NLP Project Report

Team Exsilio

Vigneshwaran Sampath (vxs180021)
Vignesh Viswanathan (vxv190028)

1. Introduction:

This report highlights and describes our system for the SemEval-2010 dataset task to classify the multi-way relations between the tagged entities. First, the data features are selected for the model from the dataset and then fed into the model. For the model, we have utilized the Decision Tree Classifier and improved it by using Bagging and Boosting. The classification is performed using a number of features that capture the context, semantic affiliations, and possible hidden features between the tagged entities. Our approach achieved the best accuracy of 40%.

2. Problem Description:

The ideation behind this project was to design and implement models for extracting the relations between two named entities in the SemEval-2010 Dataset. The SemEval-2010 dataset contains over 8000 examples with about 18 different types of relations. For the dataset, the following annotations hold:

a) The spans of the two named entities between which the relation holds(indicated by the delimiters e1 and e2)

b) The relation between the two entities also marked with its directionality.

Now, Given two nominals embedded in a sentence, the problem requires identifying which of the following nine semantic relations holds between the nominals: ***Cause-Effect, Instrument-Agency, Product-Producer, Content-Container, Entity-Origin, Entity-Destination, Component-Whole, Member-Collection,***

Message-Topic, or Other if no other relationship is appropriate. To disambiguate which relations holds in which sentence requires some textual and semantic cues in the sentence. These cues can be extracted using the rich feature extraction using the various libraries in natural language processing. Moreover, as the directionality is considered as a separate class identifier, the model also needs to disambiguate the directional property of the relation.

The problem description also states to measure the performance of the model employed for the relation classification in terms of any performance metric. The metric should quantify the sense of directionality in the type of relation classified.

3. Proposed Solution:

We cast the task of determining a semantic relation and its direction as a classification task. The classifier deploys a single classifier for classifying both the relation and it's direction. We have employed a total of 4 features to be used by the relation-direction classifier. These features can be grouped as: hypernyms from WordNet, Shortest Dependency parse and the Part-of-Speech tags of the window. All the features were treated as Feature Type: Value which were then presented to the Decision Tree Classifier, encoded with the CountVectorizer. The class labels for the multi-way classification were encoded using a LabelEncoder, to convert the 18 relation types into 18 numerical categories to be classified.

a) Linguistic Features:

Before we can extract the Lexical features, we first convert the line containing the entities into a window. This window spans from one word before the word tagged as E1 to one word after the word tagged E2, ofcourse, if E1 and E2 are not in either the start or end of the sentence. For this created window, we extract the words and their parts of speech. This is done as the words between the nominals tagged can be a strong indicator of the type of the relation. As a simple example, words such as *into, produced, caused, contains,* are likely to occur in relations such as *Entity-Destination*, *Product-Producer*, *Cause-Effect*, *Member-Collection* respectively.

After this window is created, we extract the Part of Speech sequence for the words between the nominals. This is particularly useful for the relation types such as *Member-Collection*, wherein prepositional phrases such as: *of, in* could be distinctly utilized. Also, this window creation also reinforces the classification of relations such as *Product-Producer* and *Entity-Origin* which for the most part do not have any intervening tokens in between, for example: *corn flour, mustard oil.*

For capturing of long distance relationships between the phrases in a sentence the shortest dependency parse feature is used. Here, the phrasal constituents do not play a role, instead the syntactic structure of the sentence is described solely in terms of the words(or lemmas) in the sentence and the associated grammatical relations that hold among the words. Here too, we employ a shortest dependency path approach instead of the entire dependency parse of the whole sentence. This helps us because this approach will most probably classify the nominals between which the relation holds as the Root and the resultant parse structure will better represent the relation and its direction.

b) <u>Nominal Entities Background Knowledge:</u>

Although all the contextual knowledge presented in the previous section can be useful and critical to identify the semantic relation present in most of the relations, sometimes the background knowledge can help us disambiguate the types of the nominals involved in the sentence.

To utilize this word sense disambiguation we use *all of the hypernyms* of the tagged entities of the sentence. This can help us in cases for example, if one sentence contains the tagged entities as *writer-book* and the other sentence has *author-article*, both can be correctly identified to be of type *Product-Producer(e2, e1)*. This is extracted using WordNet.

c) <u>Decision Tree Classifier and Methodology:</u>

Decision Trees (DTs) are a supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features and the deeper the tree, the more complex the decision rules and the fitter the model. Some of the advantages of using a Decision Tree Classifier are:

a) It is simple and not a bloated model such as an SVM, which makes it easier to train on a large corpus and the prediction task quickly.

b) Capable to handle multi-output and multi-class problems such as this one.

To pass our extracted features as an input to the Decision Tree, we implement a CountVectorizer. The CountVectorizer provides a simple way to both; tokenize a collection of text documents or features and build a vocabulary of known words, and also to encode new documents using that vocabulary. The encoded vector is returned with the length of the entire vocabulary and an integer count for the number of times each word appeared in the document. The CountVectorizer produces a lot of zeroes and hence is sparse. This sparse vector is handled using the scipy.sparse package.

