A Comparative study of different Machine Learning techniques on Kaggle Twitter Social Influencers Data Set

Arora, Pragya*, Ghai, Piyush† and Ramkrishnan, Navnith§
Department of Computer Science & Engineering, The Ohio State University
Columbus, OH 43202
Email: *arora.170@osu.edu, †ghai.8@osu.edu, ‡ramkrishnan.1@osu.edu

***Abstract*—AbstractThe advent of social networks has made it significant to analyze social media data. These graph networks have insights on different trends which are interesting to study. Certain nodes act as sinks whereas others as sources emancipating various other nodes. Understanding these patterns will help in various applications related to National Security and even in experiments like Network A/B Testing where we new features are released to a treatment group restricting the control group. In our project we intended to experiment and analyze on one such pattern where we try to identify the influential node among a given pair. To understand such a behavior, we used a Twitter data set which had 11 features specific to each user. We applied machine learning techniques to understanding the underlying features and classify the task at hand.**

I. Introduction

The growing importance of Social Networks has given an impetus to understand in detail about its implications because these networks influence millions of people. Contrary to print media or television, social media has a greater reach to audiences across the world and information exchange has never seen such a revolution. The need of the hour is to have the ability to understand the nuances of these networks because with more connectivity it becomes crucial to understand the behavior of different nodes.

Some nodes/users are more influential than the others with their behavior impacting a larger set of nodes in the graph. Literature in psychology and cognitive science suggest that ones life is greatly impacted by immediate surroundings and their influence. As social networks have become a part of our lives, it is worthwhile to identify such influential users. One application of such a study is to predict the general mood of the crowd during elections by analyzing the social network handles of the candidates.

In this project, we solve a Kaggle[1] challenge, in which influencers on Twitter are predicted from a set of features derived from user activity on the network, such as follower count, retweets received, etc. Rather than identifying influencers in the larger network, the goal is to identify given a pair of users, A and B, who is more influential. This is a binary-classification problem. To find an optimal prediction method, we applied pre-processing techniques and feature transformations. We used four different machine learning algorithms: Logistic Regression, Support Vector Machines (SVM), Neural Networks and Gradient Boosting, to model the data. We also tried varying some of the decision boundary parameters for the SVM by using different kernels and neural network classifiers and adjusted the activation functions on the hidden layer. Our approach includes a few methods that we did not see in previous attempts at this problem, specifically a logarithmic feature transformation, the application of feature selection and gradient boosting. Outside of gradient boosting, the model configurations which produced the best results used linear-like decision boundaries (Logistic Regression).

II. DATASET

This section expands on the data set which we used to try the different models on.

A. Original Dataset

The data set which we used came from a Kaggle contest. The data set had the following files :

- train.csv
- test.csv
- sample_predicitions.csv

The train.csv file contains data points where each data point is a pair wise features for two users along with a binary value which reflects which user is more influential. If the value is 0 the first user is more influential and vice-verse. Each user has 11 unique features which are extracted from their twitter profile and recorded in numeric form. The 11 features extracted includes:

- Followers
- Following
- Listed
- Mentions received
- Re-tweets received
- Mentions sent
- Re-tweets sent
- Posts
- Network Feature 1
- Network Feature 2

- Network Feature 2

Similarly we have features for two users in each data item in the testing file but the testing file has no class label. There are 5500 training samples and 5953 testing samples in our data set. Given a test sample, our job is to predict which individual in this test sample is more influential.

B. Kaggle Competition

The Kaggle competition is hosted at : [2]. The evaluation for this competition is based on area under the ROC curve. [3].

C. Data Preprocessing

Since the data is a numerical data, we use some pre-processing techniques on it. The dataset consists of 22 attributes which is essentially 11 attributes for each user in contention. Since we have to compare who is more influential in the network, we subtract related attributes for each user, to reduce the dimensionality to 11. The following is the transformation applied :

$$a_{\mathrm{j}} = x_{\mathrm{j}} - x_{\mathrm{j+11}}, j = 1, 2, 3...., 11.$$

Since the attributes are of a different scale, we also apply a logarithmic transformation on the data. The following is the transformation applied :

```
def transform_features(x):
    return np.log(1 + x)
```

Listing 1: Log Transformation

All of these transformations are applied to all the models used. In addition to these, we also apply Z-Score normalization on the dataset for using Gausian Naïve Bayes model.

