probabilities less reliable than predicted classes
Estimated probability for a prediction easy to obtain	

What if we have multiple features for prediction?

- Before we only considered the term “Viagra”
- What if we also have other terms, such as “money”, “groceries”, “unsubscribe”
- Let’s compute a new likelihood table…

Likelihood table…

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Total**
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

Computing the posterior probability

- Let's call the words W_1, W_2, W_3, W_4

- For example, let's compute the probability that a massage is spam given that it contains: Viagra=Yes, Money=No, Groceries=No, Unsubscribe=Yes

$$P(spam \mid W_1 \cap W_2' \cap W_3' \cap W4) = \frac{P(W_1 \cap W_2' \cap W_3' \cap W_4 \mid spam)\ P(spam)}{P(W_1 \cap W_2' \cap W_3' \cap W_4)}$$

- This formula is computationally intensive to solve, you can imagine what happens if we add more words…

Moreover…

If an event (a word) was not observed in past data, it will result in a zero probability, and the product will become zero!

Using Naive assumption

- All predicting features are independent from each other
- Therefore, the numerator of our formula becomes a simple multiplication of probabilities rather than a complex, conditional joint probability
- Finally, the denominator does not depend on the target class (spam or ham), therefore it can be treated as a constant value and hence ignored

Therefore

- The conditional probability of spam is:

$$P(spam \,|\, W_1 \cap W_2' \cap W_3' \cap W_4) \propto P(W_1 \,|\, spam)\; P(W_2' \,|\, spam)\; P(W_3' \,|\, spam)\; P(W_4 \,|\, spam) P(spam)$$

- Similarly, the conditional probability of ham is:

$$P(ham \,|\, W_1 \cap W_2' \cap W_3' \cap W_4) \propto P(W_1 \,|\, ham)\; P(W_2' \,|\, ham)\; P(W_3' \,|\, ham)\; P(W_4 \,|\, ham) P(ham)$$

We ignore the denominator as it is independent on spam or ham

Applying it

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	Yes	No	Yes	No	Yes	No	Yes	No	Total
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

Viagra=Yes, Money=No, Groceries=No, Unsubscribe=Yes

- Overall likelihood of spam:
  (4/20)*(10/20)*(20/20)*(12/20)*(20/100)=0.012
- Overall likelihood of ham:
  (1/80)*(66/80)*(71/80)*(23/80)*(80/100)=0.002
- 0.012/0.002=6, hence this message is six times more likely to be spam than ham

Converting likelihoods to probabilities

- We must reintroduce the denominator that we ignored before
- The denominator is the sum of all possible likelihoods, i.e., likelihood of ham + likelihood of spam:
  - P(spam)=0.012/(0.012+0.002)=**0.857**
  - P(ham)=0.002/(0.012+0.002)=0.143

On summary

Given the found pattern, we expect the message to be spam with 85.7% probability

Generalized formula

The probability of level L for class C, given the evidence of features F_1, Fn, is equal to the product of

- probabilities of each piece of evidence conditioned on the class level
- the prior probability of the class level $p(C_L)$
- a scaling factor 1/Z which converts likelihood into probabilities

$$P(C_L \mid F_1, \ldots, F_n) = \frac{1}{Z} p(C_l) \prod_{i=1}^{n} p(F_i \mid C_L)$$

Laplace Estimator

Let’s assume we have a new message containing all four terms: “Viagra”, “groceries”, “money”, and “Unsubscribe”

Let's compute p(spam) and p(ham)

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	Yes	No	Yes	No	Yes	No	Yes	No	Total
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

- Using the likelihood table, we compute the likelihood of spam:
  (4/20)*(10/20)*(0/20)*(12/20)*(20/100)=0
- Similarly, the likelihood of ham:
  (1/80)*(14/80)*(8/80)*(23/80)*(80/100)=0.00005
- The probability of spam is:
  0/(0+0.00005)=0
- The probability of ham is:
  0.00005/(0+0.00005)=1

This doesn’t make sense!

