DEGREE PROJECT, IN COMPUTER SCIENCE , SECOND LEVEL
STOCKHOLM, SWEDEN 2015

Using Naive Bayes and N-Gram for Document Classification

KHALIF FARAH MOHAMED

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION (CSC)

Using Naive Bayes and N-Gram for Document Classification

Användning av Naive Bayes och N-Gram för dokumentklassificering

khaliffm@kth.se

Computer Science

Supervisor: Stefan Arnborg
Examiner: Stefan Arnborg

Abstract

The purpose of this degree project is to present, evaluate and improve probabilistic machine-learning methods for supervised text classification. We will explore Naive Bayes algorithm and character level n-gram, two probabilistic methods. The two methods will then be compared. Probabilistic algorithms like Naive Bayes and character level n-gram are some of the most effective methods in text classification, but to get accurate results they need a large training set. Because of too simple assumptions, Naive Bayes is a poor classifier. To rectify the problem, we will try to improve the algorithm, by using some transformed word and n-gram counts.

Sammanfattning

Syftet med det här examensarbetet är att presentera, utvärdera och förbättra probabilistiska maskin-lärande metoder för övervakad textklassificering. Vi ska bekanta oss med Naive Bayes och tecken-baserad n-gram, två probabilistiska metoder. Vi ska sedan jämföra metoderna. Probabilistiska algoritmerna är bland de mest effektiva metoder för övervakad textklassificering, men för att de ska ge noggranna resultat behövs det att de tränas med en stor mängd data. På grund av antaganden som görs i modellen, är Naive Bayes en dålig klassificerare. För att åtgärda problemet, ska vi försöka förbättra algoritmerna genom att modifiera ordfrekvenserna i dokumentet.

Acknowledgments

I would like to thank my supervisor Stefan Arnborg, who also happens to be my examiner, for his great help and direction. Without his inputs, this degree job would be very poor.

Table of Contents

1 Machine Learning

In machine learning we are interested in computer programs that learn from experience. Software algorithms are trained to learn a task. After training a new example/examples unseen during training are presented to the trained algorithm for test. The algorithm builds a model from training examples. The algorithm uses then this model to handle new unseen cases [1].
A learning problem has the following well defined form [1]:

Task T: is the task to be learned.
Performance measure P: variable/variables to optimize.
Training experience E: the source of experience.

For example, an algorithm that classifies documents into different classes may improve its performance as measured by the percent of documents that are correctly classified, through experience obtained by providing to it examples of documents that belong to the different classes.
A document classification problem:

Task T: classify text documents into classes or categories.
Performance measure P: percent of documents correctly classified.
Training experience E: a database of text documents sorted into classes.

There are different kinds of algorithms for learning: concept learning, decision tree, artificial neural network (ANN), genetics, probabilistic algorithms like Naïve Bayes and several others. These algorithms use different strategies/approaches to learn a task. The most common and widely used approach is the predictive or supervised learning. The goal of supervised learning is to learn a mapping from inputs x to outputs y, given a labeled set of input-output pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$. Here $\mathcal{D}$ is the training set and N is the number of training examples. In the predictive/supervised learning approach the problems to be learned have a well defined form or structure.
A second machine learning approach is descriptive or unsupervised learning. In this approach we have only inputs, $\mathcal{D} = \{x_i\}_{i=1}^{N}$, and the goal is to find "interesting patterns" in the data. This is a less well-defined problem, since we do not know what types of patterns to look for.
There is a third approach to machine learning, known as reinforcement learning, which is less common. This approach is useful when learning how to act when given occasional reward or punishment signals.
In this degree project we will use the first approach, that is to say, supervised learning.

2 Text Classification and Its Applications

Text Classification is the task to classify documents into predefined classes. Text Classification is also called:

- Text Categorization
- Document Classification
- Document Categorization

There are two approaches to Text Classification: manual classification and automatic classification. In this thesis we are interested in automatic classification using a training sample of already classified examples, in particular, probabilistic methods.

