S21CSE471: Statistical Methods in AI **Spring 2021**

Naive Bayes, BoW, tf-idf

Prepared by: Siddhant Bansal, Soham Choudhuri, Sitaram Tadepalli

1 Naive Bayes

In this section, we will learn about Naive Bayes while focusing on various tasks surrounding text.

1.1 Motivation

Many language processing tasks involve classification. We can use the Naive Bayes algorithm for many of the tasks surrounding text. For example,

1. **Text categorization**: Task of assigning a label or category to an entire text or document.
2. **Sentiment analysis**: Involves extraction of sentiment, the positive or negative orientation that a writer expresses toward some object.
3. **Spam detection**: It is a binary classification task of assigning an email to one of the two classes spam or not-spam.
4. **Language ID**: Detection of the language in which the text is written.
5. **Authorship attribution**: Determining a text's author.

Most of the above tasks are done using *supervised machine learning*. Formally, the task of supervised classification is to take an input x and a fixed set of output classes $Y = y1, y2, \dots, y_M$ and, return a predicted class $y \in Y$. For text classification, instead of y, c (for "class") is used as an output variable and, instead of x, d (for "document") is used as the input variable.

In the case of *supervised machine learning*, we have a training set of N documents which are hand-labelled with a class: $(d_1, c_1), \dots, (d_N, c_N)$. The goal is to learn a classifier that is capable of mapping from a new document d to its correct class $c \in C$. A **probabilistic classifier** additionally tells us the probability of an observation being in a class. Getting a distribution over the classes proves to be useful for downstream decisions. It helps to avoid making discrete decisions early on when combining different systems.

There are two different classification types:

1. Generative classifiers: Classifiers like naive Bayes, build a model of how a class could *generate* some input data.
2. Discriminative classifiers: Classifiers like logistic regression instead learn what features from the input are most useful to *discriminate* between the different possible classes.

1.2 Naive Bayes Classifiers

The classifier is called Naive Bayes as it makes a simplifying (*naive*) assumption about how the features interact. For applying Naive Bayes to a text document, the document is represented as if it was a

bag-of-words. As we can see in Fig. 1, we do not represent the word in a phrase like “The dialogue is great”, we simply consider the number of times *the* occurred in the document, which is 4 times in this case.

