Github Collaborator Recommendation

Albert Jian[a], **Bill Wen**[a], **and Shan Zhong**[a]

[a]University of California, Los Angeles

This manuscript was compiled on June 12, 2020

Recommender systems in the context of social networks recommends a likely acquaintance to a user. In this paper, we use concepts from network theory to develop features for machine learning to produce a recommender system to suggest GitHub followees to users. Using training and testing sets, we are able to demonstrate highly accurate recommender system using machine learning models. From the result, we found that the most important features are personalized PageRank scores, and how similar is the suggested followee to the current user's followees.

Machine Learning | Recommender System | Social Networks

During the recent rise of web services, many companies utilize different version of recommender systems to recommend personalized items to their users, including Google's infamous YouTube algorithm. In social networks, recommender systems recommend the more probable connections to the user.

In this paper we produce a recommendation system for a GitHub social network using machine learning approaches. We will first analyze our results and discuss their relevance, then we will describe the dataset and concepts from network theory and machine learning we used to develop the recommender system.

Results and Discussion

Hyperparameter Selection. Table 1 shows our hyperparameter selection and F_1 accuracy score for a Random Forest model and Gradient Boosting Decision Tree (GBDT) Model. From the F_1 score,

Table 1. Model Hyperparameters and Training and Testing F_1 Score

Model	Hyperparameters(max_depth, n_estimators)	Train F_1	Test F_1
Random Forest	(14, 121)	0.96	0.92
Gradient Boosting Decision Tree	(10, 200)	0.99	0.99

Confusion Matrix and ROC Curve. The confusion matrix for both the testing and training set for the GBDT model is shown in figure 1. While both models are highly accurate for both the training and testing set for determining whether where exists links between two users, GBDT model benefitted from our additional feature engineering and produce a more accurate model. The ROC curve shows that the model is produce very few false positive results, i.e., false connection between two users in the test data network.

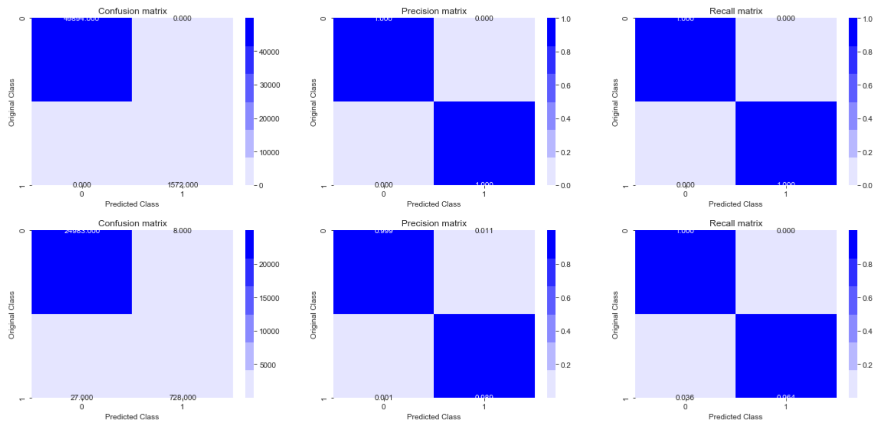
Feature Importance. The area under the receiver-operator characteristic (ROC) curve is abbreviated AUC. In figure 2, we can see the relative importance of features in each model.

PageRank. We find that personalized PageRank score (denoted as page_rank_d for PageRank of destination node) is have the most relevance in our GBDT model on the right, and the PageRank score for the destination node is more important than the PageRank score of the source node. This is expected and consistent with Network Theory, since the most prominent destination user is likely followed by another person.

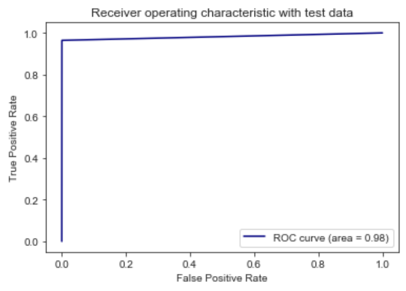
Weight_f2. We find that weight_f2 is ranked high on both models. Weight_f2 is the product of weight of incoming edges for destination node and weight of outgoing edges for source node, and was suggested as a simple permutation of the two weights above in the paper (1).

