POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Semantic Web and Social Networks Final Project

KG EMBEDDINGS: EXPLORING CLEORA [1]: A SIMPLE, STRONG AND SCALABLE GRAPH EMBEDDING SCHEME FOR GITHUB USERS CLASSIFICATION

Group "the":
Filip Firkowski, 151946
Jerzy Pawlik, 151964
Eliza Czaplicka, 151963
Weronika Zawadzka, 151943

Contents

1 Introduction **1**

2 Dataset **2**

2.1 Class distribution 2

2.2 Node Features 3

3 Related work **4**

3.1 Embeddings 4

3.1.1 Cleora embeddings 4

3.2 Dataset 4

4 Experimental setup **5**

4.1 Goal 5

4.2 Metrics used 5

4.3 Input to Cleora 5

4.4 Including node features 6

4.5 Experiment settings 6

5 Results **8**

5.1 Best model in terms of micro-F1 8

5.2 Best model in terms of macro-F1 8

6 Conclusion **10**

Bibliography **11**

Chapter 1

Introduction

In this task, we take on the role of a recruiter aiming to identify top talent for machine learning engineer positions. To perform our search, we look at GitHub, a platform where most developers showcase their work and collaborate on projects. Our goal is to focus on active and engaged users, specifically those with at least 10 starred repositories, as these individuals will likely be better candidates. Since the dataset we are using, GitHub Social Network [2], was created in June 2019, these developers likely now have even more experience and expertise. An additional challenge in this task is the imbalance within the dataset. Machine learning developers are the minority so it adds to the difficulty of the task.
For the task of node embeddings, we will explore the performance of a simple, strong, and scalable graph embedding technique known as Cleora [1]. Cleora has proven to be effective at capturing the structural and relational properties of graphs while being computationally efficient, which makes it an ideal choice for large-scale social graphs like the one used for this work. To evaluate the effectiveness of these embeddings, we will perform binary node classification, where the goal is to classify users as either machine learning developers or not (web developers), based on their network structure and activity patterns on GitHub.

Implementation

The implementation behind this task is publicly available through `https://github.com/wkzawadzka/the-cleora`.

Chapter 2

Dataset

We will be using the ”GitHub Social Network Dataset” [2], which consists of a large-scale social network of developers collected via GitHub’s public API in June 2019. In this dataset:

- Nodes represent developers who have starred at least 10 repositories, and edges indicate **mutual follower** relationships.
- Vertex features were extracted based on information like **location**, **repositories starred**, **employer**, and **email**.

The summary of the structure of the dataset can be seen in Table 2.1:

	GitHub Social Network
Nodes	37,700
Edges	289,003
Density	0.001
Transitivity	0.013
Node labels	Yes, binary-labeled

Table 2.1: Statisics of the dataset

2.1 Class distribution

Dataset is imbalanced. Machine learning engineers are the minority class. In total, there are 27961 Web devs and 9739 ML Engs; global ratio is therefore 0.348. The class distribution can be seen in Figure 2.1.

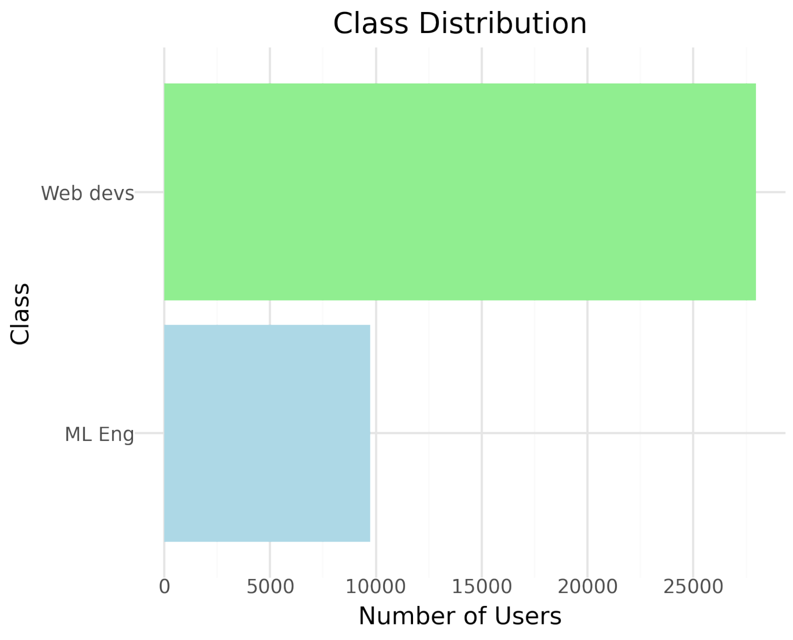

FIGURE 2.1: Class distribution

2.2 Node Features

The node features in this dataset are provided in a '.json' file, where each node (ranging from 0 to 37,699) is associated with a vector of numerical values. These numerical features capture user's geographical location, the number of repositories they have starred, their employer, and their email address, all of which have been numerically encoded. On average, there are 18.31 features per node in the dataset (with minimum of 8 to a maximum of 42).