III. SYSTEM MODELS

A. Baseline

For baseline, we tried a dumb baseline model, where we classified an influencer based on the number of followers, i.e. if X has more followers than Y, then X is more influential. Using this dumb baseline, we get an accuracy of about **70.2%**. This result shows that the number of followers is a strong indicator of the influencers. However, 70.2% is not a very satisfying result and hence we use it only as a benchmark. We know further experiment with more models and a bit more pre-processing. This also suggested that the data can be linearly separable if we choose the right set of features.

B. Logistic Regression

For Logistic Regression, we implemented our own model using two different loss functions : *Logistic Loss* & *Exponential Loss*. We also used *scikit-learn*'s Logistic Loss for comparison to our own implementation of Logistic Regression. We used Stochastic Gradient Descent for implementing the LR algorithm.

C. SVM

We applied the technique of SVM's with a linear kernel as well as a radial basis kernel function for SVM. For SVM we use scikit learn packages implementation. [4]

D. Gaussian Naïve Bayes

Logistic regression and SVM are discriminative learning algorithms. Gaussian Naïve Bayes, is a generative model. The distribution of the original data is not Gaussian, so we tried Z-Score normalization in order to make it as a Gaussian distribution. In Z-Score, we do the following for every value :

$$z = \frac{x - \mu}{\sigma}$$

This converts the data into 0 mean and deviation 1. We then used Gaussian Naïve Bayes from *sklearn* package. [5]

E. Neural Network

We wanted to explore and see how the dataset behaves in a non-linear environment. The idea was to compare the traditional techniques alongside state of the art machine learning models which have gained a lot of importance today. Hence we implemented a basic Neural Network model in Python to see how it behaves in this classification task. Our model was customizable, we could add or remove any number of hidden layers with any number of units in each of them. To our surprise the model worked well but didn't outperform the traditional model. The intuition behind this is that the data set and the features we have are linearly separable.

F. Boosting

We also used scikit-learn's boosting method : specifically, XgBoost [6]. Boosting is an ensemble classifier which combines several weak classifiers to form a hypothesis. Boosting also has advantages over other classifiers in the fact that it is more robust to overfitting on the training data.

IV. TUNING THE MODELS

A. Feature Selection

We did a correlation plot of all the 11 features with the class label. This is represented in Figure IV.1. From the figure we see that there are some features which will not be useful. We were thus motivated to further prune features using a feature selction algorithm.

We used forward selection algorithm to zero in on the best features in order to improve the test accuracy. We used Logistic Regression model in order to select the best accuracies for feature selection. The best features are as follows :

- Follower Count
- Listed_Count
- Retweets Received
- Network Feature 1
- Mentions_Received
- Network_Feature2

We note that *network_feature 2* was not showing up as important in the correlation plot, but it's addition to the best features set led to an improvement in accuracy.

