Text Emotion Recognition

Varun Srinivasan
Dept of Computer Science & Engineering
Santa Clara University
Santa Clara, USA
v1srinivasan@scu.edu

***Abstract*—Present AI technology addresses classification problem for text, images and media. Among those, text classification stands distinctively as a real-life problem as users demand their devices to be smart to identify, translate and suggest lexical content. Intent, emotion and sentiment analysis are primarily addressed by all companies' research teams. This paper covers a few such classifiers and compare their accuracy.**

***Keywords*—Natural Language Processing, Deep Learning, Text Classifiers, Bag of Words, Vector Space Representation**

I. INTRODUCTION

The current technology is headed towards Artificial Intelligence incorporating Machine/Deep Learning techniques to any application. One such branch that is gaining attention is Natural Language Processing. Top tech giants like Google researches on NLP with semantic (label phrases with types such as person, location, or organization) and syntactic (predict part-of-speech tags & morphological features like gender or number from a word) algorithms.

A. *What is NLP?*

The structure and meaning of text can be understood by offering powerful machine learning models. It is used to extract information about people, places, events and much more, mentioned in text documents, news articles or blog posts. It can also be used to understand sentiment about content on social media or parse intent from conversations in a messaging app.

Natural Language Processing (NLP) research currently focuses on algorithms that apply at scale, across languages, and across domains. NLP nowadays impacts user experience in search, mobile, apps, ads, translate and more. It spans the range of traditional NLP tasks, with general-purpose syntax and semantic algorithms underpinning more specialized systems. It is particularly interesting that the algorithms scale well and can be run efficiently in a highly distributed environment.

B. Syntactic systems

The *syntactic systems* [1] predict part-of-speech tags for each word in a given sentence, as well as morphological features such as gender and number. They also label relationships between words, such as subject, object, modification, and others. We focus on efficient algorithms that leverage large amounts of unlabeled data, and recently have incorporated neural net technology.

C. Semantic systems

Semantic systems [1] focus on identifying entities in free text, label them with types (such as person, location, or organization), cluster mentions of those entities within and across documents (co-reference resolution) and resolve the entities to the Knowledge Graph is abundant. Recent work has focused on incorporating multiple sources of knowledge and information to aid with analysis of text, as well as applying frame semantics at the noun phrase, sentence, and document level.

II. TEXT CLASSIFIERS

Text classification is an important task in Natural Language Processing with many applications, such as web search, information retrieval, ranking and document classification [2]. Recently, models based on neural networks have become increasingly popular. While these models achieve very good performance in practice, they tend to be relatively slow both at train and test time, limiting their use on very large datasets.

Meanwhile, linear classifiers are often considered as strong baselines for text classification problems[3]. Despite their simplicity, they often obtain state-of-the-art performances if the right features are used. They also have the potential to scale to very large corpus.

A. *MaxEnt Classifier*

The Maximum Entropy classifier is a probabilistic classifier which belongs to the class of exponential models. Unlike the Naive Bayes classifier that we discussed in the previous article, the Max Entropy does not assume that the features are conditionally independent of each other. The MaxEnt is based on the Principle of Maximum Entropy and from all the models that fit our training data, selects the one which has the largest entropy [4]. The Maximum Entropy classifier can be used to solve a large variety of text classification problems such as language detection, topic classification, sentiment analysis and more.

Due to the minimum assumptions that the Maximum Entropy classifier makes, we regularly use it when we don't know anything about the prior distributions and when it is unsafe to make any such assumptions. Moreover Maximum Entropy classifier is used when we can't assume the conditional independence of the features. This is particularly true in Text Classification problems where our features are usually words which obviously are not independent. The Max Entropy requires more time to train comparing to Naive Bayes, primarily due to the optimization problem that needs to be solved in order to estimate the parameters of the model. Nevertheless, after computing these parameters, the method provides robust results and it is competitive in terms of CPU and memory consumption [5].

