Proposal for Comparing performance of Text classification algorithms on various fraction of sequential text extracted from original document in a data set

By Yen Hoang, Anbang Xu, Taewoo Kim

1.Project Definition [Yen]

Our project is a problem-based approach type project. The goal of our project is to compare performance of three text classification algorithms on the data sets which have separate training and test data. Also, we would like to evaluate the impact of document size on these classification algorithms by comparing results between using whole document and using just some fraction of sequential text in the original document.

2. Motivation [Anbang]

Text classification is one type of supervised learning. In this setting, usually there is one data set for training or learning process to construct an effective classifier. After this process, that classifier can be used as a tool to classify some other texts. There are many algorithms which can be applied to the text classification such as vector space model using cosine similarity, Decision tree and Naive Bayes algorithm. In all cases, in order to properly train the text classifier, it is crucial that each document in the training data set needs to be represented properly. For example, noises such as stop words, non alphanumeric letters should be removed. Further, stemming can be applied to receive some unified forms of text regardless of part of speech and tenses in each document. Our question starts from here. If some sequential fraction of an original document can present it well, then we do not need to spend precious time to process the whole fraction of that document. For example, if we can read just 10% of sequential chunk from the beginning of some article (original article, not a bag of words), successfully train the text classifier, and it will show similar classification results, we do not need to spend another time to process rest 90% of that article. Nowadays, since data size is getting larger and larger, it costs a huge time to process entire text of each document in the big data set. What if processing just small sequential text of each document is enough to generate similar text classification result? It is important since if we achieve similar classification results when we present small sequential fraction of original text of the original document, the processing speed will boost. Suppose that if we present first 10% of sequential text of each document as a replacement to each document and the similar classification result is presented, then processing speed can be a lot faster since we do not need to process rest text of each document. As far as we researched, there is no publication paper concerning this matter. Therefore, we think it is worthwhile to put efforts to check whether extracting some sequential text of original document makes sense.

3. Data Sets [Taewoo]

We will use the text categorization data sets - Reuters-21578 collection, and Ohsumed collection since they have well sorted text categorization information and each of collection has training and test data set. Reuters-21578 collection contains about 13,000 documents from Reuters. There are 115 categories in this data set. Ohsumed collection contains about first 20,000 medical abstracts from MeSH categories and there are 23 categories in this data set. Since each document in the each category contains similar words, we think that there is high possibility that these documents can be classified as same category, too. We can discuss the result on Evaluation section.

4. Software [Naive Bayes - Yen, Decision Tree - Anbang, Vector Space Model - Taewoo, Other: Taewoo]

Python and some libraries needed to show plots will be used. Also, for proper processing, pre-processing of each document such as removing unnecessary letters and stop words, and applying stemming are needed. For these operations, we will use libraries such as Natural Language Toolkit (NLTK). This library supports above operations. For text classification algorithms, we will write our own code for Vector Space model using cosine similarity, Decision Tree, and Naive Bayes algorithm and use them to execute our experiments.

4.1 Naive Bayes

A Naive Bayes classifier is a probabilistic classifier based on Bayes rule. It relies on very simple representation of documents “bag of words” and conditional independence assumptions between features. In Naive Bayes classifier, we build a Vocabulary that contains all words from the data set. Then, each document is represented as a vector of word frequency with individual words forming features of document. In training phase, we compute the probability of document given category p(d|c) and probability of category p(c) from the training data set, with d is document and c is category. From the conditional independence assumption, p(d|c) can be computed by product of p(wi|c), probability of word wi given class c, for all words wi from the Vocabulary. In prediction phase, to classify document d, we need to compare probabilities of category c given document d, p(c|d), for all categories. From the Bayes rule, we have p(c|d) = p(d|c)*p(c)/p(d) and we can based on parameters from training phase to compare those values and assign d to the category with maximum value, max(p(c|d)).

4.2 Decision Tree

A decision tree is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a decision[1]. According to [2], it is commonly used for gaining information for the purpose of decision-making. It starts with a root node on which it is for users to take actions. From this node, users split each node recursively according to decision tree learning algorithm. The final result is a decision tree in which each branch represents a possible scenario of decision and its outcome. Here learning function is represented by a decision tree. It classifies instances by traversing from root node to leaf node, testing the attribute specified by each path node, then moving down the tree branch according to the attribute value in the given set. This process is the repeated at the sub-tree level.

Algorithm:

1. load learning sets first, create decision tree node “root”, and add learning set S into root node as its subset
2. for root, we compute Entropy[3](root.subset) first
3. if Entropy(root.subset) == 0, then root.subset consists of records all with the same value for the categorical attribute, return a leaf node with decision attribute: categorical attribute value
4. else, then compute information gain for each attribute left(have not been used in splitting), find attribute A with Maximum(IG(S, A)). Create child nodes of this root and add it to root in the decision tree
5. for each child of the root, apply this algorithm(S, A, V) recursively until reaching node that has entropy = 0 or reach leaf node

4.3 Vector Space model using cosine similarity algorithms

This algorithm explores training data set and create vector set A which includes term, its normalized frequency (TF) and inverted document frequency (IDF) per each class (category). Then when it processes each document in the test data set, it creates a vector B which includes same elements in the vector A. After creating a vector B, it calculates cosine similarity distance between each vectors in A and B and based on this similarity score, each document in B is classified as one category.

5. Evaluation Method [Yen]

We will use training data sets to learn the text classifier by using above algorithms and process each document in the test set to verify the result. Since each document in the test sets has its own label, we can easily check the result whether it was successful or not. After we check results by supplying all original test document, we will extract various sequential fraction text of original document and present them as input to algorithms. Sequential fraction text of document can be extracted from the beginning of the original document. Fraction parameter can be as small as 1%. Also, it can be 100%. For example, if the fraction parameter is 30%, we extract 30% of sequential text from the beginning of the original document. We have not yet decided the interval between each fraction. We can compare the result from fraction of each document to the result from original documents and find most similar result. Then, we can use this fraction parameter to test other data sets to check whether this fraction parameter can be also applied generally to other data sets.

6. Milestones and Plan [Anbang]

- Explore Data Set and Analyze Data Set.
- Pre-Process Data Set before applying classification algorithms.
- Fully understand Naive Bayes, Decision Tree and Vector Space model using cosine similarity algorithm.
- Write Python code for Naive Bayes, Decision Tree, and Vector Space model using cosine similarity algorithm.
- Execute three algorithms on the training data set to train classifier in each algorithm and use test sets to check results of each algorithm.
- Extract some sequential fraction text of each document in the data sets and present it to the algorithms and compare those results to the results from original documents. Find the fraction parameter which generates most similar results when applied.
- If time permits, use other data sets to check whether fraction parameter can be applied in general or not.

7. References [Yen, Anbang, Taewoo]

[1] Wiki page, “Decision tree”. http://en.wikipedia.org/wiki/Decision_tree

[2] Paul E. Utgoff and Carla E. Brodley, (1990). 'An Incremental Method for Finding Multivariate Splits for Decision Trees', Machine Learning: Proceedings of the Seventh International Conference, (pp.58). Palo Alto, CA: Morgan Kaufmann

[3] Tom M. Mitchell, (1997). Machine Learning, Singapore, McGraw-Hill

[4] Pazzani, M. J., Muramatsu, J., & Billsus, D. (1996, August). Syskill & Webert: Identifying interesting web sites. In AAAI/IAAI, Vol. 1 (pp. 54-61).