- This message won’t be classified as spam, but its content may suggest it is spam, instead
- This problem arises if an event never occurred in our training set
- In our case the term “groceries” never appeared in spam messages, therefore P(spam|groceries)=0%
- Because probabilities in the Naive Bayes formula are multiplied, a zero for a feature results in a zero probability!

Solution:
the Laplace estimator

- Idea: add a small number to each frequency, so the multiplication would never become zero

- In practice, this can be any number

- For a large training set a Laplace value=1 is often used

Let’s recompute the number

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Total**
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

Since we have 4 features (4 words) we add 1 to all numerators of our formula, but we have to add 4 to the denominators

Let’s recompute the number

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Total**
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

- Likelihood of spam:
  (5/24)*(11/24)*(1/24)*(13/24)*(20/100)=0.0004
- Similarly, the likelihood of ham:
  (2/84)*(15/84)*(9/85)*(24/84)*(80/100)=0.0001
- The probability of spam is:
  0.0004/(0.0004+0.0001)=0.80
- The probability of ham is:
  0.0001/(0.0004+0.0001)=0.20

This makes more sense!

Note

- We added the Laplace estimator (1) to each numerator and to the denominators of the likelihood, we did not add it to the prior probabilities
- Therefore, they remained equal to 20/100 and 80/100
- This is because they are just computed based on observed data and on such adjustment is necessary

Using numeric features with Naive Bayes

- With Naive Bayes, we use frequency tables from learning over the data
- Therefore, each feature must be categorical to create the likelihood table

What to do with numeric features?

Solution: discretize

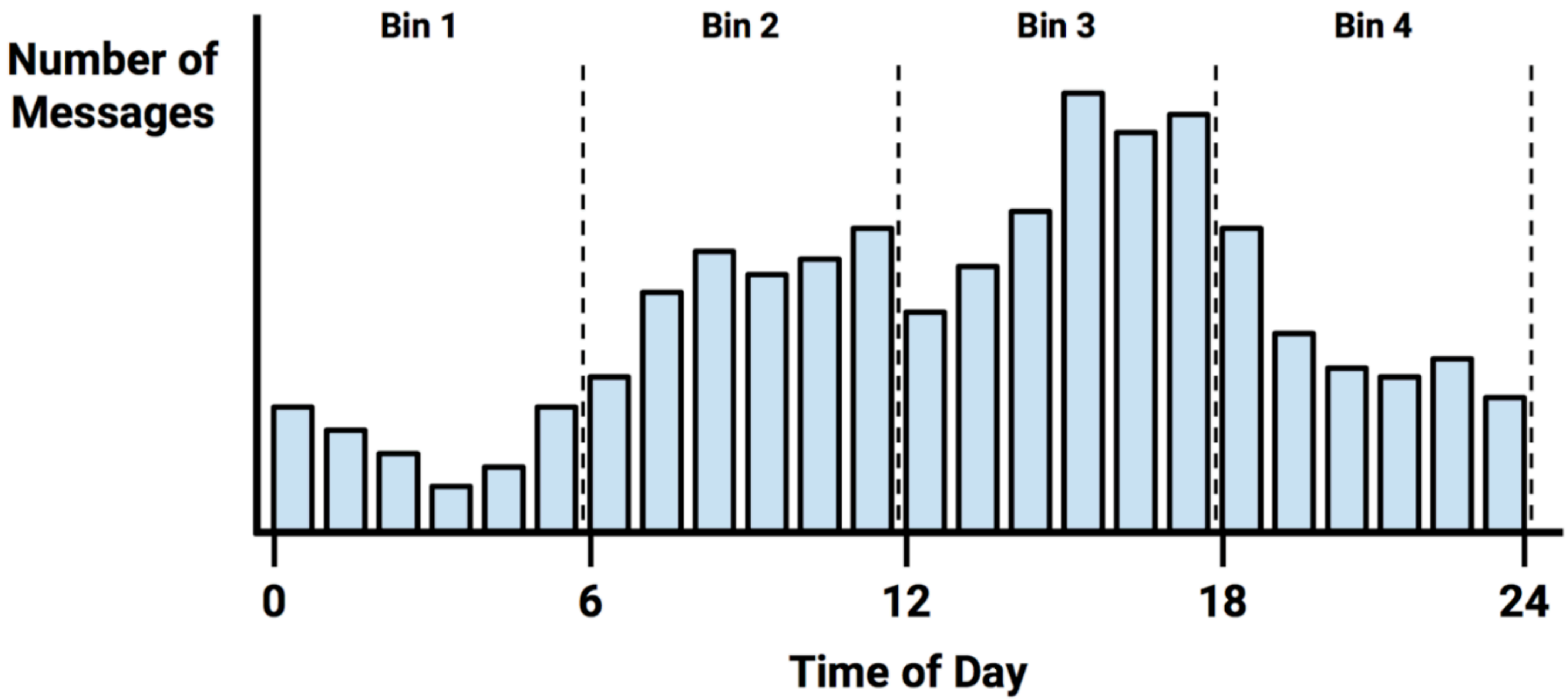
- Similar to what done in histograms, we split numerical data into bins and for each bin we create a different categories