Our target is to use the contextual information of the document (unigrams, bigrams, other characteristics within the text) in order to categorize it to a given class (positive/neutral/negative, objective/subjective etc). Following the standard bag-of-words framework that is commonly used in natural language processing and information retrieval, let $\{w_1,\ldots,w_m\}$ be the m words that can appear in a document. Then each document is represented by a sparse array with 1s and 0s that indicate whether a particular word w_i exists or not in the context of the document. This approach was proposed by Bo Pang and Lillian Lee (2002).

Our target is to construct a stochastic model, as described by Adam Berger (1996), which accurately represents the behavior of the random process: take as input the contextual information x of a document and produce the output value y. As in the case of Naive Bayes, the first step of constructing this model is to collect a large number of training data which consists of samples represented on the following format: (x_i,y_i) where the x_i includes the contextual information of the document (the sparse array) and y_i its class. The second step is to summarize the training sample in terms of its empirical probability distribution:

$$\tilde{p}(x,y) \equiv \frac{1}{N} \times \text{number of times that } (x,y) \text{ occurs in the sample} \tag{1}$$

Where N is the size of the training dataset.

We will use the above empirical probability distribution in order to construct the statistical model of the random process which assigns texts to a particular class by taking into account their contextual information. The building blocks of our model will be the set of statistics that come from our training dataset i.e. the empirical probability distribution.

We introduce the following indicator function:

$$\mathrm{f}_j(x,y) = \begin{cases} 1 & \text{if } y = c_i \text{ and x contains } w_k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We call the above indicator function as “feature”. This binary valued indicator function returns 1 only when the class of a particular document is ci and the document contains the word wk.

We express any statistic of the training dataset as the expected value of the appropriate binary-valued indicator function fj. Thus the expected value of feature fj with respect to the empirical distribution p(x,y) is equal to:

$$\tilde{p}(f_j) \equiv \sum_{x,y} \tilde{p}(x,y) f_j(x,y) \tag{3}$$

If each training sample (x,y) occurs once in training dataset then p(x,y) is equal to 1/N.

When a particular statistic is useful to our classification, we require our model to accord with it. To do so, we constrain the expected value that the model assigns to the expected value of the feature function fj. The expected value of feature fj with respect to the model p(y | x) is equal to:

$$p(f_j) \equiv \sum_{x,y} \tilde{p}(x) p(y \mid x) f_j(x,y) \tag{4}$$

Where p(x) is the empirical distribution of x in the training dataset and it is usually set equal to 1/N.

By constraining the expected value to be the equal to the empirical value and from equations [3],[4] we have that:

$$\sum_{x,y} \tilde{p}(x) p(y \mid x) f_j(x,y) = \sum_{x,y} \tilde{p}(x,y) f_j(x,y) \tag{5}$$

Equation [5] is called constrain and we have as many constrains as the number of j feature functions.

The above constrains can be satisfied by an infinite number of models. So, in order to build our model, we need to select the best candidate based on a specific criterion. According to the principle of Maximum Entropy, we should select the model that is as close as possible to uniform. In order words, we should select the model p* with Maximum Entropy:

$$p^* = \arg\max_{p \in C} \left(-\sum_{x,y} \tilde{p}(x) p(y \mid x) \log p(y \mid x) \right) \tag{6}$$

Given that:

$$p(y \mid x) \geq 0 \quad \text{for all } x, y \tag{7}$$

$$\sum_y p(y \mid x) = 1 \quad \text{for all } x \tag{8}$$

$$\sum_{x,y} \tilde{p}(x) p(y \mid x) f_j(x,y) = \sum_{x,y} \tilde{p}(x,y) f_j(x,y) \quad \text{for } j \in \{1,2,\ldots,n\}. \tag{9}$$

To solve the above optimization problem we introduce the Lagrangian multipliers, we focus on the unconstrained dual problem and we estimate the lambda free variables {λ1,…,λn} with the Maximum Likelihood Estimation method.

