Sentiment Analysis with SVM algorithm for trees

Project for the Advanced Topics in Computer Science Course - Univeristy of Padua

Alessandro Pegoraro, *matr.* 1240466
alessandro.pegoraro.4@studenti.unipd.it

September 23, 2021

Abstract

The sentiment analysis is a text classification task that aims to predict the attitude of people towards the object described in the text.
Most problem that we can find proposed online are of binary sentiment analysis, in which the classification default to two classes: "Negative" and "Positive".
In this project we analyse possible solution for a fine-grained sentiment analysis problem with five possible classes and its derivative restricted to only two classes.

1 Dataset and Problem

This project was centered around the Stanford Sentiment Treebank (SST) dataset that is a corpus with fully labeled parse trees and consist of 11,855 single sentences extracted from movie reviews. It was parsed with the Stanford parser generating 215,154 unique phrases each labeled by three human judges.
Each phrase does not represent a complete sentence, but instead a group of labeled words with a grammatical structure that in turn is labeled.
Each root sentence is composed of phrases, and in the fine-grained SST-5 version is labeled as either:

- *0: negative*
- *1: somewhat negative*
- *2: neutral*
- *3: somewhat positive*
- *4: positive*

While in the binary SST-2 version the *neutral* sentences are discarded leaving us with:

- *-1: negative* or *somewhat negative*

vs

- *1: somewhat positive* or *positive*

1.1 Related work

The peculiarity of this dataset is obviously its entries representation as a tree and how each node is labeled, a good model should be able to classify any text, not only the movie review.
Making use of this features an initial model was published in 2013 [6] which used a Recursive Neural Tensor Network and obtained an accuracy score of 45.7% for SST-5 and 85.4% for SST-2.
While looking at the latest solutions proposed for SST-5 and SST-2 we can see how all the successive models to [6] maintain the underlying idea of using Neural Network with the addition of LSTM [1] for SST-5 (56.20% of accuracy) or Transformer for SST-2 [3] (97.50% of accuracy).

With this project I wanted to test if it would be possible to obtain similar result using a Support Vector Machine and in particular a SVM that make use of tree kernel.

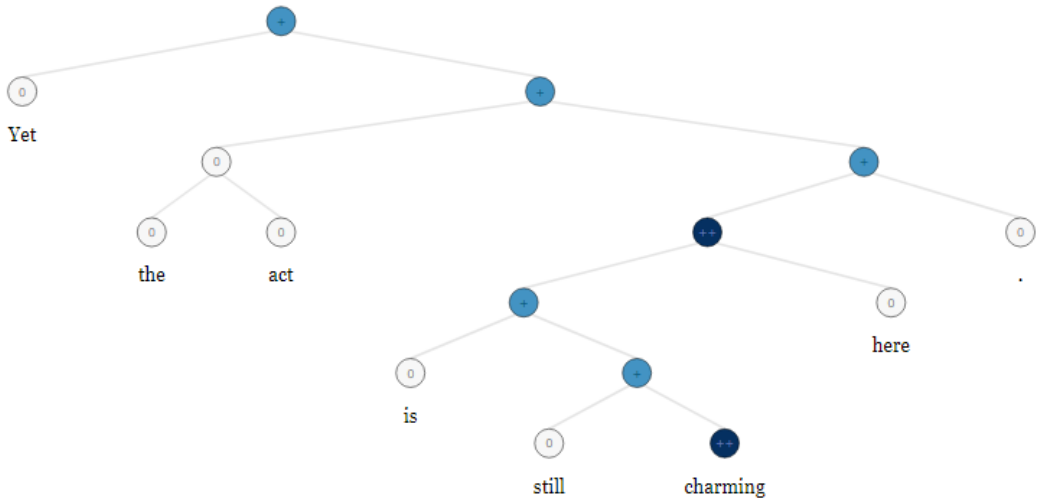

Figure 1: Example of a parse tree

2 SVM

For the first part of the project I tried to measure how having data represented as a parse tree instead of simple plain text, could improve the accuracy of an SVM classifier.
I used the python library scikit-learn, in particular I used the *CountVectorizer* and *TfidfTransformer* modules for feature extraction and the *SVC* module as the C-support Vector Classifier.

2.1 SVM Models

Model1: The first model implemented was a simple classifier that used only the plain text encoded using *CountVectorizer* and *TfidfTransformer*.
This model was used as the stepping stone to compare and discard any new model with less accuracy.

Successive test like trying to learn all the phrases' label and trying to reconstruct the root sentences using the phrases, yield no better results than **Model1**.
The introduction of Part of Speech tagging lead to the improved:

Model2: This second model instead of using each words of the root sentence made use of their Part of speech tag, encoded using *CountVectorizer*. This in combination with the labels of the first two children of the root node created the feature vector used to train this new model

I tried to further develop **Model2** using the labels of the nodes more depth in the parse tree, but the accuracy only worsened.

Model3: This third model was an improvement of **Model2**, the use of *TfidfTransformer* is added for the embedding
Further research for new feature to insert like adding more than 2 label or the height of the tree did not yield any improvement in accuracy.