When looking at specific subsets of the data:

- For Web Developers, the average number of features per node is 18.13 (with the minimum being 8 and the maximum being 42).
- For ML Engineers, the average number of features per node is slightly higher, at 18.82 (with the minimum being 9 and the maximum being 38).

These can potentially help us further on when included in the binary classification model.

Chapter 3

Related work

3.1 Embeddings

For each node, an embedding is created: a vector of numbers that encodes certain properties of that node. Traditionally, to capture the neighborhood structure of a node, we would use an adjacency matrix, a large V×V matrix where each cell indicates a connection between two nodes. Embeddings offer a more efficient alternative by representing nodes with fixed-size vectors. By doing so structural information is preserved in a compact form.

3.1.1 Cleora embeddings

In this work, we are using Cleora [1] embeddings. The main goal for Cleora was to be fast and scalable, as the social network datasets are huge in size. It was implemented in Rust. Secondly, the algorithm aimed to be conceptually simple. Cleora can be used for link prediction or node classification; we will focus on the latter. On a conceptual level, the process in Cleora is simple: given a graph $G = (V, E)$ with a set of nodes V and a set of edges E, and a random walk transition matrix M, one column of the embedding matrix (e.g., the first dimension of the vector for all V) is taken. This column is then weighted by the matrix M and normalized (to prevent the values from diminishing or exploding). This process gives the embedding value for node 1 in the first dimension. It is repeated for all nodes to obtain T_1. Then, $T_2, T_3, \ldots$ are simply the same steps applied to the previous matrices, one after another.

3.2 Dataset

The dataset GitHub Social Network Dataset was taken from a publication [2], where authors applied pooled (AE) and multi-scale (MUSAE) attributed node embedding methods and performed classification using those vectors. Classification performance was evaluated by training l2-regularized logistic/softmax regression to predict a (test) attribute given a node embedding [2]. The average micro F1 score achieved on this dataset, which is equivalent to accuracy, is 0.864 ± 0.001. We will investigate whether using Cleora, a simpler algorithm, can achieve similar results.

Chapter 4

Experimental setup

4.1 Goal

The goal of machine learning experiment is to evaluate the quality of Cleora embeddings, by training models on them to accurately classify GitHub users into ML a web developer groups. During the experiments we compare different models and parameter settings to find the most effective combination. The obtained metrics can be compared to other embedding schemes presented in the referenced paper [2].

4.2 Metrics used

Since the publication [2] where our dataset was introduced used micro f1-score, we also wanted to measure it to see how comparable are our results. However, the authors compared many different datasets there, including multi-class ones. In our case, we are dealing with binary classification. In sklearn, micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise and would be the same for all metrics [3]. Therefore we do measure accuracy and will compare it to results achieved in mentioned paper. [todo equation]

Additionally, since the dataset is imbalanced, we also will use macro metrics: precision, recall, f1-score [todo equations]. These are treating all classes equally and so are insensitive to the imbalance.

4.3 Input to Cleora

The input for the Cleora model, consists of a plain text file with an edge list that defines the relationships between nodes in the graph. The edge list typically consists of two columns: the source node and the target node, representing each edge in the graph. Each row in the file corresponds to an individual edge, and the nodes are represented by their unique identifiers (in our case integer node id). Once the edge list is loaded, Cleora can use various methods to generate graph embeddings, including expansion modes. Expansion modes enhance the embedding process by extending the graph structure to include higher-order relationships, providing a better representation of node interactions. These modes focus on expanding the local neighborhood of nodes and capture a richer graph features for embedding. To make Cleora use the expansion scheme, the edge list file must be properly formatted and and column modifiers must be specified in the Cleora command. For example in our experiments we tested two expansion modes:

