Sentiment Analysis of Twitter Data

Krtin Kumar (260713550)
krtin.kumar@mail.mcgill.ca

***Abstract*— The original version of Tree Kernel for sentiment analysis on twitter data was written by Aggarwal A. et al[1]. We implement a modified version of Tree Kernel and evaluate its performance against the original algorithm and Uni-gram baseline model. In the end we discuss how to improve the Tree Kernel in order to account for semantic information in the text and to achieve a more general algorithm which can be used for more diverse text.**

I. Introduction

Sentiment in Text is an important problem to solve, it has varied usage in different domains. For example, sentiment in news text can help us detect majority sentiment on a particular issue, sentiment in customer reviews of product and service companies can help them to automatically recognize the top key strengths and weaknesses of their products. Twitter is a service which allows people to express their views on domains discussed above and more, through micro-blogging on their websites known as Tweets. Tweets are relatively small in size and thus simple models such as Uni-gram have proven to work well. Work has been done to classify tweets using Tree Kernels as well[1], it has proven to work better compared to uni-gram models. Our objective in this paper is to validate this hypothesis and test another hypothesis that degree of emotion of words, plays an important role in classification. Further, we shall discuss the limitations of our work and further expansion of our work, which shall enable us to move beyond small length sentiment classification to a more general purpose algorithm.

II. Data set and processing

Our data-set consisted of **5513** labelled tweets provided by sanders[1], only tweet id was present in the data as per twitter norms. Actual tweets were obtained using twitter api which left us with **4445** tweets out of which **1479** were not in english, the data set summary is shown in Table I

TABLE I
Twitter data statistics

	Count	Percentage
Neutral	2050	69.1 %
Positive	432	14.5 %
Negative	484	16.3 %
Irrelevant	1479	Non English
Total	4445	

[1] http://www.sananalytics.com/lab/twitter-sentiment/

TABLE II
Data processing

Word Type	Tag Conversion	Method
Acronym	Conversion	Mapping[2]
Contractions	Conversion	Mapping[3]
URL	$\|\|$U$\|\|$	Regex
Emoticons	$\|\|$EN$\|\|$, $\|\|$N$\|\|$, $\|\|NE\|\|$, $\|\|P\|\|$, $\|\|EP\|\|$	Mapping[4]
Hashtags	$\|\|T\|\|$	for @ and #
Punctuations	! $\rightarrow$ EXC, ; $\rightarrow$ SEMI	Manual Mapping
English Word	Tree node$\rightarrow$EN	using Enchant
Stop Word	Tree node$\rightarrow$STOP	using NLTK
Non-English Word	Tree node$\rightarrow$NE	everything else

II-A. Handling Class Imbalance

As seen from Table I class imbalance issue is present only if we perform a tertiary classification, as 70 % of the data is neutral. To handle this we removed neutral tweets so as to make them approximately equal to the number of negative and positive tweets. We made sure that data was removed proportionally from each domain of tweets, namely we have twitter data in four domains or for four companies Apple, Microsoft, Google and Twitter. We will also compare the results with the imbalanced data set for bi-gram and uni-gram models.

II-B. Processing the Data

Unstructured data possess a lot of challenges before it can be reliably fed into a learning algorithm. The steps involved and the complexity often depends upon the type of unstructured data. Particularly, the issues we need to deal with are english slang acronyms (such as converting 'gr8' to great), emoticon conversion, english contractions, URLs, Hashtags, punctuations and stop words. For acronyms, contractions and emoticons we used mapping from an external data source, URLs were given a $\|U\|$ tag, # and @ were given a $\|T\|$ tag and manual mapping was defined for punctuations. The emoticons were mapped from a scale of -1 to 1 with intervals of 0.5, negative values signifying negative emotions and vice versa. Based upon these values five different tags were assigned, the conversion process is summarized in Table II.