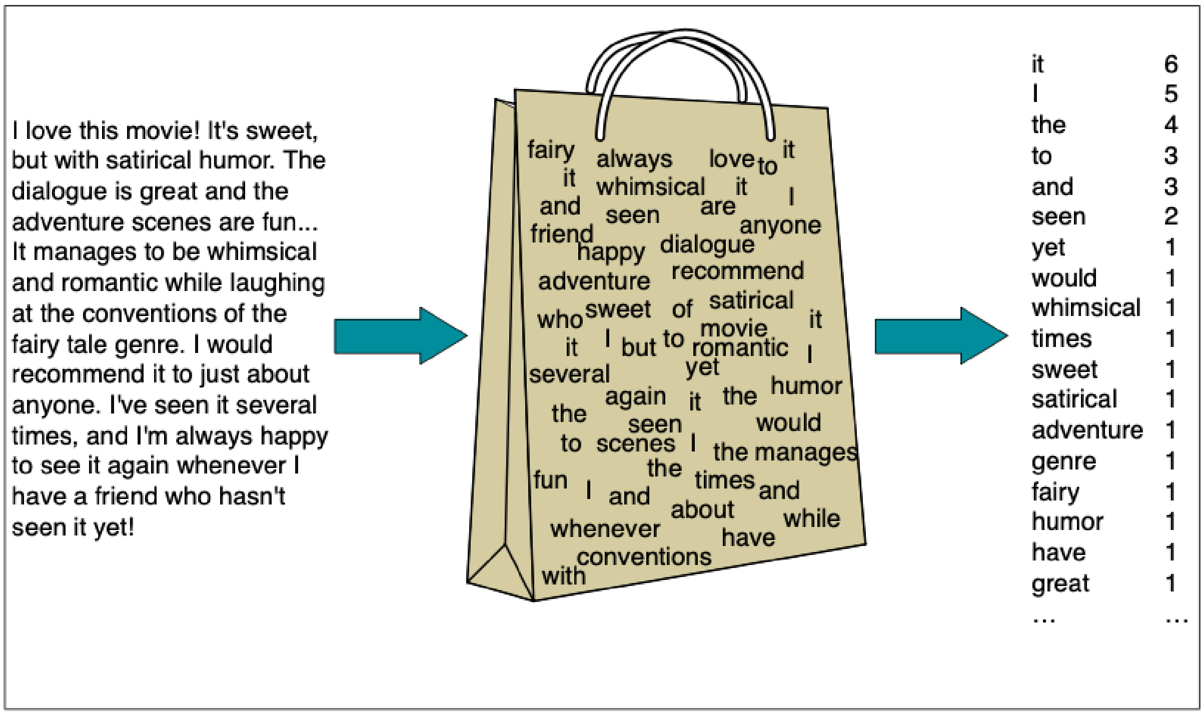

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!
fairy always love to it it whimsical it I and seen are anyone friend happy dialogue adventure recommend who sweet of satirical it movie it I but to romantic I several yet again it the humor the seen would to scenes I the manages fun the times and I and about while whenever have conventions with
it 6
I 5
the 4
to 3
and 3
seen 2
yet 1
would 1
whimsical 1
times 1
sweet 1
satirical 1
adventure 1
genre 1
fairy 1
humor 1
have 1
great 1
... ...

Figure 1: **Bag of words** representation for a movie review. Here, the position of the word is ignored (a major assumption) and, frequency of each word is used.

Naive Bayes is a probabilistic classifier, which means that for a document d, out of all classes $c \in C$ the classifier returns the class $\hat{c}$ which has the **maximum posterior probability** given the document. In Eq. 1, the hat notation $\hat{}$ means, “it is an estimate of the correct class”.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) \tag{1}$$

The intuition of Bayesian classification is to use Bayes’ rule to transform Eq. 1 into other probabilities that have some useful properties. Bayes’ rule is presented in Eq. 2; it gives us a way to break down any conditional probability $P(x|y)$ into three other probabilities:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2}$$

Substituting Eq. 2 into Eq. 1 we get:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) = \underset{c \in C}{\operatorname{argmax}}\, \frac{P(d|c)P(c)}{P(d)} \tag{3}$$

As we will be calculating $\frac{P(d|c)P(c)}{P(d)}$ for each possible class, we can conveniently drop $P(d)$ from Eq. 3. Therefore, we get a simpler formula:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) = \underset{c \in C}{\operatorname{argmax}}\, P(d|c)P(c) \tag{4}$$

Now we can see why Naive Bayes is a *generative model.* Because, in Eq. 4, there is an assumption of how a document is generated ($P(d|c)$: probability of observing the document given a class). We can imagine generating artificial documents, or at least their word counts by following this process!

In Eq. 4, $P(d|c)$ is the **likelihood** of a document d given class c and, $P(c)$ is the **prior probability** of class c. If document d has a set of features $f_1, f_2, \dots, f_n$, then we can write Eq. 4 as:

$$\hat{c} = \underset{c \in C}{\text{argmax}}\ P(f_1, f_2, \dots, f_n|c)P(c) \tag{5}$$

But, Eq. 5 is hard to compute directly. As, estimating the probability of every possible combination of features would require huge numbers of parameters and impossibly large training sets. To overcome this issue, Naive Bayes considers two simplifying assumptions:

1. Position of the word does not matter. Therefore, we assume that features $f_1, f_2, \dots, f_n$ encode only word identity and not position.
2. **Naive Bayes assumption**: The probabilities $P(f_i|c)$ are independent given the class c and hence can be *naively* multiplied as follows:

$$P(f_1, f_2, \dots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c) \tag{6}$$

Therefore, the final equation for the class chosen by a Naive Bayes classifier is:

$$c_{NB} = \underset{c \in C}{\text{argmax}}\ P(c) \prod_{f \in F} P(f|c) \tag{7}$$

In order to avoid underflow and increase speed, calculations for Eq. 7 are done in log space.

$$c_{NB} = \underset{c \in C}{\text{argmax}}\ \log P(c) + \sum_{f \in F} \log P(f|c) \tag{8}$$

As Eq. 8 computes the class as a linear function of input features, Naive Bayes classifier is also referred to as **linear classifiers**.

