Notes on Natural Language Processing by Eisenstein

compiled by D.Gueorguiev 9/21/2024

# Linear Text Classification

**Definition**: *Text Classification Problem*

Given a text document, assign a label from a set of discrete labels.

## The Bag of Words

Question: how do we represent a document with text?

Use a column vector of word counts:

where is count for the -th word in a vocabulary . Here the size of the vocabulary is denoted with .

In linear classification the decision is based on a weighted sum of individual word counts where the word set is the feature set of the classification problem. The classification object is the vector ; this object is often called *a bag of words*. With a bag of words representation we are ignoring everything else but the frequency count of each word – we are not accounting for grammatical and syntactic constructs, sentence boundaries, paragraphs.

To predict a label from a bag of words we assign a score to each word in the vocabulary measuring fitness of the word with this label. These word scores are known as weights and are stored in column vector .

Let us consider multi-class classifier where The goal is to predict a label , given the bag of words , using weights . For each label , we compute a score , which is a scalar measure of the compatibility between the bag-of-words and the label . In a linear bag-of-words classifier, this score is the vector inner product between the weights and the output of a *feature function*

(1)

For example, given arguments and , element of this feature vector might be

(2)

returns the count of the word *whale* if the label is , and it returns zero otherwise. The index depends on the position of *whale* in the vocabulary and of in the set of possible labels. The corresponding weight then scores the fitness of the word *whale* for the label . Positive score means that this word makes the label more likely.

The output of the feature function can be formalized as a vector:

(3)

(4)

(5)

where is a column vector of zeros, and the semicolon indicates vertical concatenation.

For each of the possible labels, the feature function returns a vector that is mostly zeros, with a column vector of word counts inserted in a location that depends on the specific label .

This notation may seem awkward but generalizes to a range of learning settings, particularly *structure prediction*.

Given a vector of weights, , we can now compute the score by Eq (1). This inner product gives a scalar measure of the fitness of the observation with respect to the label .

Note that only features and weights are necessary. That is true because we can require that regardless of . With this requirement it is possible to implement any classification rule that can be achieved with features and weights. This is akin to the binary classification rule , where is a vector of weights, is an offset , and the label set is . For we have – that is, we need one dimensional simplex (a line) to separate a label set of size 2.

Question: how can we obtain the weights ?

One option is to manually set those. For example if we want to distinguish English from Spanish, we can use English and Spanish dictionaries, and set the weight to one for each word that appears in the associated dictionary. For example:

,

,

,

,

Usually it is not easy to set classification weights by hand, due to the large number of words and the difficulty of selecting exact numerical weights. We will introduce a probabilistic method to generate such weights.

## Naïve Bayes

Notation:

– the joint probability that the random variables and take specific values and respectively. The subscripts indicating the random variables will be omitted hereon. Thus we end up with the more concise:

– the joint probability of a bag of words and its true label

– a dataset of labeled instances which are i.i.d.

– the joint probability of the entire dataset given with

One approach to classification is to set the weights as to maximize the joint probability of a *training set* of labeled documents. This is known as *maximum likelihood estimation*:

(8)

(9)

(10)

Note: is a parameter of the probability function .

The probability function is defined through a generative model. **Algorithm 1** describes the generative model underlying the Naïve Bayes classifier with parameters .

**Algorithm 1**: *Generative process for the Naïve Bayes classification model*

for instance do:

draw the label

draw the word counts

i ) The first line of the model shown in Algorithm 1 encodes the assumption that all instances are mutually independent – neither the label nor the text of document affects the label or the text of document .

Furthermore, the instances are identically distributed – the distributions over the label and the text (conditioned on ) are the same for all instances . In other words, we make the assumption that every document has the same distribution over labels, and that each document’s distribution over words depends on the label, and not on anything else about the document. We also assume that the documents do not affect each other: if the word *whale* appears in document , that does not make it any more or less likely that it will appear again in document .

ii ) The second line of the generative model states that the random variable is drawn from a categorical distribution with parameter . The column vector gives the probabilities of each label, so the probability of drawing label is equal to . For example, let , with . Obviously, and .

iii ) The third line describes how the bag-of-words counts are generated. This line indicates that the word counts are conditioned on the label, so that the joint probability is factored using the chain rule

(11)

The distribution is the *multinomial* over vectors of non-negative counts. The PMF for the multinomial is:

(12)

(13)

The parameter is interpreted as a probability that any token in the document is the word .

The term is known as the multinomial coefficient, it does not depend on and can usually be ignored.

The notation indicates the conditional probability of word counts given label y, with parameter which is equal to . By specifying the multinomial distribution, we describe the *multinomial Naïve Bayes* classifier. This classifier treats each word token independently, conditioned on the class as the PMF factorizes across the counts.

### Types and Tokens

Let us consider a modification of the generative model of Naïve Bayes shown in **Algorithm 2**.

Instead of generating a vector of counts of *types* , x, this model generates a *sequence* of *tokens* . The distinction between types and tokens is: is the count of word type in the vocabulary e.g., the number of times the word *cannibal* appears; is the identity of token in the document, e.g. .