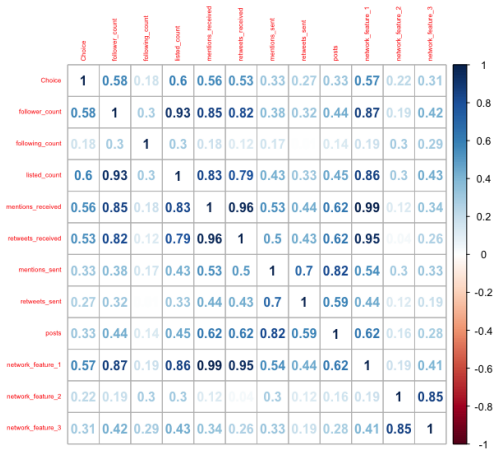

Figure IV.1: Scatter plot for co-relation among all the features

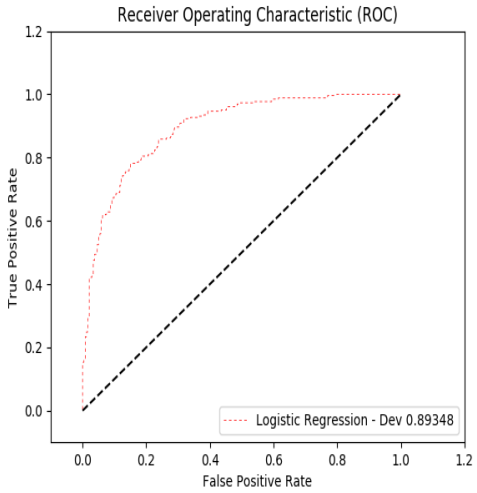

Figure V.1: AUC for Logistic Regression

B. *k- fold Cross Validation*

We used k-fold cross validation with k=20, where the data is partitioned into k subsets, with k-1 for training and the remaining for test. The results are averaged across all the iterations for uniformity.

V. Results & Discussions

A. *Evaluation Criteria*

The metric that we use to quantify our results was *Area Under the ROC Curve*. ROC is a standard evaluation metric for binary classification problems. We used probabalistic outcomes from the models in order to plot the ROC Curves.
For comparing two models, or results from the same model, we compare their AUC values. A ROC Curve with a larger area signifies a better classification model on the given data.
To improve our evaluations, we applied k-fold cross validation technique. We try this with a k value of 20. In k-fold cross validation, the model is split into k subsets and is run for k iterations, where k-1 subsets are used to train. The AUC results are averaged across all the iterations.

B. *Logistic Regression (Our Implementation)*

For our implementation of Logistic Regression, we tried different loss functions - *Logistic Loss*, *Exponential Loss*. The hyperparameters tuned were : λ (L2 Regularization) & η (Learning Rate). The results are summarized in Table VI.1. The results in this table are calculated at $\lambda = 0.1 \& \eta = 0.001$.

Table V.1: AUC for Logistic Regression

Data	Loss Function	AUC
Train	Logistic Loss	0.850868
Dev set	Logistic Loss	0.8838471
Train	Exponential Loss	0.8572271
Dev set	Exponential Loss	0.88924958

C. *Logistic Regression (From scikit-learn)*

We also tried using Logistic Regression from *scikit-learn* package. We tried tuning the *C* parameter, which is inverse of regularization. The best value of C was found to be 100. The following table lists some results. We also evaluate the results with and without feature selection.

Table V.2: AUC for Logistic Regression, C = 100

Data	Feature Selection	AUC
Train	Yes	0.8574375
Dev set	Yes	0.89348
Train	No	0.849165911
Dev set	No	0.88155468

In Table **??** we see that the best AUC for Logistic Regression, where the area under the ROC curve is 0.89348.

So we see, that we get nearly comparable results with both the Logistic Regression implementations, indicating that the data is linearly separable. We thus move onto more linear models.

D. *Support Vector Machine (RBF)*

We tried different regularization parameters. The only hyperparameter tuned was : C (Slack Penalty). The results are summarized in Table V.3. The results in this table are calculated at $C = 0.3$.

Table V.3: AUC for SVM (rbf)

Data	AUC
Train	0.9288767
Dev set	0.8558008

E. Support Vector Machine (Linear)

We tried different regularization parameters. The only hyperparameter tuned was : C (Slack Penalty). The results are summarized in Table V.4. The results in this table are calculated at $C = 0.3$.

Table V.4: AUC for SVM (linear)

Data	AUC
Train	0.8596409
Dev set	0.8929928

So we see, that we get nearly different results with different SVM kernel implementations. This is because the data is linearly separable and hence the linear kernel performs so well.

F. Gaussian Naïve Bayes

With Gaussian Naïve Bayes we normalize the data using Z-score normalization, as mentioned in the Pre-Processing. The AUC is presented in Table V.5.