Now, for the output from the Decision Tree, we encode our target variables viz. *The Relation* using a LabelEncoder. The LabelEncoder can be used to transform non-numerical labels into numerical labels and encodes them with a value between 0 and number_of_classes - 1, 18 in our case.

To improve the accuracy of the model, Ensemble models such as the Adaptive Boosting and the Bagging Classifier were utilized. This certainly did improve our predictions and the results are reported in the Results and Analysis section below. This helped us conquer some bias and the greedy classification done by the Decision Tree and tackle the imbalanced number of classes present in the training data.

4. <u>Implementation Details:</u>

a) <u>Programming Tools and Frameworks Used:</u>

This entire project along with all it's tasks and the Decision Tree were coded in **<u>Python</u>**. The IDE used was a combination of Pycharm and Google Colab Notebooks.

The data from the Corpus was parsed into a DataFrame using Pandas in Python.

For the Feature Extraction we utilized libraries such as NLTK for Tokenization and the SpaCy API for Tasks such as Lemmatization, POS Tagging and the Dependency Parsing. SpaCy was also used for NER Recognition. All the Hypernyms were extracted using the WordNet corpus in NLTK.

For the Decision Tree Classifier, we used Scikit-Learn's Tree library. Scikit-Learn's feature_extraction and preprocessing libraries were utilized for the CountVectorizer and LabelEncoder respectively. We also utilized the Accuracy_score metric available as part of this library.

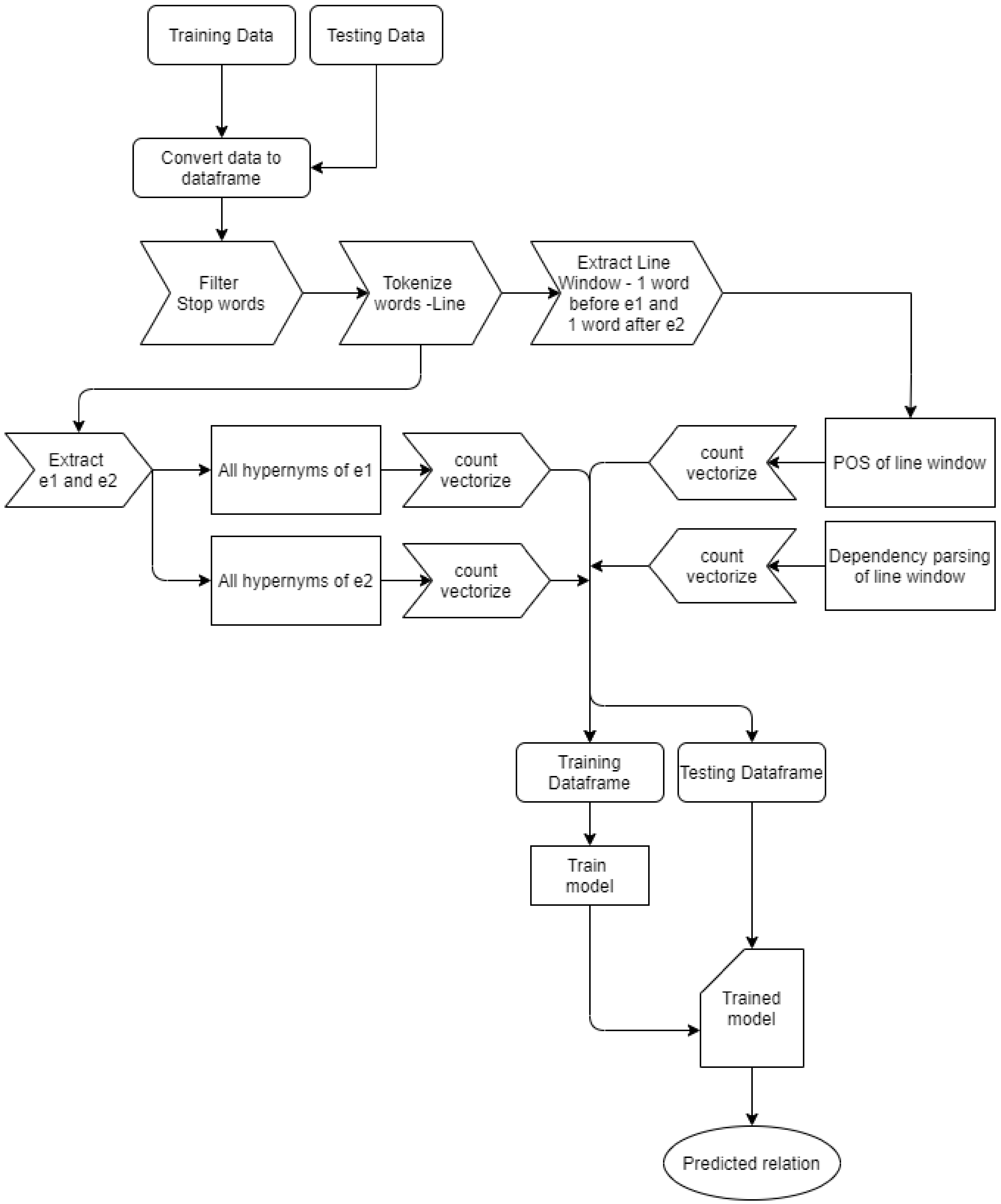
The Bagging and Boosting were done using the Ensemble methods available as part of Scikit-Learn.

b) Architectural Diagram:

The following diagram shows how all various tasks and modules are integrated for this project. All our features were selected for giving us a good mix of linguistic features and background knowledge of the nominals. Also, using the shortest window and shortest dependency parsing, we do cut off a lot of inaccurate and misleading root word tagging in the sentences other than the nominals. More or less, the distinctive window lengths in certain relation types supports this model.