It can be proven that if we find the $\{\lambda 1,\ldots,\lambda_n\}$ parameters which maximize the dual problem, the probability given a document x to be classified as y is equal to:

$$p^*(y \mid x) = \frac{\exp\left(\sum_i \lambda_i f_i(x,y)\right)}{\sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)} \tag{10}$$

Thus given that we have found the lambda parameters of our model, all we need to do in order to classify a new document is use the “maximum a posteriori” decision rule and select the category with the highest probability.

Estimating the lambda parameters requires using an iterative scaling algorithm such as the GIS (Generalized Iterative Scaling) or the IIS (Improved Iterative Scaling).

B. *Naïve Bayes Classifier*

The Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem with strong and naïve independence assumptions. It is one of the most basic text classification techniques with various applications in email spam detection, personal email sorting, document categorization, sexually explicit content detection, language detection and sentiment detection. Despite the naïve design and oversimplified assumptions that this technique uses, Naive Bayes performs well in many complex real-world problems.

Even though it is often outperformed by other techniques such as boosted trees, random forests, Max Entropy, Support Vector Machines etc, Naive Bayes classifier is very efficient since it is less computationally intensive (in both CPU and memory) and it requires a small amount of training data. Moreover, the training time with Naive Bayes is significantly smaller as opposed to alternative methods.

Naive Bayes classifier is superior in terms of CPU and memory consumption, and in several cases its performance is very close to more complicated and slower techniques. You can use Naive Bayes when you have limited resources in terms of CPU and Memory. Moreover when the training time is a crucial factor, Naive Bayes comes handy since it can be trained very quickly. Indeed Naive Bayes is usually outperformed by other classifiers, but not always! Make sure you test it before you exclude it from your research. Keep in mind that the Naive Bayes classifier is used as a baseline in many researches.

There are several Naive Bayes Variations. Here we will discuss about 3 of them: Multinomial Naive Bayes, the Binarized Multinomial Naive Bayes and the Bernoulli Naive Bayes [6]. Note that each can deliver completely different results since they use completely different models.

Usually Multinomial Naive Bayes is used when the multiple occurrences of the words matter a lot in the classification problem. Such an example is when we try to perform Topic Classification. The Binarized Multinomial Naive Bayes is used when the frequencies of the words don’t play a key role in our classification. Such an example is Sentiment Analysis, where it does not really matter how many times someone mentions the word “bad” but rather only the fact that he does. Finally the Bernoulli Naive Bayes can be used when in our problem the absence of a particular word matters. For example Bernoulli is commonly used in Spam or Adult Content Detection with very good results.

As stated earlier, the Naive Bayes classifier assumes that the features used in the classification are independent. Despite the fact that this assumption is usually false, analysis of the Bayesian classification problem has shown that there are some theoretical reasons for the apparently unreasonable efficacy of Naive Bayes classifiers. It can be proven that even though the probability estimates of Naive Bayes are of low quality, its classification decisions are quite good. Thus, despite the fact that Naive Bayes usually over estimates the probability of the selected class, given that we use it only to make the decision and not to accurately predict the actual probabilities, the decision making is correct and thus the model is accurate.

In a text classification problem, we will use the words (or terms/tokens) of the document in order to classify it on the appropriate class. By using the “maximum a posteriori (MAP)” decision rule, we come up with the following classifier:

$$c_{map} = \arg\max_{c \in C} \left(P(c \mid d)\right) = \arg\max_{c \in C} \left(P(c) \prod_{1 \le k \le n_d} P(t_k \mid c) \right)$$

Where t_k are the tokens (terms/words) of the document, C is the set of classes that is used in the classification, P(c | d) the conditional probability of class c given document d, P(c) the prior probability of class c and $P(t_k \mid c)$ the conditional probability of token t_k given class c.

This means that in order to find in which class we should classify a new document, we must estimate the product of the probability of each word of the document given a particular class (likelihood), multiplied by the probability of the particular class (prior). After calculating the above for all the classes of set C, we select the one with the highest probability.