2.1 Relevant technologies

- Text Clustering
  - Create clusters of documents without any external information
- Information Retrieval (IR)
  - Retrieve a set of documents relevant to a query
  The best known example is Googling
- Information Filtering
  - Filter out irrelevant documents through user interactions
- Information Extraction (IE)
  - Extract fragments of information, e.g., personal names, dates, and places, in documents
- Text Classification
  - No query
  - decide topics of documents

2.2 Text Classification Applications

- E-mail spam filtering – classify an e-mail as spam or not spam
- Categorize newspaper articles and newswires into topics, such as sports, politics, etc
- Identify trends in the media flow
- Language identification, automatically determining the language of a text
- Genre classification, automatically determining the genre of a text
- Readability assessment, automatically determining the degree of readability of a text
- Sentiment analysis, “determining the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document” [Wikipedia]
- Authorship attribution is the task of determining the author of a text
  This has applications in historical sciences and forensics

3 Probabilistic Algorithms

In this degree project we are interested in probabilistic algorithms that are based on statistics. We will examine Naive Bayes algorithm and character level n-gram. Probability is suitable to apply to problems which involve uncertainty. In machine learning problems, uncertainty can appear in many different forms: what is the best prediction given some data? What is the best model given data? Bayesian learning algorithms explicitly manipulate probabilities. Each training example can either increase or decrease the estimated probability that a hypothesis is true. This provides a more flexible approach to learning compared to algorithms that discard a hypothesis altogether if it contradicts a single training example. In Bayesian learning a hypothesis is more or less probable [1].

3.1 Bayesian learning models

The discussion in this section follows Hertzmann[6], which in turn is inspired by Ed Jaynes' popularization of Bayesian methods [12].
Bayes' theorem is widely used today as the basis of statistical or probabilistic inference. Bayesian reasoning provides a natural approach to many difficult data-modeling problems. Bayesian reasoning is a model for logic in the presence of uncertainty**.** Bayesian methods match human intuition very closely. The mathematical foundations of Bayesian reasoning are more than 100 years old, and have become widely-used in many areas of science and engineering, such as astronomy, geology, bioinformatics, evolutionary linguistics, and electrical engineering [6]. Thomas Bayes and Pierre de Laplace used the method on simple examples 270 years ago [12].

Bayesian reasoning provides three main benefits:
1. Principled modeling of uncertainty – uncertainty is described as probability.
2. General purpose models for unstructured data – any data can be described by its probability distribution, which in turn can be uncertain (hierarchical models).
3. Effective algorithms for data fitting and analysis under uncertainty – the analytical methods used before computers are unsuitable for Bayesian analysis, because many problems do not have analytical solutions.

In classical logic, we have a number of statements that may be true or false, and we have a set of rules which allow us to determine the truth or falsity of new statements.

Classical logic provides a model of how humans might reason. But, classical logic assumes that all knowledge is absolute. Logic requires that we know some facts about the world with absolute certainty, and, then, we may deduce only those facts which must follow with absolute certainty. This is how pure mathematicians work.

In the real world, there are almost no facts that we know with absolute certainty — most of what we know about the world we acquire indirectly, by observing and measuring the external world, or from dialogue with other people. In other words, most of what we know about the world is **uncertain**.

What we need is a way of discussing not just true or false statements, but statements that have different levels of certainty, statements in which we have varying degrees of **belief.** In addition to defining such statements, we would like to be able to use our beliefs to reason about the world and interpret it. As we gain new information, our beliefs should change to reflect our greater knowledge. This is how people who are not pure mathematicians work.

Thomas Bayes has invented what we now call Bayes' Theorem, or Bayes' Rule. According to Bayes'

Rule, we have a model of the world described by some unknown variables h, and we observe some data D; our goal is to determine h from the D. We describe the probability model as *P*(D|h)—if we know the value of h, then this h will tell us what data we expect. Furthermore, we must have some prior beliefs as to what h is (P(h)), even if these beliefs are completely non-committal (e.g., a uniform distribution). Bayesian analysis gives a recipe to answer this question: Given the data, what do we know about h?