II-C. Tree Generation

In addition to data processing defined in the previous section, additional data processing tasks were done in order to obtain

[2] http://www.noslang.com
[3] http://138.197.133.21/contractions.csv
[4] http://138.197.133.21/emoticon.csv

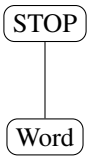


Fig. 1. Stop words Tree

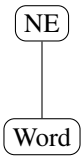
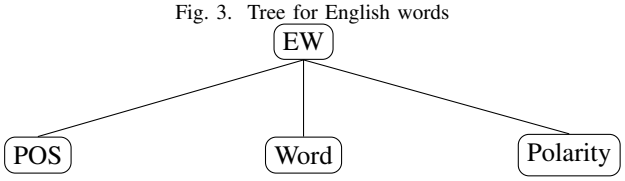


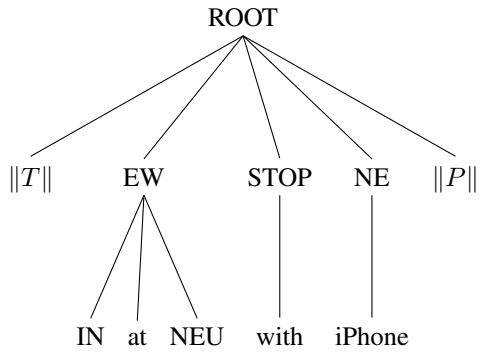
Fig. 2. Non-English words Tree

Fig. 3. Tree for English words



tree representation of a tweet. Stop words were recognized and converted to a sub-tree as shown in Fig 1, non-english words were converted in a similar way (Fig 2). We recognized english words using Enchant[5] and transformed them into a subtree with three nodes, representing the Part of Speech (POS) tag, the word itself and the polarity of the word (Fig 3). The POS tag was obtained using NLTK pos tagger[6] and the polarity of the word was found out using Dictionary of Affect in Language (DAL)[7]. The DAL is a dictionary of approximately 8000 words, scoring each word on the basis of their pleasantness score, imagery and activation on the scale of 1 to 3[3]. To find polarity values we only need the pleasantness scores, we normalize this value and define predefined cut-offs to categorized words as Positive, Negative or Neutral. If the pleasantness score is >0.8 it is categorized as Positive, if it is <0.5 then negative otherwise neutral. We also experiment with another scheme in order to take into account greater granularity, later on we will discuss whether this approach was useful or not. For this modified scheme we took score >0.9 as POS1, >0.8 as POS2, <0.2 as NEG1, <0.4 as NEG2 and <0.5 as NEG3 and the rest as neutral, we will call this scheme as modified tree kernel. An example of a tree is shown in Fig. 4.

Fig. 4. **An example** of a tree



[5]http://pythonhosted.org/pyenchant/
[6]http://www.nltk.org/book/ch02.html
[7]http://compling.org/cgi-bin/DAL_sentence_xml.cgi?sentence=happy

III. Algorithm Details

Now that we have a Tree representation of our tweet, if we can find some measure of similarity between two trees then we can use a machine learning algorithm, which can be represented in the form of a kernel. The partial tree kernel is one such algorithm which can give a measure of similarity, it was first introduced by Moschitti[2] and computes the similarity score by finding the number of common partial trees for all the sub-trees between two given trees and then normalizing the score using equation 1. $K(T_x, T_y)$ is the number of common partial trees between T_x and T_y and $K^{'}(T_x, T_y)$ represents the similarity score.

$$K^{'}(T_1, T_2) = \frac{K(T_1, T_2)}{K(T_1, T_1)K(T_2, T_2)} \tag{1}$$

III-A. Kernel Algorithm in Depth

The algorithm to find the number of common partial trees, is a recursive algorithm using equation 2 and 3. The summation terms in Equation 2 is summation over number of sub-trees in T_1 and T_2, which essentially accounts for the different combinations of sub-trees between T_1 and T_2. Equation 3 finds the number of common partial trees given two sub-trees from T_1 and T_2 with nodes n_1 and n_2, this is done in a recursive fashion. The equation for $\Delta(n_1, n_2)$ is used only if n_1 and n_2 are equal otherwise its value is taken to be zero.

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2}$$

$$\Delta(n_1, n_2) = 1 + \sum_{J_1, J_2, l(J_1)=l(J_2)} \prod_{i=1}^{l(J_1)} \Delta(c_{n_1}[J_{1i}], c_{n_2}[J_{2i}]) \tag{3}$$

