CS-3730 HW: Deception Detection and Parsing

Ravneet Singh

October 15, 2020

1 Introduction

A generally difficult task for humans is to detect whether someone is telling the truth and this happens almost everyday in the Court system. One tool that is occasionlly used is a polygraph test for measuring whether someone is being intentionally deceptive. Unfortunately using such a tool is difficult to set up and requires physical touch to get working, and even then it may not be fully accurate [1]. Therefore other methods may be necessary to attempt to identify deception, especially in a court setting. A deception dataset is provided by Pérez-Rosas et al. that covers the use of deception in the court settings [2]. This dataset was constructed by taking recordings of public testimonies of witnesses and the defendants, classifying what they said as truthful or deceptive based on the final result of the court case.

2 Program Description and Usage

We ran a feature set comparison experiment including the features that paper suggested as well as some features extracted from RST and PDTB parse trees. We use premade parsers by for both RST and PDTB in order to parse the statements made in the dataset [3]–[5]. A visual inspection of the resulting parse trees reveal that some of the sentences couldn't be parsed effectively. Furthermore, all of the successful PDTB parses started with an explicit tags, though that either be in part due to the old parser being used and how detecting implicit relations can be more dificult. As for the RST parse trees most of the productions included one or more explanation tags which makes sense given the setting of people giving testimonies to explain their view point.

2.1 Files & Folders

The folders and files are organized in the following manner. As a note I am including the downloaded parsers because it is not the most straight forward to set up the old RST parser with some dependencies missing.

- Data - the folder that contains the Court deception dataset
- Python - the code which also contains the parsers
  - main.py - this is the file that contains all of the code. It will copy all of the data into the appropriate locations, parse it if not already parsed, and then extract the appropriate featues

  - pdtbParse.sh and rst_parse.sh - these are bash files that are run automatically by main to actually run the respective discourse parsers
  - gCRF_dist - this is the rst parser made by Feng and Hirst [4], [5]. This uses an off the shelf CRF classifier through crfsuite[1] [6] to to split the text into EDUs and then using an SVM and Logistic Regression to classify them.
  - pdtb-parser - this folder contains the java based end-to-end discourse parser by Lin et al. [3]. This package produces discourse relations in a pipe delimited format using various classifiers and to find each component necessary.

- Results - This folder will contain any results that the main file produces
- requirements.txt - the file that contains all of the packages for the main program to work.

2.2 Usage

This program was written in python 3.7 and the rst parser needs python 2.7 and the pdtb parser needs java version 1.7+. To set up the rst parser head into the gCRF_dist folder and use both python2.7 and the supplied requirements.txt file to create a virtual environment called 'venv'[2]. To set up pdtb parser just make sure java is installed and the bash script should handle it from there. To get the main file working just install the pacakges in the requirements.txt at the root of the folders. Once all of those packages are installed the main program can be run with the following steps:

- cd Python
- python main.py

And the results will be placed in the Results folder.

3 Feature Description & Experiment

For this project we ran a feature set experiment using the features described by Pérez-Rosas et al. of unigrams, bigrams, and behavioral features [2]. We also include features extracted from RST and PDTB parses of the dataset.

3.1 Inspection of Parse Features

From the RST parse trees we extract the productions from each tree and excluded the productions that involve leaves. We then count up the number of times each production was used and use those counts as a feature. This results in 155 features across the dataset. Table 1 highlights the top 5 features of the dataset split by class. It is clear that much of the rst productions are elaboration in some form and that makes sense as witnesses or defendents put on trial are typically explaining what they are saying.

[1] The included crfsuite program works on Linux and you might need to download the correct CRFsuite binarys for your operating system and repace the crfsuite-stdin file with it.

[2] The name is important for the bash script to work

Truth	Count
elaboration[n][s] → elaboration[n][s] elaboration[n][s]	19.0
same-unit[n][n] → elaboration[n][s] elaboration[n][s]	6.0
joint[n][n] → attribution[s][n] joint[n][n]	6.0
elaboration[n][s] → elaboration[n][s] joint[n][n]	6.0
elaboration[n][s] → attribution[s][n] elaboration[n][s]	6.0
Deception	Count
elaboration[n][s] → elaboration[n][s] elaboration[n][s]	12.0
same-unit[n][n] → elaboration[n][s] elaboration[n][s]	5.0
elaboration[n][s] → elaboration[n][s] attribution[s][n]	5.0
same-unit[n][n] → elaboration[n][s] attribution[s][n]	4.0
elaboration[n][s] → elaboration[n][s] explanation[n][s]	4.0

Table 1: This table shows off the top 5 features between the documents true and deception labels. The count column is the number of times that feature occured in the respective class.

