Identification of candidate variables & their metric calculation

Project Report

CSE299

Section 1

Group members:

Ridwan Bin Khalid - 1831785642
Syed Zubair Naved – 1812823042
Tanajjul Ahmed – 1831199042

Submitted on: 01/05/22

Submitted to:
Nabeel Mohammed

Contents:

Abstract

The project at hand was to create a solution based on a Natural Language Processing technique that would correctly identify possible variables in text files and produce their spans as an output, ultimately using this output to measure the accuracy of our solution in correctly identifying candidate variables. This paper describes the problem presented, how we devised its solution and the challenges we faced. To start with, we gathered 100 text files of varying length and complexity. To tackle a problem of this sort we needed to work with Natural Language processing tools so we could take into account the linguistic aspects of texts and hence used NER (Name Entity Recognition) annotations to tag potential variables in the text files we gathered. Name Entity recognition works by scanning documents and tagging the appropriate keywords such as name, place, etc and organizing them to be used at convenience. To proceed, we used an open source library known as spaCy to create the baseline solution as we trained it using dummy data to understand the criteria of a candidate variable. Lastly we took the output from our system and compared it to the annotations we had manually set to calculate our desired metrics- precision, recall, accuracy and F1. To complete the calculations for this metrics we took the aid of another library known as sklearn to import confusion matrix (a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known) from our data set which would allow us to correctly identify True positive, True negative, false positive and false negative which are crucial in each of the metric calculation. Our ultimate accuracy for the model came out a modest 77.83% however our precision was a little lacking at 0.17. A problem we however failed to tackle was the correct implementation of a REST API and the calculation of an additional metric MUC to factor in the partial matches produced.

Introduction

The problem provided, was to find possible variables in a text file and give the spans as output, furthermore use relevant metrics to show how well our solution was working.

Specification of deliverables

- 100 text files filled with any text containing characters or numbers with 33 of them considered as easy, 33 medium, and the rest as hard
- Provide annotations for these texts and consider them as the ‘hard truth’
- Provide our approach or testing method
- Metrics to measure the success of the solution
- A baseline solution that identifies the correct spans
- A rest API
- Gives the output in json format
- A script to compare the output of the program with the hard truth
- Further metrics to define partial matches and multi-word expressions
- An improvement upon the baseline solution in terms of the metrics used or overall efficiency

Technologies used:

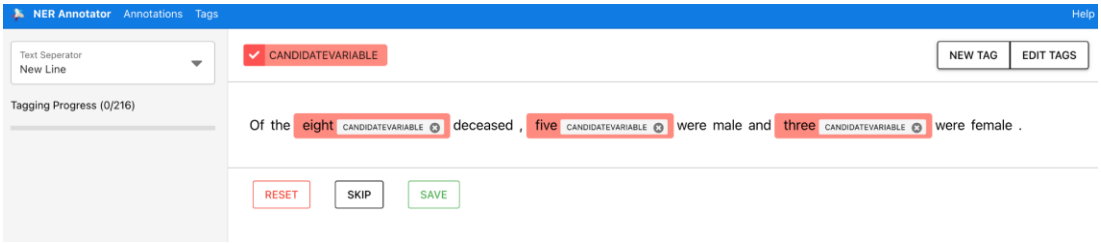
NLP - Natural Language Processing.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly even in real time. It has several tasks that break down human text and voice data in ways that help the computer make sense of what it's ingesting.

There are various NLP tools and approaches available. Python has a wide range of tools and libraries to attack the NLP tasks, these are mostly found in Natural Language Toolkit or NLTK, which is an open source collection for libraries , programs, and resources to build these NLP programs.

Named Entity Recognition is a part of NLP, one of the most important methods to extract relevant information from a text document. NER annotation helps to recognize the entity by labeling various entities like name, location, time and organization.

For our project, we created 100 text files as our training dataset. To identify the candidate variables from the texts we used Name Entity Recognition which identifies words or phrases as useful entities. NER identifies ‘Kentucky’ as a location or ‘Fred’ as a man's name.