1.3 Training a Naive Bayes classifier

Now the question to answer is, how can we learn the probabilities $P(c)$ (prior) and $P(f_i|c)$ (likelihood)? For $P(c)$, we can ask what percentage of the documents in the training set are in each class c. Let N_c be the number of documents in the training data with class c and N_{doc} be the total number of documents. Therefore:

$$\hat{P(c)} = \frac{N_c}{N_{doc}} \tag{9}$$

To learn the probability $P(f_i|c)$, we will use the **Naive Bayes assumption** and assume a feature is just the existence of a word in the document's bag of words, and so we'll want $P(w_i|c)$, which we compute as the fraction of times the word w appears among all words in all documents of topic c. We first concatenate all documents with category c into one big "category c" text. Then we use the frequency of w_i in this concatenated document to give a maximum likelihood estimate of the probability:

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \tag{10}$$

The vocabulary V consists of the union of all the word types in all classes, not just the words in one class c.

However, there is an issue with this method. If we are trying to estimate the likelihood of word “Siddhant” given class positive, but there are no training documents that contain “Siddhant” and classified as positive. In such a case, the probability of this feature is zero:

$$\hat{P}(\text{“}Siddhant\text{”}|\text{positive}) = \frac{\text{count}(\text{“}Siddhant\text{”}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0 \tag{11}$$

And as Naive Bayes *naively* multiplies all the likelihoods together, the probability of class will become zero, no matter what the evidence is! To solve this we can use the **add-one (Laplace) smoothing**. Eq. 11 will be transformed to:

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V}(\text{count}(w, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V}(\text{count}(w, c)) + |V|} \tag{12}$$

It is crucial that the vocabulary V consists of the union of the word types in all classes, not just the words in one class c.

But, if there is a case when words occur in test data but are not in the vocabulary because they did not occur in the train data. For such **unknown words** we do not keep them in the test document and do not include any probability for them at all.

In some cases, we can ignore a class of words also known as **stop words**. These are words which are very frequent like *the* and *a*. Every instance of these stop words are removed from training and testing data.

Final algorithm for training Naive Bayes is:

vlined 1 Training Naive Bayes

```
for each class $c \in C$ do
    $N_{doc}$ = number of documents in $D$
    $N_c$ = number of documents from $D$ in class $c$
    logprior[$c$] ← log $\frac{N_c}{N_{doc}}$
    $V$ ← vocabulary of $D$
    bigdoc[$c$] ← append(d) for $d \in D$ with class $c$
    for each word $w$ in $V$ do
        count($w, c$) ← # of occurrences of $w$ in bigdoc[$c$]
        loglikelihood[w, c] ← log $\frac{\text{count(w, c)+1}}{\sum_{w' \in V}\text{(count(w', c)+1)}}$
    end for
end for
return logprior, loglikelihood, V
```

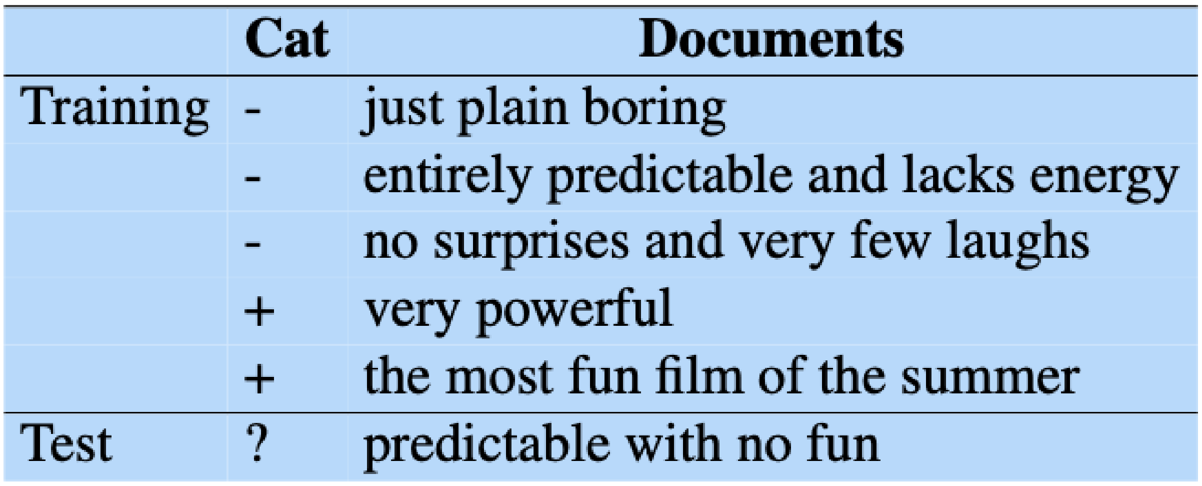
Algorithm for testing Naive Bayes is:

1.4 Worked example

Here we explore an example of training and testing Naive Bayes with add-one smoothing. Here we consider two classes, positive (+) and negative (−), and take the following training training and testing documents:

vlined 2 Testing Naive Bayes

```
for each class $c \in C$ do
  $sum[c] \leftarrow$ logprior[c]
  for each position $i$ in testdoc do
    $word \leftarrow testdoc[i]$
    if $word \in V$ then
      sum[c] $\leftarrow$ sum[c] + loglikelihood[$word, c$]
    end if
  end for
end for
return argmax$_c sum[c]$
```



	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

Figure 2: Dataset for training and testing Naive Bayes

The prior $P(c)$ for the two classes is computed using Eq. 9:

$$P(-) = \frac{3}{5} \tag{13}$$

$$P(+) = \frac{2}{5} \tag{14}$$

The word *with* does not occur in the training set so, we drop it. We calculate the likelihood of the remaining three words using Eq. 12 as:

$$P(\text{“}predictable\text{”}|-) = \frac{1+1}{14+20}\,; P(\text{“}predictable\text{”}|+) = \frac{0+1}{9+20} \tag{15}$$

$$P(\text{“}no\text{”}|-) = \frac{1+1}{14+20}\,; P(\text{“}no\text{”}|+) = \frac{0+1}{9+20} \tag{16}$$

$$P(\text{“}fun\text{”}|-) = \frac{0+1}{14+20}\,; P(\text{“}fun\text{”}|+) = \frac{1+1}{9+20} \tag{17}$$

Therefore, for the test sentence $S =$ ”predictable with no fun”, after removing the word ”with”, the chosen class is comptued as:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5} \tag{18}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5} \tag{19}$$

Thus, according to the model, S is a *negative* sentence.

Coded example: Please follow (https://colab.research.google.com/github/jakevdp/Python DataScienceHandbook/blob/master/notebooks/05.05-Naive-Bayes.ipynb) for exploring a coding exercise for Naive Bayes.

2 BoW

The Bag of Words (BoW) model is generally used to representation text into numbers. This method is used to extract feature from text document. Machine learning algorithms can be trained using these features. It provides a vocabulary of all the unique words found in the training set's documents. BOW is an approach widely used with: Natural language processing, Information retrieval from documents, Document classifications. A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things (1) A vocabulary of known words. (2) A measure of the presence of known words. It's called a "bag" of words because all information about the document's word order or structure is discarded. The model only cares about whether or not known words exist in the text, not where they appear.

2.1 BoW model

In this section, we will see how BoW model works with an example. BoW modelling contains three steps (i) Text collection (ii) Design vocabulary (iii) Create text vector. We will give brief description of this steps:

(i) Text collection: Lets take two lines from a text, (1) He is a good boy (2) He performs very well.

(ii) Design vocabulary: Now, we will make a list of unique words from these two sentences. These words are ('He','is','a','good','boy','performs','very','well'). This is a vocabulary of 8 words from a corpus containing 9 words.

(iii) Create text vector: In this step, we will score the words in each sentences. We will convert each sentence into a numerical vector. we have 8 words in the vocabulary. So, we can use a fixed-length document representation of 8, with one position in the vector to score each word. The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.

The scoring of the sentence would look like:

- 'He'=1
- 'is'=1
- 'a'=1
- 'good'=0
- 'boy'=1
- 'performs'=0
- 'very'=0
- 'well'=0

for this 8 words binary vector will look like [1,1,1,0,1,0,0,0]

Vector of the two sentence will look like:

(1) 'He is a good boy' = [1,1,1,0,1,0,0,0]

(2) 'He performs very well' = [1,0,0,0,1,1,0,1]

As a visual reference, see image 1 in this text.