Due to the fact that computers can handle numbers with specific decimal point accuracy, calculating the product of the above probabilities will lead to float point underflow. This means that we will end up with a number so small, that will not be able to fit in memory and thus it will be rounded to zero, rendering our analysis useless. To avoid this instead of maximizing the product of the probabilities we will maximize the sum of their logarithms:

$$c_{map} = \arg\max_{c \in C} \left(\log P(c) + \sum_{1 \le k \le n_d} \log P(t_k \mid c) \right)$$

Thus instead of choosing the class with the highest probability we choose the one with the highest log score. Given that the logarithm function is monotonic, the decision of MAP remains the same.

The last problem that we should address is that if a particular feature/word does not appear in a particular class, then its conditional probability is equal to 0. If we use the first decision method (product of probabilities) the product becomes 0, while if we use the second (sum of their logarithms) the log(0) is undefined. To avoid this, we will use add-one or Laplace smoothing by adding 1 to each count:

$$P\ (t \mid c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'}) + B'}$$

Where B’ is equal to the number of the terms contained in the vocabulary V.

C. *Decision Trees*

A decision tree is a tool used for supporting the decision-making process to make good choices. They are part of the family of machine learning methods and allow a hierarchical distribution of a dataset collection. Using a decision tree algorithm, is generated knowledge in the form of a hierarchical tree structure that can be used to classify instances based on a training dataset. An example of the decision tree used in natural language processing are the syntactic tree generated with a grammar (figure 1).

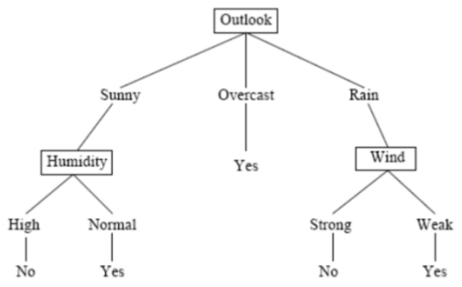

Fig. 1. *The decision tree for “Outlook” attribute*

In the above example the given instances can be divided based on the values it take for the attribute “outlook”. The instances are split based on attributes and the one which gives the maximum information is selected as the decision for that node. Hence in the above example we could say that selecting "Outlook" at the root node gives maximum information at that level. And the edges represent the values the attributes can take and the instances are divided accordingly to each child nodes. The tree can be a m-ary tree depending upon the values that the attributes can take [7]. The attribute selection is based on a heuristic approach that the particular attribute will give the best split at a particular level. But this approach has been successful over the past.

An important part of a decision tree algorithm is the method used for selecting attributes at each node of the tree. Each of these algorithms uses a certain methods for splitting the set of items. The C4.5 algorithm uses for splitting the information gain, a notion based on entropy concept. Another most known algorithm, CART algorithm, uses the Gini impurity that measure how often a randomly chosen item from the dataset would be incorrectly labelled.

The decision trees can be used to classify unseen instances because given a training dataset it can be induced a decision tree from which can be easily extracted rules concerning the dataset. Another advantage offered by the decision trees is the fact that they are able to handle both categorical and numerical [8]. Also they are able to classify large datasets in a short period of time.

III. Results

TABLE I. COMPARISION OF ACCURACIES

Accuracies of various classifiers	
Classifier	***Accuracy***
Conditional Exponential	0.308
Maximum Entropy	0.7985
Naïve Bayes	0.8308
Decision Tree	0.7462

Fig. 2. Table of comparisons of accuracies of various classifiers

References

[1] https://research.google.com/pubs/NaturalLanguageProcessing.html

[2] https://arxiv.org/pdf/1607.01759.pdf

[3] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. Springer

[4] http://blog.datumbox.com/machine-learning-tutorial-the-max-entropy-text-classifier/

[5] https://web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers.pdf

[6] http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/

[7] https://pdfs.semanticscholar.org/64d4/3db78cdf2b7eaacd0e949003f410d28a2ee2.pdf

[8] http://www.imsar.ro/SISOM_Papers_2015/SISOM_2015_A_04.pdf