Predict Influencers in the Social Network

Ruishan Liu, Yang Zhao and Liuyu Zhou
Email: rliu2, yzhao12, lyzhou@stanford.edu
Department of Electrical Engineering, Stanford University

***Abstract*—Given two persons and their social network features, our job is to predict which one is more influential. In our project, we collect training samples from Kaggle based on human judgement. We use several different models to make predictions, such as Logistic Regression, SVM, Naive Bayes and Neural Network. We also use some auxiliary techniques like cross validation, feature selection and data preprocessing. In the results section, we compare the performances of different models and provide analysis and suggestions for future works. We implement different learning models using Matlab.**

I. Introduction

In recent years, social network plays an increasingly significant role in our daily lives. We share our experiences and opinions with our friends on Facebook, Twitter, Instagram and so on. When you're browsing your friends' posts, you may find that some of them are more influential than others and we call them influencers. According to the research in Sociology, influencers have a great impact on other people's lives because people always tend to follow the opinions of these influencers in social networks. Therefore, it is important for us to figure out which persons are influencers and how they shape public opinions. In our project, the goal is to find influencers in a specific social network—Twitter.

II. Dataset

A. Original Dataset

We use the dataset from Kaggle provided by PeerIndex, consisting of a standard, pair-wise preference learning task [1]. Each datapoint describes two individuals, A and B. For each person, there are 11 pre-computed, non-negative numeric features based on Twitter activity provided, which include:

1. # followers	2. # followings	3. #listed
4. # mentions_received	5. # retweets_received	6. # mentions_sent
7. # retweets_sent	8. # posts	9. #network_feature_1
10. # network_feature_2	11. # network_feature_3	

Since we have two persons in each data sample, we have 22 features in total. What's more, there is a binary label representing a human judgement about which of the two individuals is more influential in each training sample. Label 1 means A is more influential than B. Label 0 means B is more influential than A. There are 3000 training samples and 2500 testing samples in our dataset. Given a test sample, our job is to predict which individual in this test sample is more influential.

B. Data Preprocessing

Before applying different models on our data, we may want to preprocess it first. For linear models, the hypothesis function has the following form:

$$h_\theta(x) = g(\theta^T x), \theta, x \in \mathbb{R}^{22} \tag{1}$$

which results in a linear decision boundary, i.e. when $\theta^T x \geq b$, we predict 1; otherwise, we predict 0. Note that for a training example, if we change the order A and B (that is, exchange the last 11 features of the training example with its first 11 features), the label should also be reversed (1 becomes 0, and 0 becomes 1). Thus the coefficients of the first 11 features must be opposite numbers of the coefficients of the last 11 features, i.e.

$$\theta_j = -\theta_{j+11}, j = 1, 2, ..., 11 \tag{2}$$

where θ_j is the jth entry of θ. So there are only 11 independent parameters in θ, and thus we can use only 11 features to represent an example. We choose z as our new representation of training example as follows:

$$z_j = x_j - x_{j+11}, j = 1, 2, ..., 11 \quad (3)$$

where z_j and x_j are the jth attribute of z and x respectively. Note that this preprocessing method will only be used in linear models, i.e. Logistic Regression and SVM. Besides this method, we also use other preprocessing methods such K-means algorithm, which will be discussed in their own models.

III. System Models

A. Follower Count Benchmark

First, we tried a quite straightforward and trivial method based only on the number of followers, i.e. if A has more followers than B, then A is more influential than B. Using this method, the test accuracy is about 70.2%. This result shows that the number of followers is a strong indicator of the influencers. However, 70.2% is not good enough for our prediction and this result will be mainly considered as a benchmark. After adding more features and using more general models, we hope to get a better prediction on the test samples.

B. Logistic Regression

First, we preprocess the original dataset using the previous method. After preprocessing the original dataset, different attributes have different ranges which vary a lot. Thus the first thing to do is to handle the data to make it more uniform and easy to deal with. So in order to achieve this, we have a normalization on the dataset. For each feature, we do the following normalization

$$\bar{z}_j^{(i)} = \frac{z_j^{(i)}}{\sqrt{\sum_{i=1}^m z_j^{(i)2}}}, z = 1, 2, ..., n \quad (4)$$

where m is the number of training examples and $n = 11$ is the number of features.

Because our job is to predict who is more influential given two persons, the number of followers, as we know, plays a significant rule in the prediction. A person with more followers than the other is more likely to be judged influential by users. So we will multiply the normalization of the first feature, i.e. number of followers, by a positive constant factor which is greater than 1 to make it more influential than other features on the prediction.