Significance Statement

Recommender system in social network aims to predict the most probable hidden friendship within the network and recommend it to the users. In this paper, we use network theory feature to develop machine learning model for a recommender system and analyzes the importance of different features within the model.



(a) Confusion Matrix for Training and Testing Set



(b) ROC Curve

Fig. 1. Confusion Matrix and ROC Curve for GBDT Model

Irrelevant Features. There are also surprisingly irrelevant features. Our dataset is undirected, i.e. if a user follows another, the other follows back, but both our random forest model and GBDT model does not use this feature of our network.

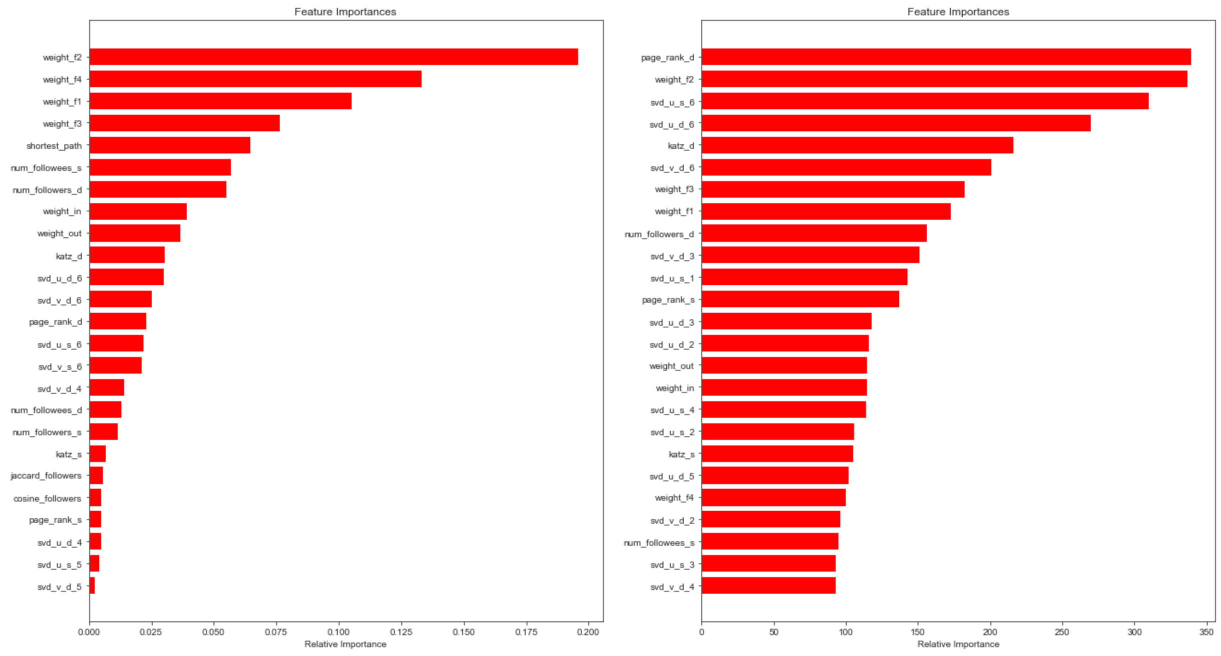


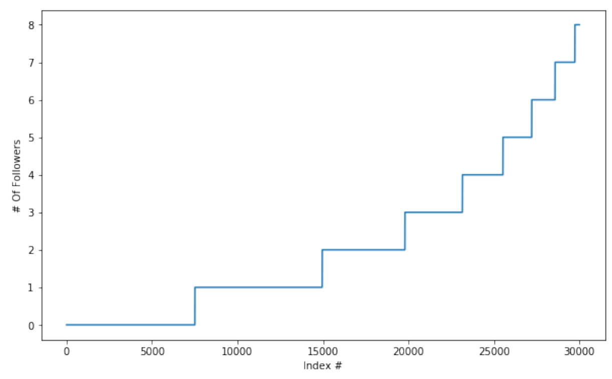
Fig. 2. AUC of Individual Features in the Random Forest Model (left) and GBDT Model (right)