- Clique expansion – in this format, first column of a row is a node id, and in the second column there is a list of all the nodes connected to that node. The command line modifier 'complex::reflexive::node' is used while invoking the cleora command
- Star expansion – in this format, for every central node a cluster id node is introduced to link it with its neighbors. The command line modifier is transient::cluster_id node

4.4 Including node features

The node features were provided in the form of a JSON file. Based on the dataset, features were extracted from attributes such as geographic location, starred repositories, employer information, and email address. The input consisted of numerical embedding strings of varying lengths, with a maximum of 41 columns. Following the application of Cleora embedding, the first 10 numerical features were appended to each node. This number was determined experimentally, as it yielded the optimal results when compared to embeddings of different lengths. The resulting feature arrays were subsequently normalized using a standard scaler to ensure uniformity across features.

4.5 Experiment settings

To achieve optimal results and evaluate the impact of different classification models on performance, experiments were conducted using five distinct models:

- Decision Tree Algorithm
- Stochastic Gradient Descent (SGD) Classifier
- Logistic Regression
- Histogram-Based Gradient Boosting Classifier
- K-Nearest Neighbors (KNN) model.

In addition to employing various models, several variables were considered during the experiments:

- Cleora Expansion Type: Cleora requires the decomposition of hyperedges into edges, as the algorithm operates on the pairwise relationships of node transitions. This can be achieved using either clique expansion or star expansion:
  - Clique Expansion: Each hyperedge is transformed into a subgraph where every pair of nodes is connected by an edge. This method is typically applied to smaller graphs.
  - Star Expansion: An additional node is introduced, linking to all original nodes within a hyperedge. This approach is recommended for larger graphs with extensive hyperedges.
- Number of Cleora Iterations: The number of iterations influences the generalization of the dataset. Based on our experiments, the classification accuracy improves with an increasing number of iterations up to 10, after which it stabilizes. Consequently, we conducted experiments with 3, 5, 6, 8, and 10 iterations.
- Inclusion of Node Features: It was uncertain whether incorporating node features would improve the results. Therefore, experiments were conducted both with and without node features.

- Use of Oversampling and Undersampling Methods: We evaluated the impact of applying these methods to address imbalances in the dataset.

Chapter 5

Results

Considering all the previously mentioned variables and models, experiments were conducted with every possible combination of these variables. This approach enabled us to see the influence of specific configurations on the results.

The findings revealed that the star expansion method outperformed the clique expansion for our dataset. Additionally, 10 iterations of Cleora produced the best results while maintaining a reasonable computation time. The application of oversampling and undersampling methods, on average, yielded a slight improvement in the F1 score.

Interestingly, experiments that excluded node features achieved slightly better results. This outcome suggested that while starred repositories and employers might have contributed to classification performance, features such as location and email address did not provide meaningful insights, potentially overwhelming the more relevant data.

We also observed differences in the performance of classifiers. The Decision Tree Algorithm performed significantly worse than all other models. In contrast, the K-Nearest Neighbors Classifier achieved the highest average accuracy (0.85).

Despite these averages, the differences in performance among variables such as node features, oversampling/undersampling, or expansion type were relatively small, typically ranging between 0.01 and 0.02 in the F1 score.

Interestingly, the single best result was achieved using the Histogram-Based Gradient Boosting Classifier with star expansion and 10 Cleora iterations. In this configuration, node features were included, and SMOTE was not applied. This experiment achieved an accuracy of 0.86 and an F1-score of 0.801.

5.1 Best model in terms of micro-F1

The best model in terms of micro F1, and therefore accuracy, is as mentioned it10_star_feat_woimblearn_6 experiment that is using Histogram-Based Gradient Boosting Classifier. The confusion matrix can be seen in Figure 5.1.

5.2 Best model in terms of macro-F1

The best model in terms of macro F1 was it10_star_wofeat_imblearn_6 experiment which used SGDClassifier and achieved score of 0.805. The confusion matrix can be seen in Figure 5.2.