For this logistic regression problem, we have two choices— gradient ascent and Newton's method to achieve the parameter θ. Since the number of features $n = 11$ is not very large, so it will not take much time to compute the inverse of a $n \times n$ matrix. Thus Newton's method should converge faster in this problem and we choose Newton's method as our implementation. The update rule of the Newton's method is:

$$\theta := \theta - H^{-1} \nabla_\theta l(\theta) \quad (5)$$

where $H \in \mathbb{R}^{n \times n}$ is the Hessian matrix and $H_{jk} = -\sum_{i=1}^m h_\theta(\bar{z}^{(i)})(1 - h_\theta(\bar{z}^{(i)}))\bar{z}_j^{(i)} \bar{z}_k^{(i)}$, $\nabla_\theta l(\theta) = \bar{Z}(\vec{y} - h_\theta(\bar{Z}))$.

Furthermore, we will add cross validation and feature selection to this model, which will be discussed in details in section IV. We write our own code to implement the Newton's method.

C. SVM

First, we also use the preprocessing method mentioned in II-B. We implement SVM through libsvm, a library offered online at http://www.csie.ntu.edu.tw/∼cjlin/libsvm/, with l_2 regularization and linear kernel function [2] [3]. Using SVM model, the problem formulation becomes:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^m \xi_i^2 \\ &s.t. \; y^{(i)}(w^T \bar{z}^{(i)} + b) \geq 1 - \xi_i, i = 1, 2, ..., m \\ &\qquad \xi_i \geq 0, i = 1, 2, ..., m \end{aligned} \quad (6)$$

The box constraint C in (6) is customizable and we let $C = 1$ as its default value.

Moreover, we add cross validation and feature selection to this model, which refers to section IV.

D. Naive Bayes

After using discriminative learning algorithms, we may want to try some generative learning algorithms. The distribution of the original data is probably not Gaussian distribution, so Gaussian discriminant analysis can hardly work. Therefore, we choose the Naive Bayes model. In this case, we use the multinomial Naive Bayes model, which is more general.

Before we apply Naive Bayes algorithm, we need to discretize our dataset first. For this purpose, we choose K-means algorithm to discretize each attribute into several clusters and each cluster corresponds to a class. Applying clustering algorithm like K-means can help us distinguish users from different levels. For instance, a film star may have a million followers while a normal user can only have hundreds of followers and in this case, their difference can't be ignored and we need to put them into different classes.

In the original dataset, some attributes have very large ranges. Thus, we consider using logarithm function on the original dataset before applying K-means. We will compare the performance using logarithm function to not using it in section V.

Once we have discretized the data, we can apply multinomial Naive Bayes model to it. Assume the jth attribute has c_j classes $\{1, 2, ..., c_j\}$, we give the parameters achieving maximum likelihood as follows:

$$\phi_y = \frac{\sum_{i=1}^m I\{y^{(i)} = 1\}}{m} \tag{7}$$

$$\phi_{jk|y=1} = \frac{\sum_{i=1}^m I\{x_j^{(i)} = k \cap y^{(i)} = 1\}}{\sum_{i=1}^m I\{y^{(i)} = 1\}}, k = 1, 2, ..., c_j \tag{8}$$

where $j = 1, 2, ..., 22$ corresponding to 22 features. ϕ_y is probability of $y = 1$; $\phi_{jk|y=1}$ is the probability that the jth attribute is class k given $y = 1$; $\phi_{jk|y=0}$ can be derived similarly and thus we don't give its expression here. With the above parameters and a test sample $y^{(i)}$, we predict 1 if and only if:

$$\begin{gathered} p(y^{(i)} = 1|x^{(i)}) \geq p(y^{(i)} = 0|x^{(i)}) \Leftrightarrow \\ p(x^{(i)}|y^{(i)} = 1)p(y^{(i)} = 1) \geq p(x^{(i)}|y^{(i)} = 0)p(y^{(i)} = 0) \\ \Leftrightarrow \phi_y \prod_{j=1}^{22} \phi_{jy_j^{(i)}|y=1} \geq (1 - \phi_y) \prod_{j=1}^{22} \phi_{jy_j^{(i)}|y=0} \end{gathered} \tag{9}$$

The decision boundary in (9) is linear. However, this model is nonlinear because K-means algorithm and logarithm function are nonlinear mappings.