2.2 SVM results

SVM	SST-5	SST-2
Model1	40.04%	80.61%
Model2	49.50%	91.32%
Model3	52.44%	92.42%

Table 1: Accuracy score of each SVM model in SST-5 and SST-2

As we can see on Table 1 the overall improvement between the models is maintained from SST-5 to SST-2.
The final model **Model3** reach a good enough accuracy in both SST-5 and SST-2, but still it is not near the value obtained by [1] and [3] as it lacks around five percentage points.

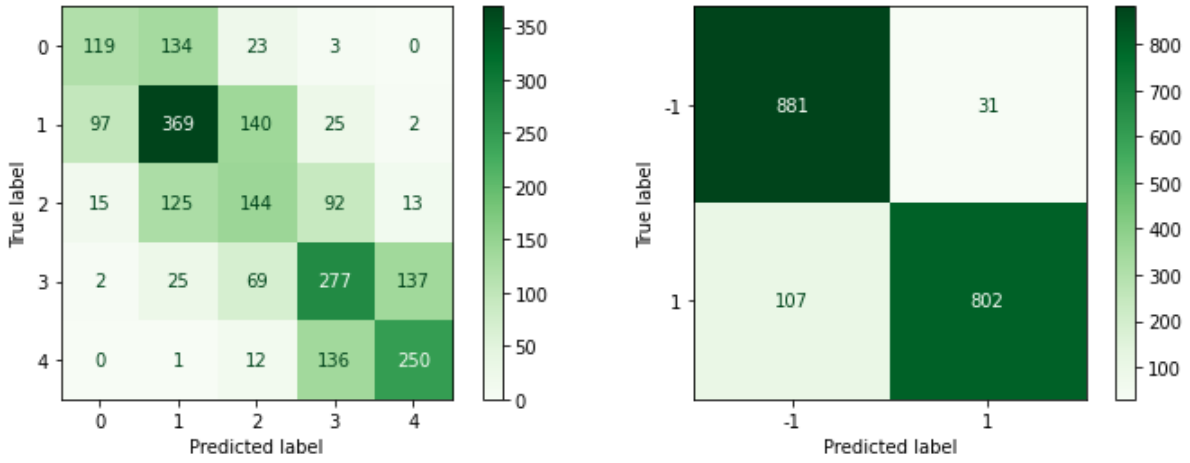

Figure 2: Confusion matrices of **Model3** in SST-5 and SST-2

If we observe the confusion matrices of **Model3** in Figure 2 we can see how most of the error on the prediction are made between *negative* and *somewhat negative*, *somewhat negative* and *neutral* and between *somewhat positive* and *positive*.
This could be seen as a data annotation problem given that the label were decided by only 3 human judges, further experimentation with the Live Demo shows that the difference between *somewhat positive* and *positive* is not always consistent, and the same for *negative* and *somewhat negative*.
As an example the word "happy" is labeled as *positive*, but most of the phrases with it are labeled as *somewhat positive*, by contrast the word "good" is labeled as *somewhat positive* and the phrases with it are labeled as *positive*.
To test if this "bias" was present in my models I created an ibrid dataset "SST-3" collapsing *negative* and *somewhat negative* together and *somewhat positive* and *positive* together. With this new ibrid dataset **Model3** was able to obtain an accuracy of 75.33%.

3 Tree Kernel

The limitations of the scikit-learn library were that its implementation of the SVM could only accept feature array of fixed length, and its entries could only be numerical.
To tackle this problem and test some function of similarity between tree i.e. tree kernel I used the SVM-LIGHT-TK library, a C library that implement multiple tree kernels.

Tree kernel: Every tree kernel in SVM-LIGHT-TK is either an exact implementation or an ibrid implementation of these kernel.

3.1 Subtree kernel ST:

Proposed [8] in 2002, it is a weighted sum of the number of **proper** subtree that two binary parse tree match.

$$K_{subtree}(T_1, T_2) = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} C(t_1, t_2)$$

With T_1 and T_2 the set of all nodes of their respective trees and $\mathbf{C}(t_1, t_2)$ defined as follow:

- $C(t_1, t_2) = 0$ if t_1 and t_2 are different
- $C(t_1, t_2) = 1$ if t_1 and t_2 are the same with only leaf children
- $C(t_1, t_2) = C(t_1.left, t_2.left) * C(t_1.right, t_2.right)$ if t_1 and t_2 are the same and are not pre-terminals node

with $t_i.left$ and $t_i.right$ the corresponding left and right children of t_i.

3.2 Subset tree kernel SST:

Proposed [2] in 2002, it is a weighted sum of the number of subset tree that two binary parse tree share.

$$K_{subset}(T_1, T_2) = \sum_{s \in m} h_s(T_1) h_s(T_2) = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} C(t_1, t_2)$$

With $h_s(T_i)$ the number of times the subset tree $s \in m$ occurs in T_i and $\mathbf{C}(t_1, t_2)$ defined as follow:

- $C(t_1, t_2) = 0$ if t_1 and t_2 are different
- $C(t_1, t_2) = 1$ if t_1 and t_2 are the same with only leaf children
- $C(t_1, t_2) = (1 + C(t_1.left, t_2.left)) * (1 + C(t_1.right, t_2.right))$ if t_1 and t_2 are the same and are not pre-terminals node

with $t_i.left$ and $t_i.right$ the corresponding left and right children of t_i.