We used an online NER annotator (https://tecoholic.github.io/ner-annotator/) to annotate our texts, by simply uploading a text on the website and tagged each word by the name 'candidate variable' and downloaded the annotation of all tagged entities in a json file which was later used to train and test our data. Below is a sample output: a json file that states the text and annotations of the tagged entities.

{'annotations': [['Of the eight deceased, five were male and three were female.,
  {'entities': [[13, 21, 'CANDIDATEVARIABLE']]}

spaCy:

To create our own NER model for training and testing, we used spaCy which is an open-source software library for advanced natural language processing. A named entity is a "real-world object" that's assigned a name – for example, a person, a country, a product or a book title. spaCy can recognize various types of named entities in a document, by asking the model for a prediction. After installing spaCy and importing our pre annotated json 'training data' and 'validation data' we converted those json files into docbin which is a spacy object.

```
import spacy
from spacy.tokens import DocBin
from tqdm import tqdm

nlp = spacy.blank("en") # load a new spacy model
db = DocBin() # create a DocBin object
```

Testing protocol:

spaCy's tagger, parser, text categorizer and many other components are powered by statistical models. Every "decision" these components make for example, which part-of-speech tag to assign, or whether a word is a named entity is a prediction based on the model's current weight values.

To train a model in Spacy, we need training data. We used 25 text files which were converted into json files as our training data, and the other text files were used to validate and test our model. We had to use a validation data set because we didn't want to test the program based only on the data it was trained on, otherwise we wouldn't know how well it is generalizing. We didn't want

our model to give us variables in a memorized form hence we made sure that the training data was representing the data we wanted to process.

Then we created a Spacy config file from the CLI where we wanted our pipeline as ner and language as english. Once our config file is created we then train the config file using spaCy. In the command we call both training and validation data doc files that we previously created. When we run this command, it created the pipeline and vocabulary, then finished initializing the nlp object and pipeline components. Then we have two new models successfully saved in the output directory: best model and last model. Below is the screenshot of the train command output once our model training finishes.

```
============================ Initializing pipeline ============================
[2022-04-30 10:59:36,364] [INFO] Set up nlp object from config
[2022-04-30 10:59:36,376] [INFO] Pipeline: ['tok2vec', 'ner']
[2022-04-30 10:59:36,380] [INFO] Created vocabulary
[2022-04-30 10:59:36,381] [INFO] Finished initializing nlp object
[2022-04-30 10:59:36,602] [INFO] Initialized pipeline components: ['tok2vec', 'ner']
✔ Initialized pipeline

============================== Training pipeline ==============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     41.17    6.45    5.03    8.99    0.06
 11     200         86.31   1720.56   99.72  100.00   99.44    1.00
 26     400        103.39     82.25   99.72   99.44  100.00    1.00
 44     600         63.00     60.49   99.72   99.44  100.00    1.00
 68     800         80.18     69.71   99.72   99.44  100.00    1.00
 98    1000         62.71     72.55   99.72   99.44  100.00    1.00
135    1200         24.29     62.33   99.72  100.00   99.44    1.00
182    1400         38.64     74.15   99.72  100.00   99.44    1.00
237    1600         37.78     83.66   99.72  100.00   99.44    1.00
304    1800         54.15    100.37   99.72  100.00   99.44    1.00
389    2000         79.94    123.70   99.72  100.00   99.44    1.00
✔ Saved pipeline to output directory
model-last
```

Then we load the best model and create a doc object where we can input any text or text file and by calling the render command we can identify and print the candidate variables of the text. A sample text file is used as the testing parameter of the `doc = nlp_ner` function in the code.

Results and outcomes:

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.

Then we use another python library sklearn in spaCy to import confusion_matrix. Sklearn is used for statistical modeling. To plot the confusion_matrix we have used matplotlib and numpy libraries. We wrote functions to create target (y_true) and predicted (y_pred) vectors. The function that generates the confusion matrix, gets the ground truth and predicted labels and compares them.