Using the formula in (9), we can predict on the testing set. However, one thing we haven't specified is the numbers of classes of each attribute and changing this parameter, we may get different testing accuracies. This parameter can be regarded as a 22-dimensional vector, i.e. $c = [c_1, c_2, ..., c_{22}]^T$. If we use a deterministic initial state instead of random initialization in the K-means algorithm, the clustering result is also deterministic given the number of clusters. In this case, all parameters are determined by the vector c and we call it discretization parameter. Thus, the testing accuracy is a function of c, say $f(c)$. By changing c, we may achieve different accuracies and thus we can optimize the accuracy. We provide a coordinate ascent based algorithm as follows:

```
• Initialize $c = [2, 2, ..., 2]^T$.
• Repeat until convergence:
      For $j = 1, 2, ..., 22$
          $c_j := \arg\max_{\hat{c}_j} f(c_1, c_2, ..., \hat{c}_j, ..., c_{22})$
          where $\hat{c}_j = c_j - d, ..., c_j, ..., c_j + d$
```

The parameter d denotes the maximum step we can take in one iteration, which is adjustable. This algorithm has high computational complexity and thus we haven't tested and improved it, which could be considered as a choice in future work. In section V, we choose some specific values of c and illustrate the performance of Naive Bayes. We write our own code to implement the K-means and Naive Bayes algorithm.

E. Neural Network

The Naive Bayes model introduced in the previous section is a nonlinear model. However, this results from our nonlinear preprocessing methods, which makes Naive Bayes itself not that "nonlinear". Besides linear models, we still want to try some nonlinear models with high capacity. For this reason, Neural Network might be a good choice.

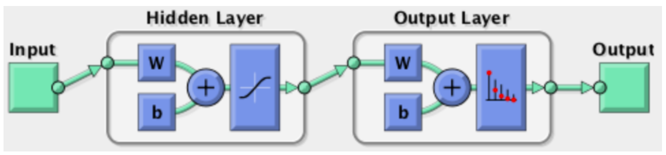

Fig. 1. Neural Network Architecture in Matlab

We use the Matlab Neural Network Pattern Recognition toolbox to implement the Neural Network algorithm, which is designed for classification problems. The network architecture is given in Fig. 1, which is a two-layer feed forward network, with sigmoid hidden neurons and softmax output neuron [4].The number of neurons in the hidden layer is customizable and we will compare the performances using different numbers of hidden neurons in section V.

IV. Model Selection

A. Cross Validation

We use hold-out cross validation and 30% of the training set are considered as validation set. What's more, we randomly split the training set for 100 times (we get 100 hypothesis functions) and pick the one with the smallest cross validation error. We also use k-fold cross validation and we will assign different values to k in section V in order to choose the best one.

B. Feature Selection

In our dataset, it is fairly possible that some features are much more indicative than other features, e.g. the number of followers is a strong indicator. We have 11 features for the linear models as stated above. Although the number of features are not very large, the test error is large and there may be only a subset of these 11 features that are relevant to the result.

In this problem, we use forward search as our feature selection algorithm, which is designed to find the most relevant features. In addition, the maximum number of features is regarded as an input argument in our implementation. Thus, we can change this parameter to optimize the performance and sort features from the most relevant to the least relevant, which will be discussed in section V.

V. Results & Discussions

A. Logistic Regression & SVM

In this section, we apply cross validation and feature selection to the linear models, i.e. Logistic Regression and SVM. The following figures compare the performance among different models.

In Fig. 2, we use different cross validation options on linear models, such as hold-out and k-fold,

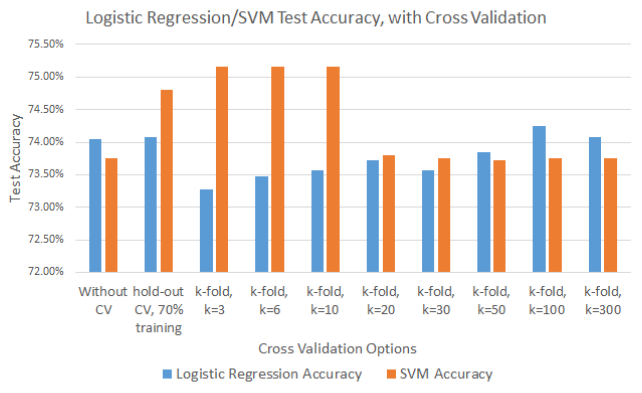

Fig. 2. Test accuracy vs. cross validation options for linear models

and compare their test accuracies. It is shown that cross validation can improve the test accuracy of SVM, while having no strong effect on Logistic Regression. What's more, the performance of SVM is better than Logistic Regression using cross validation.

