Detecting Betrayal by analyzing the features present in the communication

Marija Lazaroska
EPFL - 287052
marija.lazaroska@epfl.ch

Mert Soydinç
EPFL - 321131
mert.soydinc@epfl.ch

Utku Görkem Ertürk
EPFL - 321977
utku.erturk@epfl.ch

***Abstract*—We trained several classifiers in order to predict the ending of a Diplomacy game by examining the linguistic cues of betrayal. Two research questions were proposed, if the game ends with betrayal and when we can detect the betrayal if it ends in betrayal. Logistic Regression as well as Random Forest and Gradient Boosting Classifier, which we refer to as basic models, were used to obtain predictions. Later on a Linear Neural Network and Recurrent Neural Network was implemented in order to investigate potential improvement. Different methods were applied for the feature extraction in order to determine the best set of input features. At the end we obtained promising results for both of the questions for the basic models. The Recurrent Neural Network didn't work as expected, as the trained model gave poor predictions. We conclude that the Linear Neural Network achieves the best results in terms of classification accuracy on the test set, namely 0.69 and has a Matthews correlation coefficient of 0.42 for the first question and classification accuracy 0.86 and has a Matthews correlation coefficient of 0.31 for the second question.**

I. Introduction

In this paper, we try different methods to improve the result obtained by our assigned paper [4]. The main result we try to improve from the paper is the classification of games that end with one of the players betraying the other one. The paper uses a dataset of messages exchanged by the players during a game of Diplomacy as its training and test data. The assigned paper uses Logistic Regression as its method of classification.

We attempt to solve this classification problem using five different models: Logistic Regression, a Random Forest, a Gradient Boosting Classifier, a fully connected linear Neural Net, and a Recurrent Neural Network. We lay out the architectures for these models and provide experimental results for each of them. We implement everything on PyTorch[5], Numpy[1], sklearn[7], matplotlib[2] and imblearn[3].

II. Diplomacy, The Board Game

QDiplomacy is a World War 1 themed board game where players representing the global powers of the time form alliances with and wage war against each other. Players try to control strategic points on the game's map at the expense of other players in order to win the game. Players either support other players or attack them using the troops at their disposal. While it is possible to win the game alone, it is far more likely that that alliances emerge victorious. This incentives the players to form alliance. However, there is always the probability of betrayal. Forming of alliances and betrayals are all communicated through verbal(or written) communication between players which makes the game a perfect opportunity to analyse communication patterns between players. Diplomacy uses years as its time unit. Each years has 2 seasons(Spring and Fall) where players can communicate among themselves and decide what they are going to do for the next season. This communication can be private between some players or it can be public to all players. For the purpose of this project, we define the last act of friendship of a game as the last season a player uses their troops to support the other player in that game.

III. Dataset description

We are using the Diplomacy Betrayal Dataset[4] in our work. This dataset contains information about the communication that took place between players who are playing the online version of the board game diplomacy. The dataset has information about 500 games and for each game it has information about 2 players playing it. Each game has varying number of seasons and each season contains linguistic information about the exchanged messages in the season. Contents of these messages are not included and only the linguistic cues about the messages are stored. These cues can be seen in Table 1. Each season also contains metadata about the messages such as the number of words request and sentences used in that season. The dataset has a field for each game where it specifies whether a betrayal occurred in the game or not. We must only consider seasons before the potential victim discovers the upcoming betrayal to ensure that we are looking at unbiased data. In order to ensure this, We only take into account seasons up to the last act of friendship for all of our analysis in this project.

IV. Research Questions

We try to answer the following two questions in our project.

A. Using linguistic cues, is it possible to detect betrayal?

For this question, we try to predict if the game ended with a betrayal given the complete data for a single game(except whether the betrayal happens or not).

B. What is the probability of betrayal happening next season given the messages received so far?

For this question, we try to predict if the next season will result in a betrayal given the communication data of previous seasons for both players.

V. Methods

For the following methods, we use 5-fold cross validation on our dataset during the training. The optimal hyper-parameters for the Gradient Booster Regressor and Random Forest are obtained with 5-fold internal cross-validation using Grid search.