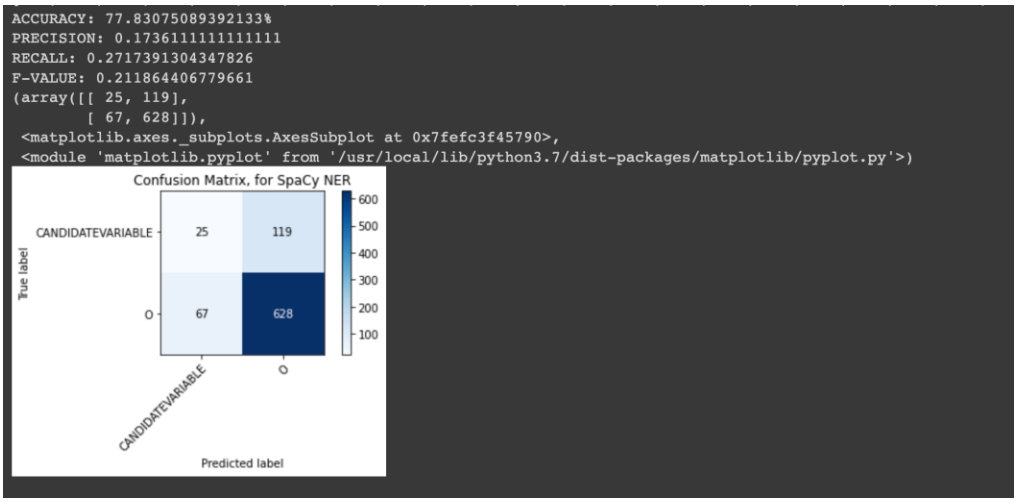
Along with generating the confusion matrix, we also plotted it to give a visualization of the total True positives, False Positives, False negatives, True negatives found.

True positive: 25 ,these are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.
False Positive: 119 , When actual class is no and predicted class is yes.
False negative: 67, When actual class is yes but predicted class in no.
True negative: 628, These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

Using these numbers we were able to calculate accuracy, precision, recall and f-score value.

Accuracy	77.83075089392133%
Precision	0.1736111111111111
Recall	0.2717391304347826
F - Value	0.211864406779661

The screenshot below is the output from the command.



Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. But it is not enough to correctly predict if ours is a good model or not , for that we need more parameters. Such as,

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate. But we got 0.1736 precision which is okay considering the small amount of data, and our false positives were 67.

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. Our recall score is 0.271.

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Our F1 score is 0.21.

Challenges and future work:

We didn't have enough data points to improve the accuracy and also had some difficulty understanding the problem, creating the API was time consuming because we couldn't figure out what the basis was for the API, some data may be mislabeled while tagging candidate variables. It took a lot of time to do research on spacy more like to find out about spacy, we had to read the documentation quite a few times to make sure we were going in the right direction. Infact, before choosing spacy we chose Gensim to do the tokenization which we later figured out was a completely wrong library to choose. We also had trouble tagging the candidate variables and didn't know what annotator to use.

Our false positive and false negative score should have been significantly less if we wanted a better score of precision, recall and f1, but considering we only used 25 data sets to train our model our scores are okay comparatively. Although, our scores should have improved little by little every week with better solutions, but since it took us a while to figure out the problem we didn't get enough time to work on better solutions to improve our scores. Moreover, it's also possible the data we used to train our model was not accurate enough or similar to the data we used to test our model. To train our model accurately to the point where we needed scores close to 80 percent we needed to use more training data.

It was also difficult to understand the regression model best fitted for our project. The main issue while implementing the metrics was the confusion matrix, before this course we had no idea about what this was and what its supposed to do, after doing a lot of research we got to know it was a must to find the confusion matrix in order to implement the metrics such as accuracy, precision, f1 and recall. We were also asked to show the MUC score but we couldn't tackle it even after reading quite a few citations and research papers. We couldn't find a specific formula which we could enact in our project. We also wanted to find the MCC score to further improve the scores but we got stuck at precision, recall and f1.

Conclusion:

From the experiment, we can say that our training model is much better than the validation model although it can be improved significantly. With no basic machine learning knowledge it was tough for us to figure out the metrics and how to implement them, as the weeks of our semester went by we invested more time into figuring out the metrics, but unfortunately we couldn't implement it until last week. Overall, we have learnt a lot from this project, especially about NLP and how vastly it works. All my teammates are interested in working with NLP now and if we can we will like to work on it further in CSE499A.

Github Link: https://github.com/tanajj17/Identification-of-Candidate-Variables