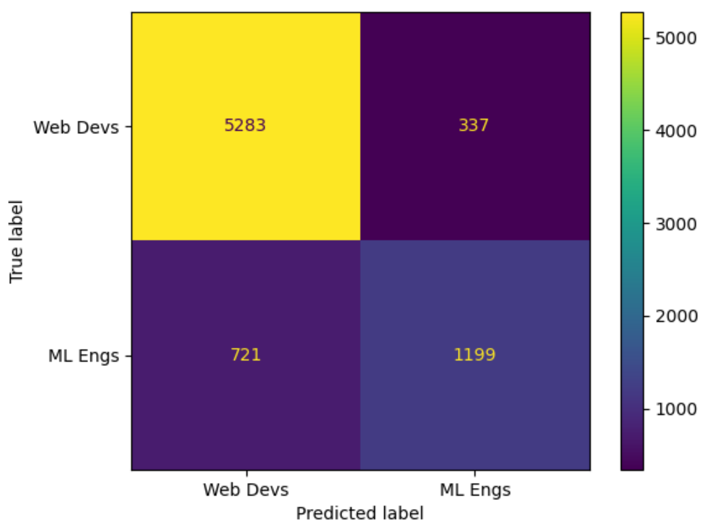

FIGURE 5.1: Confusion matrix for best micro-F1 model

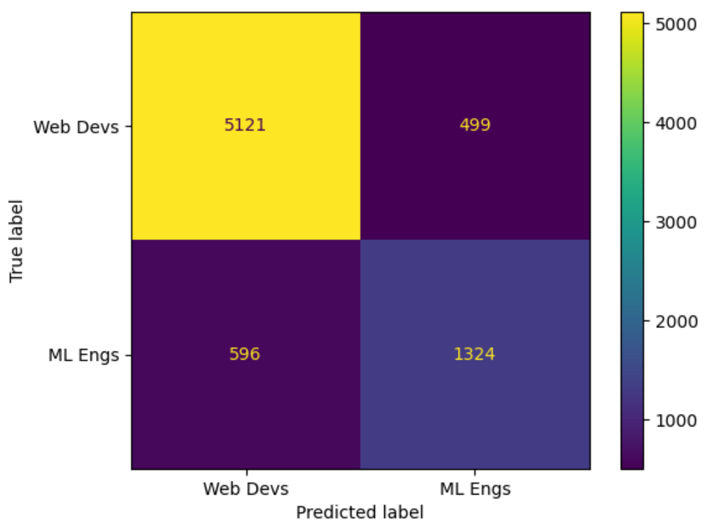

FIGURE 5.2: Confusion matrix for best macro-F1 model

Chapter 6

Conclusion

The study demonstrated that Cleora embeddings are effective for GitHub user classification, achieving competitive results compared to more complex embedding methods. Among the evaluated configurations, star expansion consistently outperformed clique expansion, particularly when paired with 10 Cleora iterations. The best-performing model was the Histogram-Based Gradient Boosting Classifier, which achieved the highest accuracy and F1-score. This highlights its suitability for the task, particularly in handling imbalanced datasets.

Interestingly, excluding node features such as geographic location and email address slightly improved classification results, suggesting that these features might introduce noise rather than enhance performance. While oversampling and undersampling methods, such as SMOTE, provided minor improvements to the F1 score, their overall impact was limited, indicating the robustness of Cleora embeddings even under imbalanced conditions.

Increasing the number of Cleora iterations up to 10 improved performance, with results stabilizing beyond that point. This indicates that 10 iterations strike a balance between computational efficiency and accuracy. Among the classifiers tested, decision trees performed poorly, while K-Nearest Neighbors and Gradient Boosting excelled, reflecting their ability to effectively model the dataset's complexity.

Future steps include extending analysis to larger datasets, incorporating temporal graph aspects, and integrating Cleora embeddings with Graph Neural Networks for richer feature capture. Exploring advanced data augmentation, fine-tuning Cleora hyperparameters, and comparing it with other embedding techniques could further enhance performance. Applying these methods to multi-class or multi-label problems would broaden their applicability.

Overall, the findings suggest that Cleora embeddings, when paired with effective classifiers and optimized configurations, are well-suited for graph-based binary classification tasks. The results underscore the importance of hyperparameter tuning, careful feature selection, and strategies to address data imbalance for achieving optimal performance.

Bibliography

[1] Barbara Rychalska, Piotr Bąbel, Konrad Gołuchowski, Andrzej Michałowski, and Jacek Dąbrowski. Cleora: A simple, strong and scalable graph embedding scheme, 2021.

[2] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.

[3] Scikit learn developers. Classification report. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html`, 2025. Accessed: 2025-01-24.