A. Feature Extraction

Our dataset contains 14 fields which we use as features. The complete list of these fields can be seen at Table I. We use slightly different processes for feature extraction for each of our models depending on their performance. Three common processes for all our methods are: we take the mean of fields that have more than a single value per season such as sentiment value of exchanged messages, we only use the data from the season if both players talk to each other and we create two sets of inputs, one excluding

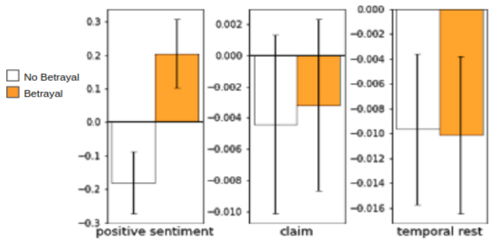


Fig. 1: Imbalance plots of the Claim and Temporal rest feature in comparison with the Positive Sentiment feature imbalance

the Claim and Temporal Rest features and the other including them. The reason why we choose to exclude these two features is because their average imbalance scores are similar and their error bars are overlapping(Figure 1) in comparison with the other features. The remaining processes are described under their respective headline.

Features	
Politeness	Premise
Number of requests	Claim
Number of words	Comparison
Number of sentences	Expansion
Positive sentiment	Contingency
Negative sentiment	Temporal
Neutral Sentiment	Planning

TABLE I: The features we used for training our models

1) ***Question A**:* For this question, We take the mean of all seasons in the game for all of the linguistics features which gives us 14 fields for each entry.As before mentioned we form 2 sets of features: one containing all 14 features and one excluding the Claim and Temporal Rest fields, which leaves us with 12 fields. While extracting the features, for both of the sets, we divide the features by players:victim and betrayer. The truth label for the sample is the betrayal field corresponding to that game in our dataset. We have as many samples as the number of games in the dataset after excluding those that had seasons where none of the players exchanged messages or one did and the other didn't. At the end we were left with 365 samples.

*a) **Neural Network**:* For the Neural Networks, we concatenate the absolute values of the imbalance of features between players(victim and betrayer) with the mean of the features for each entry between players. We use the two sets of inputs, at the end for the NN we have one set with 24 fields and the other with 28 fields per input sample.

*b) **Others**:* For training all the other models, we proceed by using the imbalance of the features between the victim and betrayer in the game. Different from the NN setting, here the 2 sets have 14 and 12 fields.

2) ***Question B**:* For this question, we only take betrayal games which have more than 4 seasons before the game's last act of friendship. As in Question A, we proceed using the 2 sets of features, but we also create two more groups of inputs that differ in how we label the seasons. For the first group we use the same approach described in the paper by labeling the seasons by the distance from the betrayal. Therefore we have 4 classes(seasons that are 4 seasons and more are distanced from the betrayal are grouped in one class by taking the mean of their feature values). For the second group we have only 2 labels: 0 if the season precedes a season that doesn't end with betrayal; 1 if the season precedes a season that ends with betrayal. The way we form the inputs can be seen in Figure 2.

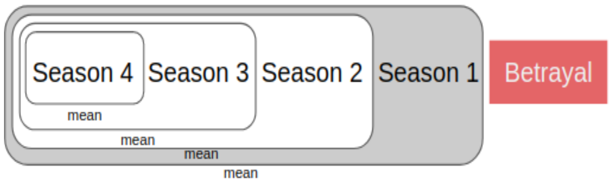


Fig. 2: An example entry with 4 seasons, showing the method of labeling the inputs for Question B for the second group of inputs(white boxes: label 0, grey box: label 1)

Due to the nature of this question, there can only be 1 season labeled the last season before the betrayal per game but a game has many seasons. This creates an unbalanced dataset where regular seasons outclass betrayal seasons by roughly 10 to 1(see Figure 3). This causes many issues while training the models. We try to overcome this by using SMOTE[8] and oversampling the smaller class labels to have a 50:50 split. This step takes place after we split the data into training and test sets during the cross-validation. Only the training set is oversampled.