Applying the product rule gives:
P(D, h) = P(D|h)P(h) = P(h|D)P(D)
Solving for the desired distribution, gives **Bayes' Rule:**

$$\boldsymbol{P(h|D) = \frac{P(D|h)P(h)}{P(D)}}$$

The different terms in Bayes' Rule are used so often that they all have names:

$$\underbrace{\boldsymbol{P(h|D)}}_{\boldsymbol{posterior}} = \frac{\overbrace{\boldsymbol{P(D|h)}}^{\boldsymbol{likelihood}}\ \overbrace{\boldsymbol{P(h)}}^{\boldsymbol{prior}}}{\underbrace{\boldsymbol{P(D)}}_{\boldsymbol{evidence}}}$$

In a document classification application, D represents the possible documents, and h represents which class they belong to – in our case, each document belongs to exactly to one class. The probabilities P(D) and P(D|h) can be found if we know enough about the language in which the documents are written, but unfortunately we don't. The second way to find probabilities is by means of the central limit theorems, which give conditions under which probabilities can be approximated by observed frequencies in a training corpus where we assume that the training corpus has the same probability distribution as the instances of the test set. This also does not work immediately, because it is obvious that most possible documents will never be seen: a 100 character document in a 32 character alphabet has 500 bits of information, and most such documents will never be seen by anyone. Even constraining analysis to meaningful documents, where it is estimated that a character has 1 bit information content[Wikipedia], the number of possible documents will be 2^100, and it is not possible to get a probability table over the quantities P(D) and P(D|h) in last section. For this reason, in most complex applications, models are built which approximate P(D) and P(D|h) by means of a feature set. Common simplifications made in model building are independence assumptions (e.g., that the features are created independently, given h). The methods tested in this thesis build on extracting a feature set from the document and assuming that each feature occurs in a document with a probability that is dependent on the class h of the document but not on the other features of it. By selecting the features well it is often possible to get reliable statistical inferences using Bayes theorem. The independence assumption between features characterizes the Naive Bayes method. For a problem with N features and H hypotheses we only need the N*H probabilities P(f|h), whereas without the independency assumption we would need 2^N*H. But we also introduce inaccuracy, because in reality the features are not independent.
.

3.2 Bayesian Learning

Bayes' Theorem is the basis of Bayesian learning algorithms. It provides a method for calculating the probability of a hypothesis, based on its prior probability, the probability of observing data given the hypothesis, and the data itself.
Some short hand notation first before we define the theorem. P(h) denotes the prior probability that

the hypothesis h holds, before we have observed the data. P(h) is based on any prior knowledge we have about the hypothesis. If we do not have such background knowledge we may assign the same prior probability to each candidate hypothesis (a non-informatic prior). P(D) denotes the prior probability that the data D holds given no prior knowledge about which hypothesis holds. P(D|h) is the probability that the data D holds in a world where the hypothesis h holds. P(h|D) is the probability that the hypothesis h holds given that the data D is observed. P(h) is independent of the observed data, whereas P(h|D) depends on it. In machine learning we are often interested to know the posterior probability P(h|D). P(h|D) is called the posterior probability of h, because it reflects on our confidence that h holds after the data is observed [1].

Bayes' theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{3.1}$$

In many learning scenarios we need to calculate the most probable hypothesis $h \in H$ in a set of hypothesis H. There can be several such most probable hypotheses in a set. Any such maximally probable hypothesis is called maximum a posteriori (MAP) hypothesis. We can calculate a MAP hypothesis by using Bayes' theorem. We can say that h_{MAP} is a MAP hypothesis if

$$\begin{aligned} h_{MAP} &\equiv \operatorname*{argmax}_{h \in H} P(h|D) \\ &= \operatorname*{argmax}_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \operatorname*{argmax}_{h \in H} P(D|h)P(h) \end{aligned} \tag{3.2}$$