**Algorithm 2**: *Alternative generative process for the Naïve Bayes classification model*

for instance do:

draw the label

for token do:

draw the token

The probability of the sequence is a product of categorical probabilities. **Algorithm 2** makes a conditional independence assumption: each token is independent of all other tokens , conditioned on the label . This is identical to the naïve independence assumption implied by the multinomial distribution, and as a result, the optimal parameters for this model are identical to those in the multinomial Naïve Bayes. For any instance, the probability assigned by this model is proportional to the probability under multinomial Naïve Bayes. The constant of proportionality is the multinomial coefficient . Because , the probability for a vector of counts is at least as large as the probability for a list of words that induces the same counts: there can be many word sequences that correspond to a single vector of counts. For example, *man bites dog* and *dog bites man* correspond to an identical count vector, and is equal to the total number of possible word orderings for count vector .

Key takeaways from the Types and Tokens discussion:

Sometimes it is useful to think of instances as counts of types , ; other times, it is better to think of them as sequences of tokens, . If the tokens are generated from a model that assumes conditional independence, then these two views lead to probability models that are identical, except for a scaling factor that does not depend on the label or the parameters.

### Prediction

The Naïve Bayes prediction rule is to choose the label which maximizes :

(14)

(15)

Using (11) and (12) in (15) gives us:

(16)

(17)

(18)

where

(19)

(20)

The feature function is a vector of word counts and an offset, padded by zeros for the label not equal to . This construction ensures that the inner product only activates the features whose weights are in . These features and weights are all we need to compute the joint log-probability for each .

Key Takeaway:

Through this notation we have converted the problem of computing the log-likelihood for a document-label pair into the computation of vector inner product.

### Estimation

The parameters of the categorical and multinomial distributions are vectors of expected frequencies for each possible event. So, we argue that we can estimate these parameters empirically as

(21)

where refers to the count of word in the documents with label .

(21) defines the relative frequency estimate for . It is also the maximum likelihood estimate which maximizes the probability . Based on the generative model in **Algorithm 1**, the log likelihood is

(22)

Since we are interest in only as a function of we drop the second term of (22) which for our purposes is constant. Thus, (22) becomes

//TODO: finish the Estimation subsection and the section on Naïve Bayes

## Discriminative Learning

A problem with Naïve Bayes is that the features are independent – this is not really true in natural languages.

A good performance on text classification requires features that are not supported by the bag-of-words assumption.

* to better handle out-of-vocabulary terms, we want features that apply to multiple words, such as prefixes and suffixes and capitalization
* we want *n-gram* features that apply to multi-word units: bigrams , trigrams and beyond.

These features violate the Naïve Bayes independence assumption. Consider what happens if we add a prefix feature. Under the Naïve Bayes assumption, the joint probability of a word and its prefix are computed with the following approximation

(31)

We apply the chain rule to the LHS of (31)

(32)

Obviously, , since is guaranteed to be the prefix for .

Therefore,

(33)

Thus, the Naïve Bayes approximation will systematically underestimate the true probabilities of conjunctions of positively correlated features. To use such features, we need learning algorithms that do not rely on an independence assumption.

The origin of the Naïve Bayes independence assumption is the learning objective, , which requires modeling the probability of the observed text. In classification problems we are always given , and are only interested in predicting the label . In this setting , modeling the probability of the text is unnecessary. Discriminative learning avoids modeling of the test probability and focus directly on predicting .

### Perceptron

In Naïve Bayes, the weights can be interpreted as parameters of a probabilistic model. The problem is that Naïve Bayes makes an independence assumption to calculate the probabilities of text occurrence and we know it is incorrect. The Perceptron learning algorithm tries to learn the weights of the text corpora in an error-driven way.

The Perceptron algorithm in a nutshell

If we make a mistake increase the feature weights that are active with the correct label and decrease the weights for features that are active with the guessed label . The Perceptron algorithm is an online learning algorithm, since the classifier weights change after every example. For comparison, Naïve Bayes is batch learning algorithm which after it has been trained with the training dataset it has static weights.

**Algorithm 3**: *Perceptron Learning Algorithm*

procedure Perceptron()

repeat

select an instance

if then

else

until interrupt\_condition

return

**Definition**: *Linear separability*

The dataset is linearly separable iff there exists a weight vector and such that for every instance , the inner product

//TODO: finish the subsection on the Perceptron algorithm

# References

[1] [Natural Language Processing, Jacob Eisenstein, 2018](https://github.com/dimitarpg13/nlp_concepts/blob/main/literature/books/nlp-eisenstein-nov18.pdf)

[2] [Natural Language Processing, Lecture Notes, Michael Collins, Columbia U., CS-4705](https://github.com/dimitarpg13/nlp_concepts/blob/main/literature/books/natural-language-processing-lecture-notes-columbia-cs4705.pdf)

[3] [Machine Learning for Natural Language Processing, Michael Collins, Lecture Notes, Columbia U., E6998](https://github.com/dimitarpg13/nlp_concepts/blob/main/literature/books/machine-learning-for-natural-language-processing-lecture-notes-columbia-e6998.pdf)

[4] [Neural Network Methods in Natural Language Processing, Yoav Goldberg, Bar Illan University, 2017](https://github.com/dimitarpg13/nlp_concepts/blob/main/literature/books/Neural_Network_Methods_in_Natural_LanguageProcessing-2017-Yoav_Goldberg.pdf)