Materials

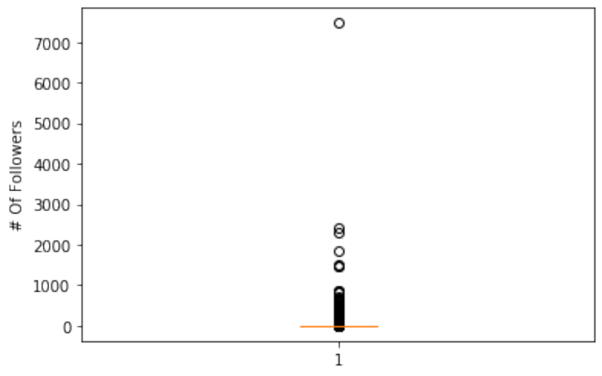
Dataset. The dataset GitHub Social Network is taken from SNAP (2). This dataset contains a large social network of GitHub developers collected from the public API in June 2019. Nodes are developers who have starred at least 10 repositories and edges are mutual follower relationships between them. The dataset consists of 37700 nodes and 289003 edges. While we have a dataset of undirected edges, we created a directed network using the dataset since a recommender system is usually directed, recommending one to another.

Exploratory Data Analysis. The degree distribution of the network is shown in figure 3. The largest number of mutual follows is 9458 and the minimum number is 1. 5045 users in the network have 1 mutual follow and around 25000 people have less than 5 mutual follows. As seen from both of the graphs, the degree distribution is quite left-skewed.

In order to visualize the network, we picked a central node from the training set and we performed a breadth-first walk of the group to a maximum distance of 3 as shown in figure 4. Nodes are sized according to their distance from the center and colored by their personalized PageRank with the central node.



(a) Mutual follows for the least 30000 users



(b) Mutual follows box plot

Fig. 3. Mutual follows per user

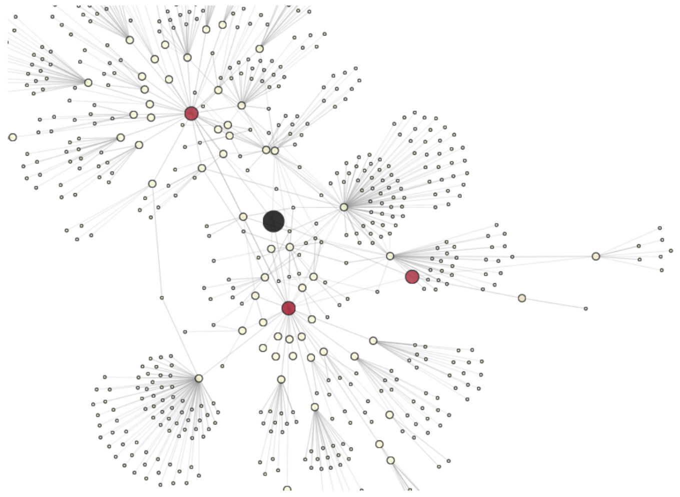


Fig. 4. Example Local Network with Distance of 3

Methods

The methods in this paper is mostly inspired by this research in link prediction in social networks (1). Binary classification method (Random Forest, Support Vector Machine and Gradient Boosting) was adopted to classify good links and bad links. SVD was applied to reduce the dimension of the dataset. Similarity measures such as jaccard distance, cosine distance and preferential attachment were calculated as features. Ranking measures (page rank), centrality measures (Katz centrality) and other metrics such as shortest path and common neighbors were also added for model training. Confusion matrix was plotted at the end to evaluate model performance. We trained for two separate models, one with Random Forest, and one with Gradient Boosting Decision Tree.

Features. Many of our features are mentioned in this paper (3). Some basic features include number of followers by source and destination nodes, number of followees by both nodes. Other more advanced features are described as follows.