A common assumption is that we do not know what the individual prior probabilities P(h) have for values. In such cases they get equal values (i.e., they are uniformly distributed). In that case we only need to consider the term P(D|h). Any hypothesis that maximizes such term is called Maximum likelihood (h_{ML}) hypothesis. When such cases occur, we can further simplify equation 3.2:

$$h_{ML} \equiv \operatorname*{argmax}_{h \in H} P(D|h) \tag{3.3}$$

It has been suggested that the posterior probability of h_{MAP} is used as a confidence measure for the prediction. This is a good idea if the probability model of the analysis is correct, but unfortunately the approximate nature of the probability model usually gives far too optimistic confidence estimate. So h_{MAP} is our best guess on the true value of h. But often we know that our guess is probably wrong. If we have posterior probabilities (0.26. 0.25, 0.25, 0.24) for a set of 4 hypothesis, then our best guess is the first hypothesis, but it is wrong 74% of the times. For posterior probabilities (0.99, 0.004, 0.003, 0.002) on the other hand, the first hypothesis is also our best guess, but we are only wrong 1% of the time. In this project it was found that the statistical models used give reasonably good accuracy despite the approximate nature of the models used, but the posteriors did not give good confidence estimates. A new method, conformal prediction [13] developed by Vovk and others, promises to give valid confidence estimates even if inaccurate statistical models are used. Unfortunately there was not time to test this method.

3.2.1 Naïve Bayes Classifier

Naïve Bayes is a Bayes classifier which uses Bayes' theorem with two simplifying assumptions: the independence between the attributes and position independence of attributes. Let us elaborate what does that mean. The instance to classify is often composed of many attributes (or features). As

we have seen before, to make use of Bayes theorem, we have to estimate the conditional probability P(x|h). Now the instance x consists of (or is described by) many attributes. Let us say it consists of a sequence of attributes ($a_1 \ldots a_n$). Then to calculate the probability $P(a_1 \ldots a_n|h)$ we need to take into account the interdependence between the attributes, in other words the joint probabilities of the attributes is to be included in the calculation. Then we have:

$$P(a_1 \ldots a_n|h) = P(a_1|h)\, P(a_2|h, a_1) P(a_3|h, a_1, a_2) \ldots P(a_n|h, a_1 \ldots a_{n-1}). \quad (3.4)$$

This is not practical, we can't keep track of all past history [4], for then we have to calculate a lot of joint probabilities. Now Naïve Bayes is based on the simple assumption that attributes are independent of each other:

$$P(a_1 \ldots a_n|h) = P(a_1|h)\, P(a_2|h) P(a_3|h) \ldots P(a_n|h). \quad (3.5)$$

Inserting (3.5) into (3.2) and using the product symbol we get:

$$h_{MAP} \equiv \underset{h \in H}{\operatorname{argmax}}\, P(h) \prod_i P(a_i|h)^{f_i} \quad (3.6)$$

Inserting (3.5) into (3.3) and using the product symbol we get:

$$h_{ML} \equiv argmax_{h \in H} \prod_i P(a_i|h)^{f_i} \quad (3.7)$$

f_i is the number of times the attribute a_i is found in the document. When NB is applied to document classification, the attributes are taken to be the words of the document.
The second assumption that Naïve Bayes is based on states that attributes are position independent, i.e. if w is the word encountered at a_5 and at a_7, then $P(a_5 = w|h) = P(a_7 = w|h)$. Since position does not matter (of course it matters, but we are assuming to simplify), one single probability is calculated for every word and hypothesis in the document to classify. This reduces the number of probabilities to calculate.

The probability of a word w given hypothesis h is estimated from the training set: if the documents classified with h contain N words in total and f occurrences of word w, then P(w|h) can be estimated by the relative frequency f/N, which maximizes expected likelihood, or with (f+1)/(N+V), which minimizes expected (over posterior) squared error of the estimate (here V is the total number of words considered). The latter estimator is Laplace's estimator.

3.2.2 Improvements to Naive Bayes Classifier

The naive Bayes classification method became popular in Artificial Intelligence during the 1980s when it gradually became obvious that the previously used methods based on logic held limited promise. The independency assumption is of course not satisfied in most applications, but the alternative makes it impossible to get good probability tables with reasonably large training data sets. It has often been observed that the method works better than one would expect. David Hand [2] has analyzed the naive Bayes method in detail, and found some explanations why the method works despite its weaknesses. Today it is however acknowledged that methods exist which are both more accurate and still useable with small training corpora. These methods make other independence assumptions than the naive Bayes method. They can be said to make less drastic independency assumptions. They are often described as special cases of graphical statistical models

[9].