Table V.5: AUC for Gaussian Naïve Bayes

Data	AUC
Train	0.85311513
Dev Set	0.88468074

G. Neural Networks

After trying different linear models, we did some experiments on non-linear models. We trained and tested the data set on Neural Networks. Instead of using a python package, we implemented the code for NN. We tuned the *hidden layer* and *learning rate* parameter, and tried different loss functions - non-linear ones like : *Sigmoid Loss*, *tanh Loss* & linear loss function : *Identity activation*. The data for results of NN is provided in Table V.6.

Table V.6: AUC for Neural Networks

Data	AUC
Train	0.8599
Dev set	0.8927

Neural Networks did not perform as well as expected. This is because of the kind of dataset we have at hand. The dataset is linearly separable hence the traditional methods outperformed the more complex non-linear models. This shows the significance of dataset analysis prior to model selection. The most promising models may behave differently depending on the features of the dataset.

H. XGBoost Classifier

In this model we used XGBoost classifier to train our model. We tried tuning various parameters *subsample*, η, *colsample_bytree*, *max_depth* and *min_child_weight*. The best parameters were found to be η : 0.001, *subsample*: 0.3, *colsample_bytree*: 0.5, *max_depth*: 3, *min_child_weight*: 9. The following table lists some results. The number of boosting rounds used was 6000.

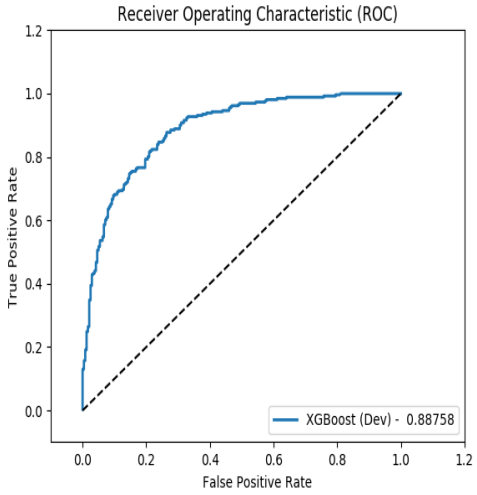

Figure V.2: AUC for XGBoost

Table V.7: AUC for XGBoost

Data	AUC
Train	0.86786
Dev set	0.88758

In Table V.7 we represent the AUC for XGBoost, where the area under the ROC curve is 0.88758.

VI. Conclusions

Our experiments have shown that the models which rely on linear decision boundaries provide the best results on the given dataset. Among the transforms that we used, logarithimic transform proved to be the most effective because it removed the disparity between the various attributes. From the above experiments we can conclude that the AUC for all our models on the dev set varies between 0.87 - 0.89. The following table provides a summary of AUC scores on Test Dataset for our submissions made on Kaggle website. We present the results of the best values of the models described above that we submitted. All the results presented here were the ones in which we used Feature selection, as well as did Z-Score & Log normalization of the data.

From the table we see that XgBoost performs best since it is an emsemble of several weak classifiers. Followed by XgBoost is Logistic regression. This shows that the data was indeed linearly separable. We also learned about doing an analysis of the dataset, as through our analysis we figured out how the dataset was linearly separable and hence we were able to apply the traditional models instead of the complex models. Future works on this problem could involve application of PCA

Table VI.1: AUC scores for Test Dataset (As per Kaggle Evaluation)

Model	AUC
Logistic Regression	0.86065
XgBoost	0.86168
Gaussian Naïve Bayes	0.82009
Neural Nets	0.84182
SVM	0.83786

to further select combinations of best attributes in order to drive better results.

REFERENCES

[1] A website for data science competitions. https://www.kaggle.com/

[2] Influencers in Social Networks : https://www.kaggle.com/c/predict-who-is-more-influential-in-a-social-network

[3] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[4] Sci-kit learn Support Vector Machines : http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[5] Sci-kit learn Gaussian Naïve Bayes : http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[6] XGBoost Library : https://pypi.python.org/pypi/xgboost/

[7] Our repo is hosted at : https://github.com/piyushghai/Social-Influencers.