- How to split into bins?

- Not really like histogram, but we can use natural splits (e.g., hours if the variable is time), or points where we observe the variable pattern changes quite a bit

Example

- Assuming we have the time of the day when emails arrive
- If we plot the histogram of this time:



Discussion

- We notice how the frequency of email varies between night, morning, afternoon, and evening
- Therefore, for each mail we might have four Boolean variables indicating the time band in which the email has arrived (only one of them will be True each time)

Example

Classifying spam

Dataset

- SMS spam collection: https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

Examples

Ham	Spam
Better. Made up for Friday and stuffed myself like a pig yesterday. Now I feel bleh. But at least its not writhing pain kind of bleh.	Congratulations ur awarded 500 of CD vouchers or 125gift guaranteed & Free entry 2 100 wkly draw txt MUSIC to 87066
If he started searching he will get job in few days. he have great potential and talent.	December only! Had your mobile 11mths+? You are entitled to update to the latest colour camera mobile for Free! Call The Mobile Update Co FREE on 08002986906

Reading and examining the dataset

```python
import pandas as pd
from nltk.corpus import stopwords
import gensim
import numpy as np


dataset=pd.read_csv("sms_spam.csv")

print(dataset.head())
print ("Shape:", dataset.shape, '\n')
```

Output

```
   type                                                   text
0   ham  Hope you are having a good week. Just checking in
1   ham                          K..give back my thanks.
2   ham        Am also doing in cbe only. But have to pay.
3  spam  complimentary 4 STAR Ibiza Holiday or £10,000 ...
4  spam  okmail: Dear Dave this is your final notice to...
Shape: (5559, 2)
```

Function for text preprocessing

- This time we will use the out-of-box features available in gensim

Preprocessing function

```
def transformText(text):
    stops = set(stopwords.words("english"))
    # Convert text to lowercase
    text = text.lower()
    # Strip multiple whitespaces
    text = gensim.corpora.textcorpus.strip_multiple_whitespaces(text)
    # Removing all the stopwords
    filtered_words = [word for word in text.split() if word not in stops]
    # Preprocessed text after stop words removal
    text = " ".join(filtered_words)
    # Remove the punctuation
    text = gensim.parsing.preprocessing.strip_punctuation(text)
    # Strip all the numerics
    text = gensim.parsing.preprocessing.strip_numeric(text)
    # Removing all the words with < 3 characters
    text = gensim.parsing.preprocessing.strip_short(text, minsize=3)
    # Strip multiple whitespaces
    text = gensim.corpora.textcorpus.strip_multiple_whitespaces(text)
    # Stemming
    return gensim.parsing.preprocessing.stem_text(text)
```

Preprocessing...

```python
#applies transformText to all rows of text
dataset['text'] = dataset['text'].map(transformText)
print(dataset['text'].head())
```

Output

```
0                                  hope good week check
1                                       give back thank
2                                    also cbe onli pai
3    complimentari star ibiza holidai cash need urg...
4    okmail dear dave final notic collect tenerif h...
Name: text, dtype: object
```

Creating training and test set

- In real life, you train your machine learning model on data for which you know the label (in this case “ham” or “spam”)
- When validating an algorithm, we have a dataset with labels everywhere
- We cannot train and test the algorithm on the same data!