A standard method to eliminate the NB assumption of attribute independence is to use the probability model of a Bayesian Network (BN). In this model, the attributes are made vertices of a directed acyclic graph and each attribute has a probability distribution that depends on values of the ancestor attributes in the graph. It is possible to learn the graph structure of a BN from training set, and typically a large training set will give a denser graph than a small one. A BN is optimized to predicting all attributes whereas in classification we are only interested in predicting the class attribute. For this reason Friedman et.al [9] invented the TAN classifier which has a tree structured graph targeting the class attribute. This classifier was found better than both BN and NB, but this result has been questioned by Madden et.al [8]. In document classification both BN and TAN classifiers are problematic because of the very large numbers of attributes. Rennie et.al [7] has studied some shortcomings of the NB method. One of these concerns the use of word counts. The assumption that words are generated independently in documents is far from true, since once a word has appeared in a document the probability that it will be repeated in the same document is much higher than the probability of its first occurrence. They propose replacing word counts fi in a document by the logarithmic transformation log(fi +1). In our experiment we will also test the binary transformation where all positive word counts are replaced by 1. Rennie remarks that the logarithmic transformation does not correspond to any statistical model. The binary transformation has the advantage (in theory) of corresponding to a statistical model.

3.3 Character level n-gram

Instead of using unigrams (single words) as NB does, we could use n-grams. We could use word n-grams or character n-grams. N-grams are consecutive sequences of tokens, where the tokens are either words or characters. When the size of n is 1, we have unigrams (the traditional bag of words) which NB uses, but we will use n-grams with sizes greater than 1. Why use n-grams? As we have seen NB's MAP and ML formulas were simplified to make the probability calculations tractable. But on the way we lost information in the form of context. N-gram reintroduces some of that lost information in the form of a short past history. We limit the history to fixed number of words or characters N:

$$P(w_k|w_1,\dots,w_{k-1}) \approx P(w_k|w_{k-N+1},\dots,w_{k-1}) \quad (3.8)$$

Bigram N=2

$$P(w_k|w_1,\dots,w_{k-1}) \approx P(w_k|w_{k-1}) \quad (3.9)$$

$$P(w_1,w_2,\dots,w_k) \approx P(w_1)P(w_2|w_1)\dots P(w_k|w_{k-1}) \quad (3.10)$$

Trigram N=3

$$P(w_k|w_1,\dots,w_{k-1}) \approx P(w_k|w_{k-2},w_{k-1}) \quad (3.11)$$

$$P(w_1,w_2,\dots,w_k) \approx P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\dots P(w_k|w_{k-2},w_{k-1}) \quad (3.12)$$

Equations 3.10 and 3.12 give probabilities of documents. In order to classify documents, we compute 3.9 – 3.12 conditional on h.
To compute ML estimates we need to calculate the individual n-gram probabilities. Both in NB and n-gram, a large corpus is needed. The corpus is then divided into two parts, training set and test set.

Bigram:

$$P(w_1,w_2) = \frac{freq(w_1,w_2)}{\sum_w freq(w_1,w)} \quad (3.13)$$

Trigram:

$$P(w_1, w_2, w_3) = \frac{freq(w_1,w_2,w_3)}{\sum_w freq(w_1,w_2,w)} \quad (3.14)$$

Character level n-grams are better than word level n-grams in many aspects. Word level n-grams increase dimensionality of the problem compared to character level n-grams [5]. There are ca. 50 000 words in the English dictionary compared to 26 characters in the alphabet. If we have bigrams, it would generate $50\ 000^2 = 2.5x10^9$ pairs of words, on the other hand 26^2 would generate only 676 pairs of characters. Another advantage of character level n-grams over word level n-grams is that character level n-grams are robust to spelling errors in comparison to word level n-grams. The majority of the character level n-grams generated by a slightly misspelled word will be the same as the original (or correct) word. Another important aspect of character level n-grams is that they minimize the problem of sparse data. The sparse data problem arises when many n-grams have zero frequency. Because there are less character combinations compared to word combinations, the probability that there are character level n-grams with zero frequency is very low. One way to reduce the dimensionality is to remove all terms occurring a few times. Another way is to remove very common words such as “the”, ”often”, ”always” and so on. In this project work the later list, that is to say the common words are put together in a list called stop list. This list is then used to remove all common words occurring in the text that is to be classified.
What is an ideal value of n in an n-gram? In other words, which n gives good results? The results we get from the tests show that an n-gram of size 5 or 6 gives the best results.