For parsing the data from the Corpus, we used a DataFrame approach and the Filtered Tokens refers to the tokens of the sentence without the actual entity opening and closing tags.

Most of our performance tuning was done around the model, like changing the depth of the Decision Tree and handling the splitting criteria for the same.


Training Data
Testing Data
Convert data to dataframe
Filter Stop words
Tokenize words -Line
Extract Line Window - 1 word before e1 and 1 word after e2
Extract e1 and e2
All hypernyms of e1
count vectorize
All hypernyms of e2
count vectorize
count vectorize
POS of line window
count vectorize
Dependency parsing of line window
Training Dataframe
Testing Dataframe
Train model
Trained model
Predicted relation

c) Results and Error Analysis:

An example of the Parsed Features from a Test sentence is:

```
Sentence            :  To the right of that is a <e1>cup</e1> with <e2>soil</e2> inside.


POS tags            :  DET, NOUN, ADP, NOUN, ADP

Dependency parsing  :  det, ROOT, prep, pobj, advmod

Hypernyms of e1     :  container, crockery

Hypernyms of e2     :  dirtiness
```

This shows all the hypernyms of E1 and E2 along with the shortest dependency parse and the POS tags. Our Result Analysis is presented below in a tabular form.

Model	**Parameters**	**Sub-Parameters**	**Time Taken(in seconds)**	**Accuracy**
Decision Tree	criterion="entropy", splitter="best", max_depth=30, class_weight='balanced'		2.18	36.54% 37.55%
Decision Tree	criterion="gini", splitter="best", max_depth=30, class_weight='balanced'		2.56	38.16% 38.96%
Bagging Classifier	base_estimator=DecisionTreeClassifier(),	criterion="entropy", splitter="best",	6.68	38.09%

	n_estimators=5	max_depth=30, class_weight='balanced'		39.01%
Bagging Classifier	base_estimator=DecisionTreeClassifier(), n_estimators=5	n_estimators=5 criterion="gini", splitter="best", max_depth=30, class_weight='balanced'	7.79	39.56% 40.09%
AdaBoost Classifier	base_estimator=DecisionTreeClassifier(), n_estimators=10	criterion="gini", splitter="best", max_depth=50, class_weight='balanced'	43.89	With direction: 39.86%, Just Relation: 40.74%

As presented, we tried out 5 different models with the AdaBoost Classifier giving us the best accuracy of 40.74% of classifying just the relation and 39.86% of classifying the relation with the direction sense.

d) Problems Encountered and Workarounds:

1) Decision Tree with overly-complex trees and overfitting problem was encountered, with the training data. We switched the splitting criterion as "gini" as opposed to "entropy" in the learner. Also, the max_depth was adjusted to avoid overly complex trees.

2) Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

3) SVM classifier was initially thought as the classification model, however, it's bloated nature made it really hard to train and test on our current systems. We hence, switched to a simpler Decision Tree Classifier for this.

4) We also considered the POS tags of the entities as features, but however, most of them were just Nouns and ProperNouns, inaccurately classifying most of the relations as 'Other'. We therefore dropped this feature.

e) Pending Issues:

1) While the large training size available in SemEval-2010 Task 8 enables achieving high scores using only word based features, richer linguistic and background knowledge resources still provide additional aid in identifying semantic relations.

2) Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

3) There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

f) Improvements and Future Work:

1) Use of Google NGram data can determine the top 1,000 words that occur most often in the context of each nominal, which can further enhance the predictions.

2) Count vectorizer here just grabs the overall occurrences of the POS and dependency parsing window which is not that adequate to be considered as a primary feature as per PCA. Different methodology has to be considered for processing the textual data to extract multiple features.

- **References:**

1) Rink, Bryan, and Sanda Harabagiu. "Utd: Classifying semantic relations by combining lexical and semantic resources." In Proceedings of the 5th International Workshop on Semantic Evaluation, pp. 256-259. 2010.

2) *https://spacy.io/usage/models*

3) CNN (Zeng et al., 2014). “Relation Classification via Convolutional Deep Neural Network”

4) Esuli, Andrea, Diego Marcheggiani, and Fabrizio Sebastiani. "ISTI@ SemEval-2 Task 8: Boosting-Based Multiway Relation Classification." In Proceedings of the 5th International Workshop on Semantic Evaluation, pp. 218-221. 2010.