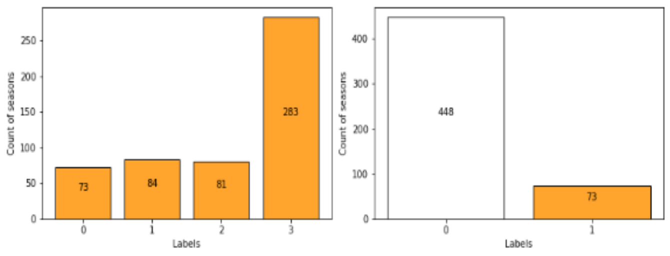


Fig. 3: Distribution of the imbalanced labels for Question B for the first group of inputs(*left*) and the second group of inputs(*right*)

*a) **Neural Network**:* For this model, we only use the second group of inputs(using the running mean) with the two sets:including and excluding Claim and Temporal Rest fields. We again use concatenation of the absolute values of the difference of features between players with the mean of the features between players. In the end, we have as many samples as there are seasons in the game that fit the criteria mentioned above.

*b) **Others**:* For the other models, we use both groups of inputs and both sets, therefore we run 4 training/testing tests. However, differently from the NN we just use the imbalance between players.

B. Logistic Regression

Same as our assigned paper[4], we are using Logistic Regression as our most basic classification method. We use sklearn[7] library to implement the logistic regression.

C. Random Forest & Gradient Boosting Classifier

We implement a Random Forest and Gradient Boosting classifiers using the sklearn[7] library. We use grid search hypertuning using different parameters in order to find the best classifier for our task.

D. Linear Fully Connected Neural Network

We are using a fully connected Neural Network for both of our questions. Since the feature size and number of classes are the same for both the questions, we use the same architecture. This model has 64 neurons in its first hidden layer and 32 neurons in its second hidden layer. All neurons use relu activation functions except the last layer which uses a sigmoid activation function. It contains 2 neurons at its output layer. We use Cross-entropy loss with several learning rates. We also use the Pytorch[6] libraries Adam optimizer to apply the gradient descent.

E. Recurrent Neural Network

We are using a Recurrent Neural Network for our second question in order capture the relations between successive seasons. The idea of a recurrent network is that we have a hidden layer which feeds into the next step in our model. This creates a memory for the model as it remembers the previous inputs using the hidden layer outputs. Other than the presence of this hidden layer, it is pretty much a linear fully connected network with a 64 hidden layer neurons and 2 output layer neurons with Softmax activation. In our case, outputs of the hidden layer is initialized to all zeros at the beginning. The input to the model is concatenated with the hidden layer values from the previous timestep(all zeros if it is the first time step) in order to calculate hidden layer output and the output of the current timestep. Output of each timestep is taken into account when calculation the loss and its gradient.

VI. Results

The best results we obtained after training the models for Question A are presented in Table II. The results from the models using all of the features are better than excluding the Claim and Temporal features. From the basic models the Logistic Regression and Random Forest give similar results and higher(accuracy 0.62 and Matthews corr. coeff. $\approx$ 0.22) in comparison with the Gradient Boosting Classifier(accuracy 0.56 and Matthews corr. coeff. 0.11). Contrary to the paper's results where they provide the mean of the results, we observe the best results from the crossvalidation. However, the results from our NN are more promising then the results from the basic models. The result we presented in Table II is the best result compared to the results from all the epochs, instead of choosing the result form the last epoch. The reason we chose this method is because of the overfitting we observed during the training(Figure 4 (a)).

The best results from training the models for Question B are shown in Table III. Models using the group of inputs with 2 classes as labels perform better in the case when Claim and Temporal Rest were included, but in the case of excluding them the models perform better using the 4 classes as labels. By observing the results from the basic models, we can say that the Gradient Boosting Classifier gives the best results when using inputs including the Claim and Temporal rest features(F1 score 0.30 and Matthews corr. coeff. 0.17), but in the case of excluding the, the Random Forest performs the best(F1 score 0.31 and Matthews corr. coeff. 0.22). In the case of Linear NN, the results show that the NN model outperforms the basic models. However, the results are again higher when using all the features, which means excluding the Claim and Temporal rest features doesn't improve the result for the Linear NN. As for Question A, we also chose the result from the epoch that was the highest because of the overfitting in the training step(Figure 4 (b)).