3.3 Partial tree kernel PT:

Proposed [4] in 2006, it is a weighted sum of the number of subtree (not only **proper**) that two binary parse tree match.

$$K_{partialtree}(T_1, T_2) = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} C(t_1, t_2)$$

With $\mathbf{C}(t_1, t_2)$ defined as follow:

- $C(t_1, t_2) = 0$ if t_1 and t_2 are different
- $C(t_1, t_2) = 1 + \sum_{J_1, J_2, |J_1| = |J_2|} \prod_{i=1}^{|J_1|} C(ch_{t_1}[J_{1i}], ch_{t_2}[J_{2i}])$

with J_{1i} and J_{2i} index associated with the child ch_{t_1} and ch_{t_2} respectively.
Given that we work with binary parse trees we will only explore J_{11}, J_{21} and J_{12}, J_{22}.

4 Experiment with Tree Kernel

The SVM-LIGHT-TK library provided the following Tree kernels:

- **SSTK kernel**: Fast Partial Tree kernel within first tree level + SubSet Tree kernel for the remaining tree level [5]
- **ST kernel**: SubTree kernel [8]
- **SST kernel**: SubSet Tree kernel [2]
- **SST-BoW**: SubSet Tree Kernel + Bag of Word with leaves as features [9]
- **PT kernel**: Partial Tree kernel [4]
- **IBRID**: Partial Tree kernel + no leaves contribution
- **STRING**: String representation comparison [7]

I tested each kernel with 3 different type of preprocessing of the trees:

- **Normal:** No preprocessing
- **PoS:** word on the leaf replaced with their Part of Speech tag
- **Empty:** word on the leaf removed (I could not really remove them so I replaced them all with a symbol)

4.1 Tree Kernel results

	SST-2		
Kernel	**Normal**	**PoS**	**Empty**
SSTK	96.21%	95.83%	94.45%
ST	78.20%	75.34%	75.01%
SST	96.21%	95.83%	94.45%
SST-BOW	96.10%	95.44%	93.79%
PT	95.50%	95.33%	94.89%
IBRID	97.20%	97.42%	96.81%
STRING	96.37%	92.20%	92.20%

Table 2: Accuracy score of each Tree kernel with different preprocessing in SST-2

As we can see form Table 2 all but the SubTree kernel outperform the SVM **Model3** in SST-2.
The **Empty** preprocessing did not brought up any improvement in any kernel. While the **PoS** preprocessing only helped the **IBRID** kernel, but with it was able to reach the current state of the art accuracy [3] shy of only 0.08 point percentile.

5 Usage

5.1 File .ipynb

To run any python code proposed, load the file on any *Jupyter Notebook* and execute the code.
Pay attention that most of the times to run the command *spacy.load()* it will need to restart the runtime. The file "*GENERATE_TEST_TRAIN.ipynb*" will try to write file into the "/content/" directory if you do not have this directory please remember to change the path.

5.2 SVM-LIGHT-TK

Download the source code for the SVM-LIGHT-TK library.
Extract the folder "svm-light-TK-1.5" in any directory, then open a terminal inside it and run the command:

```
$ make
```

this will compile all the project and output the two objects *svm_learn* and *svm_classify*.

Running the command:

```
$ ./svm_learn -t -F [TreeKernel] [file with training data] [path to save model learned]
```

will generate a new model, the option [TreeKernel] can assume the following values:

- -1 SSTK kernel
- 0 ST kernel
- 1 SST kernel
- 2 SST-BoW kernel
- 3 PT kernel
- 4 IBRID kernel
- 6 STRING kernel

It is possible to skip the training phase as I already produced each possible model with different preprocessing as described in 4.

Running the command:

```
$ ./svm_classify [file with test data] [path to model]
```

will try to classify the data provided following the learned decision rules in the model provided.
the terminal will output the accuracy score, precision/recall and F1 score.

References

[1] S. Brahma. "Improved Sentence Modeling using Suffix Bidirectional LSTM". In: (2018).

[2] M. Collins and N. Duffy. "New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron". In: (2002).

[3] H. Jiang et al. "SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization". In: (2021).

[4] A. Moschitti. "Efficient convolution kernels for dependency and constituent syntactic trees". In: (2006).

[5] A. Moschitti, S.Quarteroni, and R. Basili. "Exploiting syntactic and shallow semantic kernels for question answer classification". In: (2007).

[6] R. Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: (2013).

[7] J. Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. 2004.

[8] S.V.N. Vishwanathan and A.J. Smola. "Fast kernels on strings and trees. In Proceedings of Neural Information Processing Systems". In: (2002).

[9] D. Zhang and W. S. Lee. "Question classification using support vector machines". In: (2003).