Creating training and test set

- We split the dataset into 2 portions
- Training set: used to train the machine learning algorithm
- Test set: used to evaluate the algorithm’s performances

How?

- Simply sequentially, if the data has a temporal meaning (e.g. COVID data over time)
- Randomly
- Using cross-validation (we will see this later)
- Note: for now we will split training and test set once, in practice it needs to be done multiple times, and performances analyzed over multiple runs

Creating a random split using scikit-learn

```
## Split the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset['text'], dataset['type'],
                                                     test_size=0.33, random_state=10)

print ("Training Sample Size:", len(X_train), ' ', "Test Sample Size:" ,len(X_test))
```

Training Sample Size: 3724   Test Sample Size: 1835

Notes

- Note: the function takes as input the size of the test set and a random number seed
- Initializing the random number seed with a constant always produces the same results
- Initializing the latter with a timestamp makes your results always different

```
from sklearn.model_selection import train_test_split
import time
X_train, X_test, y_train, y_test = train_test_split(dataset['text'], dataset['type'],
                                                    test_size=0.33, random_state=int(time.time()))
```

Creating a tf-idf corpus for the training set

```
#Build the counting corpus
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)

## Get the TF-IDF vector representation of the data
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
print ('Dimension of TF-IDF vector :' , X_train_tfidf.shape)
```

Dimension of TF-IDF vector : (3724, 7007)

Creating the classifier

```
#Creating the classifier
#MultinomialNB accepts weights instead of Boolean
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
# the fit() function of any classifier takes the features from the
# training set X_train_tfidf and the labels from the training set
# y_train
clf.fit(X_train_tfidf, y_train)
```

Notes

- The fit() function applies to any classifier
- It takes as input:
    - the training set features
    - the labels from the training set

Notes

- Scikit-learn has different kinds of Naive Bayes classifier
  - BernoulliNB: accepts boolean as the examples we have seen previously
  - MultinomialNB: models counts
  - GaussianNB: useful for decimal values provided that they follow a normal distribution
- When creating the models, parameters with options are available, among others the alpha parameter specifies the value of the Lapace estimator (default=1)
  - `clf = MultinomialNB(alpha=0) # no Laplacian smoothing`
  - `clf = MultinomialNB(alpha=1) # smoothing=1`

Folding the test set into the vector space...

```
#indexing the test set
X_new_counts = count_vect.transform(X_test)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)
```

- The transform() function takes a set of words and folds them into a given vector space
- We first fold them into count_vect (obtained through the CountVectorizer) and then we fold the results into the tfidf_transformer vector space

Making the prediction

```
#performing the actual prediction
predicted = clf.predict(X_new_tfidf)
```

- the predict() function is a standard function of different scikit-learn models
- It has to be applied to previously-created model (clf)
- It takes as input the features of the test set, folded into the training set space

Showing the results

```
print(predicted)
print(np.mean(predicted==y_test))
```

```
['ham' 'ham' 'ham' ... 'spam' 'ham' 'spam']
0.9607629427792915
```

Note

- predicted==y_test compares every element of predicted (which can be “Ham” or “Spam”) with every element of the test set labels y_test (again, “Ham” or “Spam) and returns True or False

- np.mean() computes the fraction of the True over the total, i.e., like computing a mean of 0 and 1 values

- What we computed is called accuracy

- We will later formally define the performance measurements for information retrieval and machine learning

Exercises

- Try how different models produce different performance values
- For example:
  - Different values of the Laplace estimator
  - Using a BernoulliNB after having indexed the words using a Boolean model:
    `count_vect = CountVectorizer(binary=True)`
  - Filtering out words appearing rarely (try with different values
    `count_vect = CountVectorizer(min_df=10)`
  - Just using CountVectorizer instead of TfidfTransformer()

Coming up next…

- Evaluating the model
- Improving the model, by applying feature selection, or by using alternative machine learners
- Applying to sentiment analysis