A well known method to reduce the noise of textual data is the removal of stopwords. This method is based on the idea that discarding non-discriminative words reduces the feature space of the classifiers and helps them to produce more accurate results (Silva and Ribeiro, 2003). This pre-processing method, widely used in the literature of document classification and retrieval, has been applied to Twitter in the context of sentiment analysis obtaining contradictory results. While some works support their removal (Bakliwal et al., 2012; Pak and Paroubek, 2010; Zhang et al., 2012; Speriosu et al., 2011; Gokulakrishnan et al., 2012; Kouloumpis et al., 2011; Asiaee T et al., 2012) others claim that stopwords indeed carry sentiment information and removing them harms the performance of Twitter sentiment classifiers (Saif et al., 2012b; Hu et al., 2013b; Martınez-Camara et al., 2013; Hu ´ et al., 2013a).

Regarding this issue, as discussed in \sectoin{background} different approaches have been used to collect those stop-words. So, a stop-words list has been built manually. Those words were collected based on the most common words used in formal and dialect Arabic language. During building this list, it was taken into consideration that one word can have different representation but they all have the same meaning (e.g ,ابن،بن .(

After collecting those words,

After preparing the dataset, another process was started which how to extract the features of those tweets. .The first approach and the baseline of this project was the unigram words. As mentioned in

[1.1](http://www.nltk.org/book/ch07.html#fig-ie-architecture) shows the architecture for a simple information extraction system. It begins by processing a document using several of the procedures discussed in [3](http://www.nltk.org/book/ch03.html#chap-words) and [5.](http://www.nltk.org/book/ch05.html#chap-tag): first, the raw text of the document is split into sentences using a sentence segmenter, and each sentence is further subdivided into words using a tokenizer. Next, each sentence is tagged with part-of-speech tags, which will prove very helpful in the next step, **named entity detection**. In this step, we search for mentions of potentially interesting entities in each sentence. Finally, we use **relation detection** to search for likely relations between different entities in the text.

Feature selection For unigram feature, there are usually 260,000 different features. This is a very large number. It makes model higher variance. (Since more complicated model has higher variance). So it will need much more training data to avoid overfitting. Our training set contains hundreds of thousands sentences. But it is still a large number of features for our training set. It is helpful if we discard some useless features. We try 3 different feature selection algorithms. Frequency-based feature selection This is the simplest way to do feature selection. We just pick features (unigram words in our case) for each class with high frequency occurrence in this class. In practice, if the number of occurrences of a feature is larger than some threshold (3 or 100 in our experiments), this feature is a good one for that class. As we seen in the result table, this simply algorithm increases about 0.03 of accurac

naïve Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of independence between every pair of features. Given a class variable http://scikit-learn.org/stable/_images/math/276f7e256cbddeb81eee42e1efc348f3cb4ab5f8.png and a dependent feature vector _1 through _n, Bayes’ theorem states the following relationship:

(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}
                

Using the naive independence assumption that

(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),

for all http://scikit-learn.org/stable/_images/math/df0deb143e5ac127f00bd248ee8001ecae572adc.png, this relationship is simplified to

(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}
           

Since (x_1, \dots, x_n) is constant given the input, we can use the following classification rule:

(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)

\Downarrow


and we can use Maximum A Posteriori (MAP) estimation to estimate (y) and (x_i \mid y); the former is then the relative frequency of class http://scikit-learn.org/stable/_images/math/276f7e256cbddeb81eee42e1efc348f3cb4ab5f8.png in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of (x_i \mid y).

Naïve Bayes classifiers have been known

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.