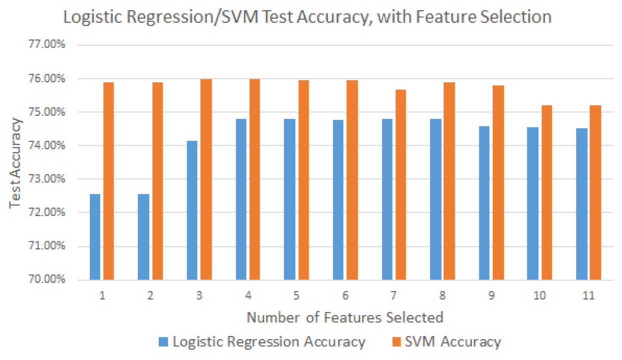

Fig. 3. Test accuracy vs. discretization parameter for linear models

Fig. 3 shows the test accuracies of SVM and Logistic Regression with different numbers of features selected. As is shown above, the performance of SVM is better, compared with logistic regression. The best performance of SVM is achieved when 4 features are selected, which shows that some features are weak indicators. After feature selection, we sort the features from the most relevant to the least relevant as follows:

SVM	9	6	7	10	8	5	3	2	1	4	11
LR	3	6	8	2	9	5	4	7	1	11	10

where the corresponding feature name refers to section II-A. From this table, we can see that although the sort results shown above are quite different for these two methods, their test accuracies are very close. Using linear models, the best accuracy we can achieve is 76.00% (highest test accuracy in Fig. 2 and Fig. 3).

B. Naive Bayes

In this model, we compare the performances of different discretization parameters. Here we assume that all features have the same number of classes (the coordinate ascent based algorithm takes too much time). We also mentioned in section III-D that logarithm function is used to preprocess the data.

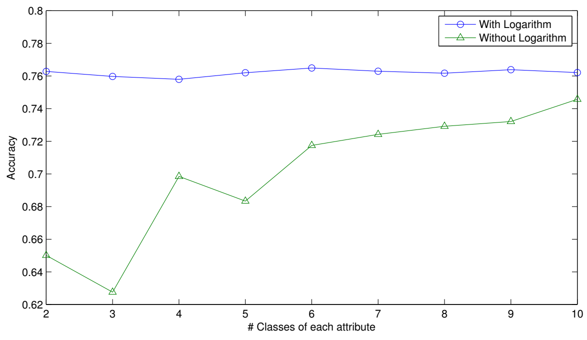

Fig. 4. Test accuracies vs. number of classes of each feature

In Fig. 4, we also compare the performance with logarithm preprocessing and the one without logarithm processing. We can see that logarithm processing helps increase the test accuracy and when the number of classes of each feature increases, their performances approaches. In addition, it turns out that different discretization parameters don't have much impact on test accuracy. In this case, the best test accuracy is 76.48%.

C. Neural Network

In this model, we use 50 neurons in the hidden layer. We also apply hold-out cross validation with 30% as the validation set. The ROC (Receiver Operating Characteristic) curves are shown in Fig. 5. After training the two-layer network, we get the test accuracy 79.04%, which corresponds to the area under the test ROC curve.

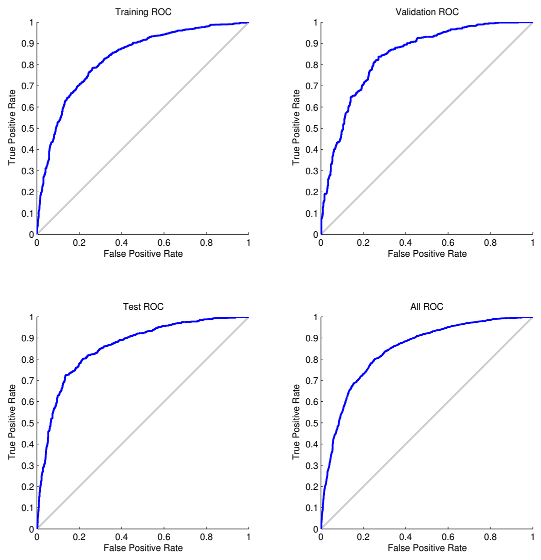

Fig. 5. Receiver operating characteristic

VI. Conclusion & Future Works

From the above results, we can conclude that the best accuracy we can achieve is about 76% using linear models, which is not much better than our benchmark 70.2%. This suggests that the testing examples might not be linearly separable. The test accuracy of Naive Bayes is close to linear models. However, if we can apply the coordinate ascent based algorithm, we may achieve much better performance, which is a good choice for future works. Furthermore, the accuracy of nonlinear models such as Neural Network is better than linear models, although it's not as good as we expected. Moreover, there might be data corruptions since sometimes human judgement can be highly biased. In order to achieve better performance, we can either try more nonlinear models or use decision trees to improve it.

References

[1] J. Furnkranz and E. Hullermeier. Preference Learning: A Tutorial Introducton, *DS 2011*, Espoo, Finland, Oct 2011.

[2] Hsu C W, Chang C C, Lin C J. A practical guide to support vector classification[J]. 2003.

[3] Chang C C, Lin C J. LIBSVM: a library for support vector machines[J]. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011, 2(3): 27.

[4] Swingler K. Applying neural networks: a practical guide[M]. *Morgan Kaufmann*, 1996.