We do something similiar with the pdtb parses extracting the top level tag for each relation extracted and the 1st semantic class inside it. This results in only 4 features and the distribution of these features across classes is essentially the same. The only difference is the how temporal relations are slightly more common in true statements than in deception. This is possible in scenarios where the deceiver is vague about the exact timing of events whereas those speaking truthfully have a better sense of when events occur.

Truth	Count	Deception	Count
explicit=expansion	84.0	explicit=expansion	88.0
explicit=temporal	47.0	explicit=contingency	49.0
explicit=contingency	43.0	explicit=temporal	30.0
explicit=comparison	13.0	explicit=comparison	27.0

Table 2: This table highlights the counts for each of the 4 features similiar to the rst table.

Due to the sheer variety of features present it is possible that rst features are more useful than pdtb features, but it is possible there is more information present in a pdtb parse that were not extracted in this pass.

3.2 Experiment Details

Due to the size of the dataset (121 observations with 60/61 truth/deception split) it is fairly easy to run test all combinations of features extracted with multiple models as well. So we tested all combinations of the five feature sets described above (uniX, biX, behavFeatures, rstX, pdtbX) against three base models of Logistic Regression, Support Vector Machines, and Decision Trees. We also do Leave one out cross validation, the same as the author. Table 3 highlights the top five models and features set combinations that worked the best.

Feature Set	Model	Accuracy	Precision	Recall	F1
biX_behavFeatures	Logistic Regression	0.826	0.380	0.380	0.380
biX_behavFeatures_rstX	Logistic Regression	0.809	0.371	0.371	0.371
biX_behavFeatures	SVM	0.801	0.380	0.380	0.380
biX_behavFeatures_rstX	SVM	0.793	0.380	0.380	0.380
behavFeatures	Logistic Regression	0.785	0.405	0.405	0.405
biX_behavFeatures_pdtbX	Logistic Regression	0.785	0.380	0.380	0.380

Table 3: This table shows the top 6 feature model combinations over all of the ones tested.

From the results it is clear that the behavioral with just bigrams works the best with sklearn's default Logistic Regression. It seems that the features extracted from coherence parsers do generally help but only in a limited sense. This also highlights that bigram and behavioral features are always necessary to produce decent results very similar to what the authors reported in their paper.

4 Conclusion

We run a feature experiment testing various models and features highlighting that logistic regression with bigrams and behavioral features works the best. Furthermore, even though the parse features can help in certain combinations, it doesn't always help models identify deception.

Something to note is it is possible the differences we see in the parse features across classes may only be due to the limited data size. Many of the features in rst were just variations of the same productions. Furthermore, the pdtb features have essentially the same distribution in features. It is possible that either higher quality features need to be extracted or more data needs to be collected to get a clearer picture of the efficacy of discourse parsing on this deception data.

As for my comments on this task, I am generally fine with running deception detection especially in a court setting where the polygraph test is inaccurate in many cases. My main issue is in the construction of this dataset. The biggest assumption the authors make is that the result of the court case is the complete arbiter of truth. This scenario only works if the persone on the stand is not a good liar, there was on behind the scene deal to erroneously exonerate or pass a guilty sentence, or the data the courts had was not conclusive enough to confirm the charge. This leads to the issue of the data being almost too superficial and realistically this should not be used in real scenarios of deception detection until there is a dataset that is more robust to the issue.

References

[1] W. G. Iacono, "Accuracy of polygraph techniques: Problems using confessions to determine ground truth," *Physiology & Behavior*, vol. 95, no. 1, pp. 24–26, 2008, ISSN: 0031-9384. DOI: 10.1016/j.physbeh.2008.06.001.

[2] V. Pérez-Rosas, M. Abouelenien, R. Mihalcea, and M. Burzo, "Deception Detection using Real-life Trial Data," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, ser. ICMI '15, New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 59–66, ISBN: 978-1-4503-3912-4. DOI: 10.1145/2818346.2820758.

[3] Z. Lin, H. T. Ng, and M.-Y. Kan, "A PDTB-styled end-to-end discourse parser," en, *Natural Language Engineering*, vol. 20, no. 2, pp. 151–184, Apr. 2014, ISSN: 1351-3249, 1469-8110. DOI: 10.1017/S1351324912000307.

[4] V. W. Feng and G. Hirst, "Two-pass Discourse Segmentation with Pairing and Global Features," *ArXiv:1407.8215 [cs]*, Jul. 2014.

[5] ——, "A Linear-Time Bottom-Up Discourse Parser with Constraints and Post-Editing," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: LONG PAPERS)*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 511–521. DOI: 10.3115/v1/P14-1048.

[6] N. Okazaki, *Crfsuite: A fast implementation of conditional random fields (crfs)*, 2007.