Appendix

Experiment	**Model**	**precision_macro**	**recall_macro**	**f1_macro**	**precision_weighted**	**recall_weighted**	**f1_weighted**	**accuracy**
it10_star_wofeat_imblearn_6	SGDClassifier1	0.811	0.8	0.805	0.853	0.855	0.854	0.855
it8_star_wofeat_imblearn_6	SGDClassifier1	0.826	0.787	0.803	0.854	0.859	0.855	0.859
it10_star_wofeat_imblearn_6	LogisticRegression	0.798	0.805	0.802	0.85	0.848	0.849	0.848
it10_star_wofeat_imblearn_6	HistGradientBoostingClassifier	0.796	0.807	0.801	0.849	0.846	0.848	0.846
it8_star_feat_woimblearn_6	HistGradientBoostingClassifier	0.829	0.781	0.8	0.854	0.859	0.853	0.859
it8_star_wofeat_imblearn_6	HistGradientBoostingClassifier	0.794	0.804	0.799	0.848	0.845	0.846	0.845
it6_star_wofeat_imblearn_6	SGDClassifier1	0.815	0.787	0.799	0.849	0.854	0.85	0.854
it8_star_feat_imblearn_6	HistGradientBoostingClassifier	0.794	0.804	0.799	0.847	0.844	0.846	0.844
it8_star_feat_imblearn_6	LogisticRegression	0.794	0.802	0.798	0.847	0.844	0.845	0.844
it8_star_wofeat_imblearn_6	LogisticRegression	0.788	0.805	0.796	0.846	0.84	0.843	0.84
it8_star_feat_woimblearn_6	LogisticRegression	0.784	0.807	0.794	0.845	0.837	0.84	0.837
it5_star_wofeat_imblearn_5	SGDClassifier1	0.805	0.785	0.794	0.845	0.849	0.846	0.849
it6_star_wofeat_imblearn_6	HistGradientBoostingClassifier	0.787	0.794	0.79	0.841	0.839	0.84	0.839
it6_star_wofeat_imblearn_6	LogisticRegression	0.778	0.805	0.789	0.843	0.832	0.836	0.832
it8_clique_feat_woimblearn_6	HistGradientBoostingClassifier	0.827	0.765	0.788	0.848	0.853	0.846	0.853
it8_star_feat_woimblearn_6	SGDClassifier1	0.815	0.77	0.788	0.844	0.85	0.845	0.85
it5_star_wofeat_imblearn_5	HistGradientBoostingClassifier	0.784	0.791	0.787	0.839	0.837	0.838	0.837
it6_clique_feat_woimblearn_6	HistGradientBoostingClassifier	0.823	0.766	0.787	0.846	0.852	0.845	0.852
it6_star_feat_imblearn_6	LogisticRegression	0.773	0.805	0.786	0.841	0.827	0.832	0.827
it6_star_feat_imblearn_6	HistGradientBoostingClassifier	0.774	0.803	0.786	0.841	0.829	0.833	0.829
it6_star_feat_imblearn_6	SGDClassifier1	0.777	0.787	0.782	0.835	0.831	0.833	0.831
it8_star_feat_woimblearn_6	KNeighbors	0.817	0.753	0.776	0.84	0.846	0.837	0.846
it5_star_wofeat_imblearn_5	LogisticRegression	0.761	0.799	0.775	0.835	0.815	0.822	0.815
it8_clique_feat_woimblearn_6	LogisticRegression	0.771	0.781	0.775	0.83	0.826	0.828	0.826
it5_star_wofeat_imblearn_5	KNeighbors	0.792	0.755	0.77	0.829	0.836	0.831	0.836
it10_star_wofeat_imblearn_6	KNeighbors	0.818	0.746	0.77	0.838	0.844	0.834	0.844
it8_star_wofeat_imblearn_6	KNeighbors	0.815	0.744	0.769	0.836	0.843	0.833	0.843
it6_star_feat_imblearn_6	KNeighbors	0.813	0.745	0.769	0.836	0.842	0.833	0.842
it6_clique_feat_woimblearn_6	LogisticRegression	0.757	0.776	0.765	0.823	0.815	0.818	0.815
it6_star_wofeat_imblearn_6	KNeighbors	0.816	0.738	0.764	0.835	0.841	0.83	0.841
it5_clique_wofeat_imblearn_5	HistGradientBoostingClassifier	0.763	0.764	0.763	0.82	0.82	0.82	0.82