J_1 and J_2 represents a set of indices for each child for sub-tree rooted at n_1 and n_2, more precisely the sets are $J_1 = (J_{11}, J_{12}, ..., J_{1i})$ and $J_1 = (J_{11}, J_{12}, ..., J_{1i})$. Note we used the same index i in both the sets because as per equation 3 we only consider sub-trees with equal lengths ($l(J_1) = l(J_2)$ where $l(.)$ represents the length of the set). $c_{n_1}[J_{1i}]$ represents the node of the i^{th} child of a sub-tree rooted at n_1, this allows the algorithm to recursively compute the number of common partial trees. The number of common partial trees is not ideally the best way to get a similarity score, because larger the trees in comparison greater will be the chances of some common partial tree and hence score will be higher. We need some mechanism for penalizing both large trees (i.e. the sub-trees) and large child sequences within the sub-trees. This is achieved using μ and λ decay factors respectively, in equation 4.

$$\Delta(n_1, n_2) = \mu(\lambda^2 \tag{4}$$

$$+ \lambda^{2*l(J_1)-2} \sum_{J_1, J_2, l(J_1)=l(J_2)} \prod_{i=1}^{l(J_1)} \Delta(c_{n_1}[J_{1i}], c_{n_2}[J_{2i}])) \tag{5}$$

This algorithm has an exponential complexity, instead Moschitti also introduced an algorithm which is linear in the number of children for each sub-tree. This algorithm takes into account the fact that common partial trees must have the same length and thus performs a summation over partial trees with the same length of children. The algorithm is explained in detail in [2]

III-B. Implementation Details

For all the algorithms uni-gram (baseline), bi-gram and kernel method we used SVM implementation from sci-kit[8], for uni-gram and bi-gram we used a linear kernel and for tree kernel we used a non-linear SVM with the partial tree kernel as defined earlier. The tree kernel implementation was used from Giovanni's implementation[9]. We divided our data set into 80:20 split for training and testing, on the training set we used 5-Fold cross-validation for calculating the cross-validation performance and finding the appropriate compression ratio for non-linear SVM (i.e. tree kernel). The compression ratio C was chosen as 60 for tree kernel algorithm and 10 and 30 for modified tree kernel for three and two class problem respectively. We chose the decay factors μ = 0.3 and λ = 0.2, ideally we should have chosen this using cross-validation but since calculating the score matrix once took us 25 hours, we decided not to do so given our limited resources.

IV. Results

We compared the results for four models, uni-gram, bi-gram, original tree kernel and our modified version of tree kernel against both binary (Table III) and tertiary (Table V) classification problem. We also compared the results for uni-gram and bi-gram, with an imbalanced data-set (Table IV). For 2 class problem both tree kernels performed better compared to uni-gram and bi-gram, though only performance against bi-gram was statistically significant with p-value of 0.0053. The improvement over uni-gram had a p-value of 0.2465, which is statistically insignificant. For 3 class performance in terms of Test set performance uni-gram performed the best, second was modified tree kernel and then tree kernel but none of the results were statistically significant. If we compare uni-gram and bi-gram performance against balanced and imbalanced data-set, we find that balanced data-set leads to better performance and the difference between accuracy and F1 score is less in balanced data-set.

V. Discussion

Modified vs Original Tree Kernel: In terms of percentage performance our modified tree kernel performed better than the original tree kernel for both binary and tertiary classification problems. But the results were not statistically significant, this could be because the modification influences only a small percentage of tweets. In order to affect larger number of tweets we might have to use dynamic polarity of words and take semantic information into account. We will discuss these methods in detail in section VI.

Tertiary Classification: We see that unlike the original paper[1] uni-gram performed the best, this could be because we didn't choose the decay parameters λ and μ using cross-validation or because authors didn't test tree kernel's performance across different data sets.

Binary Classification: For binary classification the best performing model was the modified tree kernel, though the improvement over the baseline was not statistically significant. This result is not directly comparable with the original paper[1], as the authors didn't report statistical significance of their results.

Balanced vs Imbalanced: We observe that F1 score for the balanced data set is better compared to imbalanced and accuracy is better for the imbalanced data set. This result is intuitive, because in an imbalanced data set the model is likely to learn a hypothesis which predicts the majority class more often, as it will reduce the training error. But this will introduce bias towards the majority class and thus will lead to an increase in recall and drop in precision for that class and a drop in recall for the other classes, hence average F1 score of all the classes is a better measure of performance.