Jaccard Distance. In our application, the Jaccard similarity coefficient compares members that the source and destination nodes follows. A higher percentage will indicate that two populations are more similar. Jaccard distance is number of common users divided by number of users that exist in at least one of the two populations

$$J(\mathbf{A},\mathbf{B}) = \frac{|\mathbf{A}\cup\mathbf{B}|}{|\mathbf{A}\cap\mathbf{B}|} = \frac{|\mathbf{A}\cup\mathbf{B}|}{|\mathbf{A}|+|\mathbf{B}|-|\mathbf{A}\cup\mathbf{B}|}$$

Regularized Jaccard similarity. When nodes are represented by the set of their followers in figure 5, we can see from the left figure, while the PageRank and propagation metrics tended to favor nodes close to the central node, the Jaccard similarity feature helps us explore nodes that are further out. We can see that the outlier nodes are much more muted this time around. However, if we look the high-scoring nodes more closely, for instance in the middle figure, we see that they often have only a single connection to the rest of the network: In other words, their high Jaccard similarity is due to the fact that they don't have many connections to begin with. This suggests that some regularization or shrinking is in order. So on the right we have a regularized version of Jaccard similarity, where we downweight nodes with few connections.

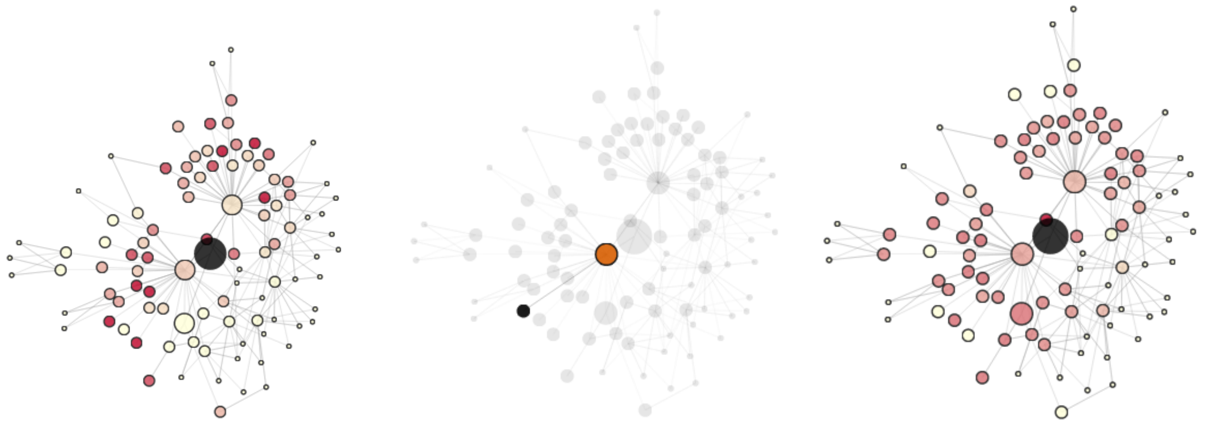


Fig. 5. Jaccard Similarity Feature Engineering Considerations

Cosine Distance. Cosine similarity coefficient is also applicable, where we can compare members the source and destination nodes follows. A higher percentage will indicate that two populations are more similar. Cosine distance is number of common users divided by the total number of users.

$$\cos(\mathbf{A},\mathbf{B}) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n}\mathbf{A}_i\mathbf{B}_i}{\sqrt{\sum_{i=1}^{n}(\mathbf{A}_i)^2}\sqrt{\sum_{i=1}^{n}(\mathbf{B}_i)^2}} \qquad [1]$$

Katz Centrality. Katz similarity coefficient is used here to indicate relative influence of a user within the network taking into account total number of walks between a pair of actors. It takes into account the total number of walks between two users. Here A is the adjacency matrix.

$$Katz(i) = \beta + \alpha\sum_{j} A_{ij}Katz(j)$$

PageRank. PageRank centrality in this context is a holistic measure of how many important users follow this user. We separate into two groups of number when developing the features, PageRank for destination nodes and source nodes. Here α is the damping factor

$$PR(D) = \frac{1-\alpha}{N} + \alpha\sum_{j}\frac{PR(j)}{NumLinks(j)}$$

Shortest Path. Shortest path is the more straightforward feature, since the users will be more likely connected with someone close to who they followed. In our network, the diameter is 16 and average path length is 3.735. In our implementation, we assign -1 to any nonexistent connections.