2.2 Managing Vocabulary

As the vocabulary size increases, so does the vector representation of documents. In this example, length of the document vector is equal to the number of known words. Now, if we have a very large corpus then the length of the vector might be thousands of positions. Further, each document may contain very few of the known words in the vocabulary.

As a result, we will get vector with lots of zero scores, called a sparse vector or sparse representation. Sparse vectors require more memory and computational resources when modeling and the vast number of positions or dimensions can make the modeling process very challenging for traditional algorithms. So, we decrease the size of the vocabulary when using a bag-of-words model.

For this case, we will use simple text cleaning techniques that can be used as a first step, such as: Ignoring case, Ignoring punctuation, Ignoring frequent words, Fixing misspelled words, stemming algorithms to Reduce words to their stem (e.g. “Eat” from “Eating”), etc.

N-gram is at popular method that helps to create a vocabulary of grouped words. This method both changes the scope of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document.

In this approach, each word or token is called a “gram”. Creating a vocabulary of two-word pairs is, in turn, called a bigram model. Again, only the bigrams that appear in the corpus are modeled, not all possible bigrams.

3 Term Frequency Inverse Document Frequency - TFIDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that captures how relevant a word is in a document, with respect to the entire collection of documents.

What does this mean? Some words will appear a lot within a text document as well as across documents, for example, the English words the, a, and is. These words generally convey little information about the actual content of the document and don't make it stand out of the crowd.

TF-IDF provides a way to weigh the importance of a word, by contemplating how many times it appears in a document, with respect to how often it appears across documents.

Hence, commonly occurring words such as the, a, and is will have a low weight, and words more specific to a topic, such as leopard, will have a higher weight.

The intuition for this measure is :

1. If a word appears frequently in a document, then it should be important and we should give that word a high score.
2. But if a word appears in too many other documents, it's probably not a unique identifier, therefore we should assign a lower score to that word.

TF-IDF is the product of two statistics, term frequency and inverse document frequency.

$$tfidf(t,d,D) = tf(t,d)idf(t,D) \tag{20}$$

Where t denotes the terms; d denotes each document; D denotes the collection of documents.

3.1 Term Frequency - TF

Term frequency is, in its simplest form, the count of the word in an individual text. So, for term t, the term frequency is calculated as tf(t) = count(t) and is determined text by text. The first part of the formula tf(t,d) is simply to calculate the number of times each word appeared in each document. Of course, as with common text mining methods: stop words like “a”, “the”, punctuation marks will be removed beforehand and words will all be converted to lower cases.

$$tf(t,d) = (1 + \log tf_{t,d}) \tag{21}$$

3.2 Inverse Document Frequency - IDF

The inverse document frequency is a measure of how common the word is across all documents and is usually calculated on a logarithmic scale. The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(t,D) = \log \frac{N}{df_t} \tag{22}$$

Putting it all together, Mathematically TF-IDF can be represented as below

$$tf - idf_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t} \tag{23}$$

Here, N is the total number of documents and df(t) the number of documents in which the term t appears. The bigger the value of df(t), the lower the weighting for the term.

TF-IDF can be also used together with n-grams. Similarly, to weight an n-gram, we compound the n-gram frequency in a certain document by the times the n-gram appears across all documents.

TF-IDF shares the characteristics of BoW when creating the term matrix, that is, high feature space and sparsity. To reduce the number of features and sparsity, we can remove stop words, set the characters to lowercase, and retain words that appear in a minimum percentage of observations.

References

[1] Naive Bayes and Sentiment Classification. Speech and Language Processing. Daniel Jurafsky James H. Martin. (PDF)

[2] Naive Bayes Wikipedia (https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

[3] https://scikit-learn.org/stable/modules/naive_bayes.html

[4] https://www.cs.cmu.edu/˜tom/mlbook/NBayesLogReg.pdf

[5] https://www.cs.cmu.edu/˜tom/10701_sp11/slides/GNB_1-25-2011-ann.pdf

[6] A Gentle Introduction to the Bag-of-Words Model. Jason Brownlee. (https://machinelearningmastery.com/gentle-introduction-bag-words-model/)

[7] An Introduction to Bag of Words (BoW) — What is Bag of Words? Hussain Mujtaba. (https://www.mygreatlearning.com/blog/bag-of-words/)

[8] TF-IDF term weighting (https://scikit-learn.org/stable/modules/feature$_{e}xtraction.html$)