VI. Future Work

Two major weaknesses of our modified tree kernel is that, it still does not account for any semantic information in the text, though this may not be that useful for small texts like tweets, but is important for large text where we need to filter out large irrelevant information. Incorporating semantic information shall thus benefit us in making our algorithm

Model	**Acc Valid**	**F1 Valid**	**Acc Test**	**F1 Test**
Uni-gram	80.7 %	80.7 %	77.1 %	76.8 %
Bi-Gram	71.2 %	71.2 %	69.6 %	69.5 %
Tree Kernel	80.9 %	80.8 %	81.5 %	81.1 %
Tree Kernel Moded	82.1 %	82.1 %	81.5 %	81.3 %

TABLE III
2 Class Classification Results

Model	**Acc Valid**	**F1 Valid**	**Acc Test**	**F1 Test**
Uni-gram	74.7 %	62.2 %	75.3 %	60.9 %
Bi-Gram	72.9 %	50.1 %	75.6 %	53.9 %

TABLE IV
3 Class Classification Results (imbalanced)

Model	**Acc Valid**	**F1 Valid**	**Acc Test**	**F1 Test**
Uni-gram	67.3 %	67.3 %	66.5 %	66.6 %
Bi-Gram	61.1 %	60.6 %	61.4 %	60.3 %
Tree Kernel	65.2 %	65.1 %	63.8 %	63.6 %
Tree Kernel Moded	65.8 %	65.7 %	65.2 %	64.7 %

TABLE V
3 Class Classification Results (balanced)

[8] http://scikit-learn.org/stable/modules/svm.html
[9] http://joedsm.altervista.org/pythontreekernels.htm

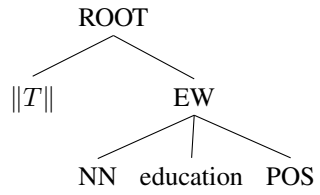


Fig. 5. Why we need semantic information

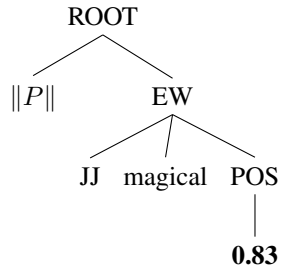


Fig. 6. dynamic degree of polarity

more robust towards different data-sets. Another, problem is that the degree of polarity of the word that we account for is static, a more ideal approach is to make this dynamic. Below, we will discuss some of the possible ways in which we can implement these schemes.

Semantic Information: The polarity of the word education in the tree in Fig 5 does not depend on the word itself but also on its context. Currently our model does not consider the context of the word for its polarity, this adds noise to our model and as a result can learn incorrect weights and increase error. In order to use semantic information we can consider automatic semantic role parsing[4], to label semantic roles in a sentence. This will enable us to find key roles in a sentence that influence the polarity of the sentence, these key roles can either be found out using domain knowledge of semantic roles or through cross-validation. We can also simply add another node with the semantic roles, to let our model automatically determine the score.

Dynamic Degree of polarity: One way to make the degree of polarity dynamic is to add a node to the polarity node in the tree (Fig 6) with the polarity value from DAL. Then modify the tree kernel algorithm and check for DAL node, if it encounters one then back trace it and multiply the weight to the λ term of the previous score.

VII. Conclusion

Our objective was to verify two hypothesis, first that partial tree kernel is an improvement over the baseline. For binary classification partial tree kernel did show an improvement of 4.5 % over the baseline (these results are similar to those obtained in [1]) but the results were not statistically significant. Hence, we cannot accept or reject the hypothesis, a similar conclusion can be made for tertiary classification. Our second hypothesis that the modified tree kernel is an improvement over the original tree kernel, also cannot be accepted or rejected, even though the modified tree kernel showed an improvement of about 1 % based upon cross-validation score and a better generalization for the tertiary classification problem. This is because the results were not statistically significant. Even then the results indicate that degree of polarity can have a positive impact on classification, probably the dynamic degree of polarity algorithm can help us answer these questions more concretely. We can also experiment with a larger data-set to accept or reject the null hypothesis.

References

[1] Agarwal, Apoorv, et al. "Sentiment analysis of twitter data."Proceedings of the workshop on languages in social media. Association for Computational Linguistics, 2011.

[2] Moschitti, Alessandro. .Efficient convolution kernels for dependency and constituent syntactic trees..European Conference on Machine Learning. Springer Berlin Heidelberg, 2006.

[3] Whissell, Cynthia, et al. .A dictionary of affect in language: IV. Reliability, validity, and applications."Perceptual and Motor Skills 62.3 (1986): 875-888.

[4] Gildea, Daniel, and Daniel Jurafsky. .Automatic labeling of semantic roles.Çomputational linguistics 28.3 (2002): 245-288.