Follow Back. Follow back is a simple binary feature to indicate whether destination node follows the source node back. In the case of our graph, if a user follows another, the other user will always follow back due to the undirected nature of the graph.

Weakly Connected Components. A digraph is weakly connected if for every pair of distinct vertices u and v there exists an undirected path. Here we consider if the two users belong to the same community, indicating by mutual follows. This feature is proved to be useless, since the dataset is fully connected.

Common Neighbors. It is simply the number of users followed by the source user and following the destination user. Two strangers who have a friend in common are more likely to be introduced than those who don't have any friends in common (3). There is a positive correlation between the number of common neighbors of x and y at time t, and the probability that x and y will collaborate at some time after t.

Weight Feature (1). Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Since the graph is directed, weighted in and weighted out are differently calculated. Here Γ is the function for number of neighbors of a user.

$$\frac{w_i^{in/out} = 1}{\sqrt{1+|\Gamma_{in/out}(v_i)|}}$$

Here are the weight features used

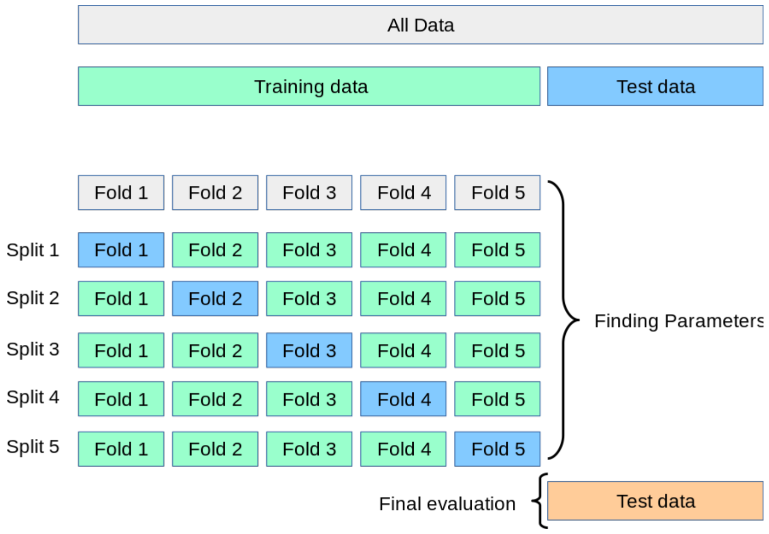
1. weight of incoming edges + weight of outgoing edges
2. weight of incoming edges * weight of outgoing edges
3. 2 * weight of incoming edges + weight of outgoing edges
4. weight of incoming edges + 2weight of outgoing edges + Page

Preferential Attachment (3). One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how rich" our two vertices are by calculating the multiplication between the number of friends ($|\Gamma(x)|$) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.

Machine Learning Models.

A. k-Fold Cross-Validation. Cross-validation is a re-sampling procedure to evaluate machine learning models when we have a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation(4).



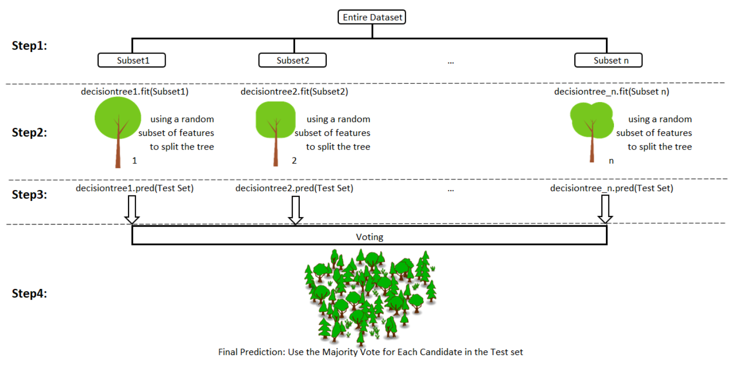
B. Singular Value Decomposition (SVD). The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. Here we derive SVD using the adjacency matrix.