3.4 Smoothing

The problem of sparse data we have mentioned in the previous section arises because if probabilities are estimated by frequencies, all probability mass is assigned to events in the training text [2]. Even if the training corpus is very large, it does not contain all possible unigrams, bigrams, trigrams, etc. The frequency based probability estimation will assign zero probability to an unseen unigram/n-gram in the test data. What is needed is to reserve some probability mass to unseen events. Smoothing: distribute the probability mass to all possible events so that there is no zero probability. Zero probability is a problem because equations 3.6 and 3.7 are products and will result in zero if one factor is zero.

There are many smoothing techniques and we will mention some of them. The most simple and oldest of them all is the one we will use and is called add-one or Laplace.
Laplace:

- Add one to all frequency counts (even unseen events).
- V is the vocabulary of all unique events (unigrams in the case of traditional bag of words, bigrams, trigrams, etc in the case of n-grams).
- N is the number of all events.
- Re-estimate the probabilities.

Unigrams:

$$P(w) = \frac{freq(w)+1}{N+V} \quad (3.15)$$

Bigrams:

$$P(w_2|w_1) = \frac{freq(w_1,w_2)+1}{\sum_w freq(w_1,w)+V} \quad (3.16)$$

Laplace is enough when dealing with categorization of documents. But there are more sophisticated techniques in use such as Good-Turing, Katz Backoff, Kneser-Ney Smoothing among others.

4 Evaluation

Which method gives the best results? How do we compare the results? So far we have worked with Naive Bayes using bag of words (unigrams of words) and character level n-grams. What we want to know is which method (NB with bag of words or character n-grams) is the best tool for document classification? We will do the comparison in the following way:
Suppose we have four classes: A,B,C and D. Count the number of correctly classified documents in class A and divide the total number of documents in the class and we get the percent of correctly classified documents in class A. In the same way we calculate the number of documents in A incorrectly classified as class B, C and D. And we do the same for each class. This gives a confusion matrix p_{ij} with probability that a document is of class i and is found to have class j. The accuracy is the sum of diagonal elements in the confusion matrix.

An experiment was conducted to check the relative performance of NB on words and n-gram on characters. Three transformations of word counts were tested: no transformation, logarithmic transformation, and binary transformation. The code for the classifiers was written in Java 8, and run on PC with AMD Turion 64 X2 CPU 1.61 GHZ. The test corpus was Mitchell's 20newsgroups, the version published in [11], and the stop list proposed there was used. Most documents were posted in 1993. Only four of the groups (talk.politics.misc, talk.politics.guns, talk.politics.mideast, talk.religion.misc), were used, in order to get confusion matrices of reasonable size. The newsgroups have approximately 1000 documents each, and the first 700 were used for training set, the remainder for test sets. We needed an external library "org.apache.lucene.analysis" from apache to generate n-grams. For NB applications it took about a minute to execute, whereas n-gram applications needed about 4 minutes.

There is another way of doing the evaluation using perplexity. Perplexity is defined as the probability of the entire test set normalized by the number of tokens [3]. Depending on whether we are using character level or word level n-grams, the number of tokens is then the number of characters respective the number of words in the test set.

$$PPT(T) = P(w_1, \dots, w_N)^{-1/N} \tag{4.1}$$

Using the chain rule and 3-gram we can write this as

$$PPT(T) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-2}, w_{i-1})}} \tag{4.2}$$

To compute perplexity we need to concatenate all test documents into one single document.

4.1 Training and Test Data

We are using the 20 newsgroups as training and test data. The 20 newsgroups is a collection of approximately 20 000 network newsgroups documents, partitioned into 20 different topics or categories. Every category contains nearly 1000 documents. Here is a list of the 20 categories, partitioned into subject matter:

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

We have used the version of this data set found in [11], with the recommended stop list. Note that there are several versions of this data set on the web. In some of these, some correction has been made, such as elimination of cross-posted entries, removal of names, etc. It should be noted that the class of the documents is quite weakly defined: it is just the newsgroup into which the author happens to post his document, and in long discussion threads authors have a tendency to deviate from the topic of the newsgroup.