it5_clique_wofeat_woimblearn_5	HistGradientBoostingClassifier	0.803	0.74	0.762	0.829	0.837	0.828	0.837
it5_clique_wofeat_imblearn_5	SGDClassifier1	0.777	0.751	0.762	0.821	0.828	0.823	0.828
it8_clique_feat_woimblearn_6	KNeighbors	0.815	0.736	0.762	0.834	0.84	0.829	0.84
it6_clique_feat_woimblearn_6	KNeighbors	0.809	0.731	0.756	0.83	0.837	0.825	0.837
it5_clique_wofeat_woimblearn_5	LogisticRegression	0.745	0.77	0.755	0.817	0.804	0.809	0.804
it5_clique_wofeat_imblearn_5	LogisticRegression	0.744	0.77	0.754	0.817	0.803	0.808	0.803
it8_clique_feat_woimblearn_6	SGDClassifier1	0.809	0.72	0.747	0.827	0.833	0.82	0.833
it3_star_wofeat_woimblearn_7	SGDClassifier1	0.746	0.747	0.747	0.808	0.807	0.807	0.807
it5_clique_wofeat_woimblearn_5	KNeighbors	0.761	0.726	0.74	0.806	0.815	0.808	0.815
it5_clique_wofeat_imblearn_5	KNeighbors	0.761	0.726	0.74	0.806	0.815	0.808	0.815
it3_star_wofeat_woimblearn_7	KNeighbors	0.766	0.722	0.738	0.807	0.817	0.809	0.817
it6_clique_feat_woimblearn_6	SGDClassifier1	0.79	0.712	0.736	0.815	0.825	0.811	0.825
it3_star_wofeat_woimblearn_7	HistGradientBoostingClassifier	0.733	0.732	0.733	0.797	0.797	0.797	0.797
it3_clique_wofeat_woimblearn_7	HistGradientBoostingClassifier	0.73	0.734	0.732	0.797	0.795	0.796	0.795
it5_clique_wofeat_woimblearn_5	SGDClassifier1	0.81	0.701	0.73	0.822	0.827	0.81	0.827
it3_star_wofeat_woimblearn_7	LogisticRegression	0.716	0.763	0.727	0.807	0.767	0.779	0.767
it3_clique_wofeat_woimblearn_7	KNeighbors	0.755	0.708	0.725	0.798	0.809	0.8	0.809
it3_clique_wofeat_woimblearn_7	SGDClassifier1	0.72	0.71	0.715	0.784	0.788	0.786	0.788
it10_star_wofeat_imblearn_6	DecisionTree	0.7	0.728	0.71	0.784	0.764	0.771	0.764
it8_star_feat_woimblearn_6	DecisionTree	0.707	0.706	0.706	0.777	0.778	0.777	0.778
it8_star_feat_imblearn_6	DecisionTree	0.69	0.716	0.699	0.776	0.755	0.762	0.755
it8_star_wofeat_imblearn_6	DecisionTree	0.688	0.716	0.698	0.776	0.752	0.761	0.752
it6_clique_feat_woimblearn_6	DecisionTree	0.7	0.695	0.697	0.77	0.773	0.772	0.773
it8_clique_feat_woimblearn_6	DecisionTree	0.696	0.697	0.697	0.77	0.769	0.769	0.769
it3_clique_wofeat_woimblearn_7	LogisticRegression	0.686	0.723	0.695	0.779	0.742	0.754	0.742
it6_star_wofeat_imblearn_6	DecisionTree	0.675	0.701	0.683	0.764	0.74	0.749	0.74
it5_star_wofeat_imblearn_5	DecisionTree	0.67	0.698	0.678	0.762	0.734	0.744	0.734
it5_clique_wofeat_woimblearn_5	DecisionTree	0.669	0.672	0.671	0.75	0.748	0.749	0.748
it6_star_feat_imblearn_6	DecisionTree	0.662	0.69	0.669	0.756	0.724	0.736	0.724
it5_clique_wofeat_imblearn_5	DecisionTree	0.652	0.678	0.659	0.748	0.717	0.728	0.717
it3_clique_wofeat_woimblearn_7	DecisionTree	0.62	0.641	0.625	0.721	0.689	0.701	0.689
it3_star_wofeat_woimblearn_7	DecisionTree	0.616	0.637	0.62	0.718	0.683	0.696	0.683
it8_star_feat_imblearn_6	KNeighbors	0.657	0.579	0.583	0.717	0.753	0.713	0.753
it8_star_feat_imblearn_6	SGDClassifier1	0.515	0.51	0.32	0.641	0.329	0.281	0.329

Table .1: All experiments