The RNN didn't perform as expected. The classification accuracy on the test set is very low in comparison with the results from the basic models and the Linear NN. The model is predicting only zeros which is why we get a Matthews correlation coefficient of 0. We believe this to be caused by the unbalanced nature of the dataset and small number of training samples we have.

VII. Conclusions

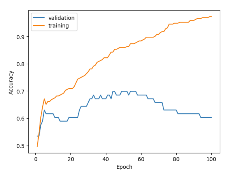
We experimented with different models in order to improve the classification accuracy of our assigned paper with respect to our research questions. All of the models except the RNN performed adequately for both of the questions. We expected RNN to perform better as it has the added benefit of remembering the previous timesteps but the nature of the dataset caused some problems during training which rendered the RNN useless. In the end, our best model turned out to be the Linear Neural Network. This is not surprising since neural networks are known to perform well in these types of classification tasks. Even tough the Linear Neural Network outperformed the other models, its results are not high enough to assure that we can predict if a game ends with betrayal or detect when the betrayal will happen. Even if we tried other NN models we would still lack data and training a NN with only 500 games is a hard task to do. If we have enough data the neural network itself can learn the feature extraction part and would give better results. We considered using another data set but we couldn't find one that would address the same research questions.

Logistic regression	Accuracy	0.62	0.60
	Matthews corr. coeff.	0.23	0.21
Random Forest	Accuracy	0.62	0.57
	Matthews corr. coeff.	0.22	0.15
Gradient Boosting Classifier	Accuracy	0.56	0.60
	Matthews corr. coeff.	0.11	0.22
Linear Neural Network	Accuracy	0.69	0.68
	Matthews corr. coeff.	0.42	0.34

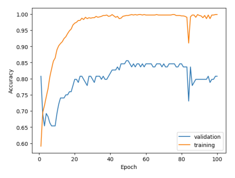
TABLE II: Results for the models referring to Question A *(in gray the results for inputs without the claim and temporal rest feature)*

Logistic regression	4 classes	F1 score	0.23	0.26
		Matthews corr. coeff.	0.001	0.16
	2 classes	F1 score	0.26	0.26
		Matthews corr. coeff.	0.06	0.06
Random Forest	4 classes	F1 score	0.15	0.31
		Matthews corr. coeff.	0.04	0.22
	2 classes	F1 score	0.20	0.20
		Matthews corr. coeff.	0.08	0.06
Gradient Boosting Classifier	4 classes	F1 score	0.17	0.19
		Matthews corr. coeff.	0.07	0.10
	2 classes	F1 score	0.30	0.13
		Matthews corr. coeff.	0.17	-0.01
Linear Neural Network	2 classes	Accuracy	0.86	0.81
		Matthews corr. coeff.	0.31	0.06

TABLE III: Results for the models referring to Question B *(in gray the results for inputs without the claim and temporal rest feature) - The F1 scores presented for the case of 4 classes is the F1 score of class 0(label for a season that precedes a betrayal season)*



(a) Question A - Epochs vs Accuracy for NN using all 14 features



(b) Question B - Epochs vs Accuracy for NN using all 14 features

Fig. 4: Plots of validation and training loss vs. epochs. Notice that first net overfit the data after 43th epoch and the second one is after 48th.

References

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[3] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.html.

[4] Vlad Niculae et al. *Linguistic Harbingers of Betrayal: A Case Study on an Online Strategy Game*. 2015. arXiv: 1506.04744 [cs.CL].

[5] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[6] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] "SMOTE: Synthetic Minority Over-sampling Technique". eng. In: *The Journal of artificial intelligence research* 16 (2002), pp. 321–357. ISSN: 1076-9757.