4.2 Test Results

We split the training corpus into two parts, a training part and a test part. The training part is made of 70% of the documents, and the remaining 30% is the test part. We could randomly choose 30% of the documents in each category to be our test documents, but we make it easy for us and simply choose the first 700 documents as training documents and the last 300 as test documents. We trained an n-gram algorithm with maxGram=6. Also Naive Bayes algorithm with bag of words (unigram) was trained. Because features are assumed to be independent from each other, a class with more training data will be favored. A weight balancing can be done to tackle the problem. But here we will focus another problem for both Naïve Bayes and Ngram. Normally most text documents contain few words with high frequency and many words which appear once or few times. The phenomenon is known as heavy-tail. Distribution of word frequency is heavy-tailed. A solution to the problem is transforming word counts. One way to do that is taking the log2 of the word count:

$$f_i = log_2(1 + f_i)$$

f_i is the frequency of the i^{th} word in the document. This will reduce the dominance or the heavy contribution to the weights by few words with high frequency. It will map 0 → 0, 1 → 1, and down weights the heavy values.

We will run tree different tests for each algorithm: a test with normal word counts, a test with $f_i = log_2(1 + f_i)$, and lastly a test with $f_i = 0$ if word count = 0 and $f_i = 1$ if word count ≥ 1. From the test results, we get the confusion-matrix, and from it we will calculate the accuracy. Accuracy is the overall correctness of the model and is calculated as the sum of correct classifications divided by the total number of classifications.

4.2.1 NB results

1) fi=total number of word i in document.

	Pol.mis	rel.mis	pol.gun	pol.mid
talk.politics.misc	196	55	30	19
talk.religion.misc	40	233	18	9
talk.politics.guns	47	15	229	9
talk.politics.mideast	63	8	1	228

Accuracy = (196 + 233 + 229 + 228) / 1200 ≈ 73.83%

2) fi = log2(1+fi).

	Pol.mis	rel.mis	pol.gun	pol.mid
talk.politics.misc	212	47	26	15
talk.religion.misc	38	244	12	6
talk.politics.guns	40	11	243	6
talk.politics.mideast	32	3	1	264

Accuracy = (212 + 244 + 243 + 264) / 1200 = 963/1200 = 80.25%

3) fi=0 if count=0, fi=1 if count>=1.

	Pol.mis	rel.mis	pol.gun	pol.mid
talk.politics.misc	215	46	25	14
talk.religion.misc	38	243	12	7
talk.politics.guns	37	10	247	6
talk.politics.mideast	19	3	1	277

Accuracy = (215 + 243 + 247 + 277) / 1200 = 982/1200 ≈ 81.83%

4.2.2 N-Gram Results

1)6-gramCount, 5-gramCount: the number of 6-, respective 5-gram generated from training corpus.

	pol.mid	pol.mis	rel.mis	polit.gun
talk.politics.mideast	299	1	0	0
talk.politics.misc	18	231	32	19
talk.religion.misc	9	47	238	6
talk.politics.guns	11	31	6	252

Accuracy = (299 + 231 + 238 + 252) / 1200 = 1020/1200 = 85%

2)6-gramCount = log2(1+6-gramCount), 5-gramCount = log2(1+5-gramCount)

	Pol.mid	pol.mis	rel.mis	pol.gun
talk.politics.mideast	298	2	Null	Null
talk.politics.misc	17	218	43	22
talk.religion.misc	4	32	258	6
talk.politics.guns	5	25	7	263

Accuracy = (298 + 218 + 258 + 263) / 1200 = 1037/1200 ≈ 86.42%

3)6-gramCount = 1, 5-gramCount=1.