$$A = USV^T$$

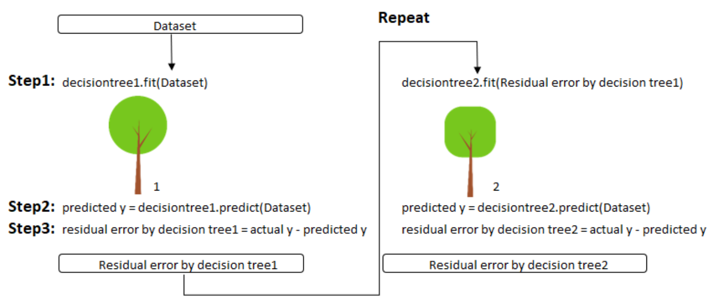
$$\underset{m\times n}{\overset{\text{A}}{\begin{pmatrix} x_{11} & x_{12} & & x_{1n} \\ & & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} = \underset{m\times m}{\overset{\text{U}}{\begin{pmatrix} u_{11} & & u_{m1} \\ & \ddots & \\ u_{1m} & & u_{mm} \end{pmatrix}}} \underset{m\times n}{\overset{\text{S}}{\begin{pmatrix} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \sigma_r & \\ & & & \ddots \\ 0 & & & 0 \end{pmatrix}}} \underset{n\times n}{\overset{\text{V}^{\text{T}}}{\begin{pmatrix} v_{11} & & v_{1n} \\ & \ddots & \\ v_{n1} & & v_{nn} \end{pmatrix}}}$$

where A is the real m x n matrix that we wish to decompose, U is an m x m matrix, S is an m x n diagonal matrix, and V^T is the transpose of an n x n matrix where T is a superscript(5).

C. Random Forest. Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual model(6).


Entire Dataset
Step1:
Subset1
Subset2
...
Subset n
Step2:
decisiontree1.fit(Subset1)
decisiontree2.fit(Subset2)
decisiontree_n.fit(Subset n)
using a random subset of features to split the tree
1
2
n
Step3:
decisiontree1.pred(Test Set)
decisiontree2.pred(Test Set)
...
decisiontree_n.pred(Test Set)
Voting
Step4:
Final Prediction: Use the Majority Vote for Each Candidate in the Test set

D. Gradient Boosting. Gradient Boosting learns from the mistake — residual error directly, rather than update the weights of data points and builds trees one at a time.


Dataset
Repeat
Step1: decisiontree1.fit(Dataset)
decisiontree2.fit(Residual error by decision tree1)
1
2
Step2: predicted y = decisiontree1.predict(Dataset)
Step3: residual error by decision tree1 = actual y - predicted y
predicted y = decisiontree2.predict(Dataset)
residual error by decision tree2 = actual y - predicted y
Residual error by decision tree1
Residual error by decision tree2

Gradient boosting steps(6):

1. Train a decision tree
2. Apply the decision tree just trained to predict
3. Calculate the residual of this decision tree, Save residual errors as the new y
4. Repeat Step 1 (until the number of trees we set to train is reached)
5. Make the final prediction

Supplemental information

Supplemental information including code and latex version of this report can be found at https://github.com/ShanZ3/Github_Collaborator_R

ACKNOWLEDGMENTS. We would like to acknowledge our professor Dr. Heather Zinn-Brooks and our teaching assistant Abigail Hickok for their accomodations during the remote instructions this quarter.

1. W Cukierski, B Hamner, B Yang, Graph-based features for supervised link prediction in *The 2011 International Joint Conference on Neural Networks.* pp. 1237–1244 (2011).
2. B Rozemberczki, C Allen, R Sarkar, Multi-scale attributed node embedding (2019).
3. A Sadraei, Link prediction algorithms (2014).
4. A Sangaiah, *Deep Learning and Parallel Computing Environment for Bioengineering Systems.* (Elsevier Science), (2019).
5. J Brownlee, *Basics of Linear Algebra for Machine Learning: Discover the Mathematical Language of Data in Python.* (Machine Learning Mastery), (2018).
6. L Chen, Basic ensemble learning (random forest, adaboost, gradient boosting)- step by step explained (2019).