	Pol.mid	pol.mis	rel.mis	pol.gun
talk.politics.mideast	282	16	1	1
talk.politics.misc	17	204	49	30
talk.religion.misc	5	21	265	9
talk.politics.guns	1	19	12	268

Accuracy = (282 + 204 + 265 + 268) / 1200 = 1019/1200 ≈ 84.92%

4.2.3 Important Words

For each word in the vocabulary, there is at least one class, for which it has a highest posterior probability. To see how important a word is for a class in its classification of a document, we take the difference between the maximum and minimum posterior probabilities. The quantity we get is a measure of the importance of the word to the class. These words are not necessarily the most common words in the class; rather they are words that are very rare or absent in the other classes.

Here is a list of the 30 most important words of each class of the four classes in our evaluation:

talk.politics.misc
politics misc alt government president optilink mr clinton stephanopoulos cramer health legal br isc men sex american insurance article desy hallam dscomsa rochester states private apr make libertarian don gay

talk.religion.misc
religion talk god cmu abortion atheism jesus srv sandvik cs christian de cantaloupe morality frank objective apple sni net ap xref writes horus kent bible good origins netcom uk science

talk.politics.guns
guns gun stratus fbi firearms usa control weapons file law state batf fire cdt udel atf bill militia sw crime waco amendment usenet transfer stanford police arms colorado constitution ucsu

talk.politics.mideast
soc turkish culture israel mideast armenian israeli armenians jews soviet jewish org greek armenia turks arab turkey zuma people history argic serdar genocide bony soldiers mcgill rights human war sera

5 Discussion

As can be seen from the test results, the enhanced algorithm with transformed word counts result in better accuracy. Are the results as expected for the NB algorithm? Definitely the results of the improved algorithm are close to what others have achieved. But there is possibility that results could still be enhanced if we implemented all steps in the article, Rennie et.al [7]. Weight balancing is one of the steps discussed there.

The list of important words consists of words which are very rare or totally absent in the other classes. These words tip the balance in favor of their class, in other words, they strengthen the weight which decides which class a document belongs to. We can see three classes of words: those that generally connect to the topic, words of current interest (like Clinton), and names of authors that tend to post in only one of the newsgroups. It seems clear that the set of important words change with time, and today we would see words as Obama, Iraq, Syria and IS. So constant tuning (retraining) is necessary in a long-lived application.

The posterior probability distribution is too peaked to show true confidence in the classification. It gives a confidence greater than 99%, while the true value lies in the interval 73-85%. Because of this, it is a bad approximation to the expected value of the real probability distribution. As a result the posterior probability distribution does not work well as a confidence measure for Bayesian classification.

The results of ngram are better than results of all versions of NB.

The transformed counts give clearly better results on NB, but not on n-gram. The binary transformation is a little better than the logarithmic one (on the test data), but for the n-gram the transformation does not give any obvious improvements.

6 Literature

[1] Tom M. Mitchell, Machine Learning, McGraw-Hill 1997.

[2] Julia Hockenmaier, Natural Language Processing, Course Notes, University of Illinios at Urbana-Champaign, 2014.

[3] Jurafsky and Manning, Natural Language Processing, Course Notes, Stanford University, 2014.

[4] Jimmy Lin, N-Gram Language Models, Course Notes, University of Maryland, 2009.

[5] Ioannis Kanaris, Konstantinos Kanaris, Ioannis Houvardas, and Efstathios Stamatatos, Words vs.Character N-grams for Anti-spam Filtering, International Journal on Artificial Intelligence Tools, 2006.

[6] Aaron Hertzmann, Introduction to Bayesian Learning, Course Notes 2009.

[7] Jason Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger, Tackling the Poor Assumptions of Naive Bayes Text Classifiers, ICML 2003.

[8] MG Madden, On the classification performance of TAN and general Bayesian networks, SGAI International Conference 2008.

[9] Nir Friedman, Dan Geiger, Moises Goldszmidt, Bayesian Network Classifiers, Machine Learning 1997.

[10] Paul Lewis, Naïve (Bayes) at forty, The Independence assumption in information retrieval, European Conference on Machine Learning, 1998.

[11] McCallum, Andrew Kachites. "Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering." http://www.cs.cmu.edu/~mccallum/bow/. 1996.

[12] Ed Jaynes: Probability Theory: The Logic of Science, Cambridge University Press 2003

[13] Glenn Shafer, A tutorial on conformal prediction, Journal on Machine Learning, 2008.