Final Project Report

STATS 102 Introduction to Data Science Session 2, 2020 Fall

Predition of the news' popularity_based on the data from Weibo

Group 4

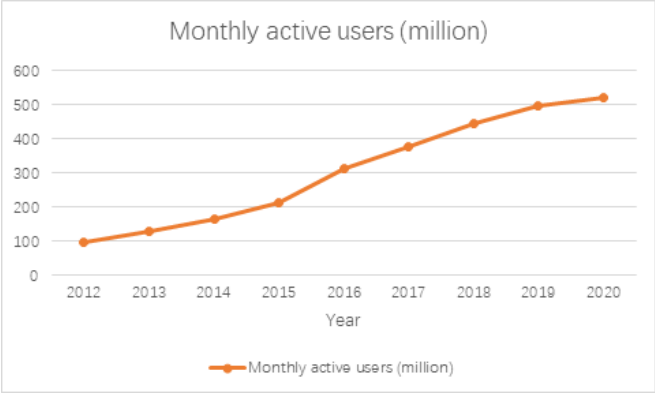
- Sissi Wang(xw187@duke.edu (mailto:xw187@duke.edu))
- Tingyi Li (tl264@duke.edu (mailto:tl264@duke.edu))
- Yijia Xue (yx179@duke.edu (mailto:yx179@duke.edu))

1 Introduction

1.1 Problem description

In the era of new media, social media and we media have shown strong communication power and influence on the spread of information. Compared to traditional channels, social media creates an open and transparent platform for all users to exchange and share information and eliminate the constraints of time and space, which is subtly changing the propagation pattern of new nowadays (Zheng, 2014).

Micro Blog (Weibo in Chinese), namely a blog with concise content, is an emerging service among social media which facilitates internet users to set up real-time information sharing and communication communities. Sina Weibo has become the biggest social media platform in China as a leading figure since its establishment in 2009. According to Sina's official data, with surging growth rate, Sina Weibo has over 523 million monthly active users by October 2020 (Sina, 2020) (see Figure 1).



Consequently, analyzing the publication of news on Sina Weibo has the typical meaning for the investigation in expanding channels of news dissemination, acceleration of news transmission, and the changing pattern of news edition and gathering （Huang, 2017).

1.2 Goals

Motivated by the phenomenon in information era that social media is subtly changing the pattern of news dissemination, we develop this project aiming to:

- Correctly classify a piece of news based on its content;
- Find out and select the properties that stand for the popularity of a microblog as well as appropriate tags that may affect the popularity of a microblog news;
- Find proper algorithm to predict the popularity of a piece of new based on the selected tags;
- Determine the most important tags.

1.3 Achievements

In this project, we reach the achievements of:

- Do words segmentation with manually-selected stop words, and cluster the news into 10 classes through TF-IDF calculation; according to our clustering result, predict the class of a piece of news based on its content;
- Clean the dataset and select 31 tags (fans number, VIP level, tweets number, at number, hash number, has video or not, has image or not, original or not, publish year, publish time, and news type);
- Predict the popularity (repost number, like number and comment number) with Naive Bayes algorithm, K-nearest Neighbor algorithm, Support Vector Machines algorithm, and Random Forest algorithm, and compare the scores of accuracy;
- Understand and adjust the code of the most suitable algorithm (Random Forest), transforming it into the version that only uses basic python libraries, including Numpy and Pandas, which are friendly to new programers;
- Find out the five most important tags that determine the popularity of a weibo news respectively for like number, comment number, and repost number.

2 Background

2.1 Feasibility of popularity prediction

According to the "three rules of epidemics" established by Gladwell (2002), a social epidemic is mainly determined by three principles: Law of the Few, Stickiness Factor, and Power of Context.

“Law of the Few” states that popularity depends on the specific persons involved, which, in the news dissemination case, can be related to number of fans or VIP level. “Stickiness Factor” emphasizes the importance of the message’s content, such as whether the blog has attractive images. “Power of Context” means the surroundings and environment, like posting time, also have a great impact on the popularity of news.

These literatures show feasibility of our model.

2.2 Time efficiency

In the context of new media, public events merge one after another. According to a research, it is proved that the spread of online news is subject to the lifespan of around 10 days based on mathematical models, which also verifies the intuitive sense that ‘the internet has a memory of a week’ (Han, 2018). Although the view number on Sina Weibo is calculated in an accumulative system, the increase of the

data fluctuates little as time washes the public attention away. Therefore, we can ignore the diverse time spans of eac blog from the present in our dataset, since it has little impact on our result of popularity prediction.

3 Design and Implementation

3.1 Data quantification

The data base we used was generated from a website called archive, a site about machine learning repository (https://archive.ics.uci.edu/ml/datasets/microblogPCU) (https://archive.ics.uci.edu/ml/datasets/microblogPCU)), which ensures the credibility of the data. The database covers the information of **24466** Weibo released by **15 users** from **2010 to 2019**.

For each Weibo, we got two kinds of information about it: the information of the Weibo itself and the information of its poster(user) . In details, the information contains: Weibo id, number of comments, number of likes, number of reposts, time it was created at, url of its image (if it has), url of its video (if it has), url of its original Weibo (if it is not original), content, id of the users, number of fans of the user, vip level of the user, number of tweets of the user. To use it forprediction, the data need to be quantified. The method we use is shown below (see Table.1).

Table.1 Quantified tags

	Name	meaning	generated from
0	Fans_num	the number of fans of the user	fans_num
1	Vip-level	the vip level of the user	vip_level
2	Tweets_num	the number of Weibo released by the user	tweets_num
3	At_num	the number of '@' in the content of the Weibo	content
4	hash _num	the number of '#' in the content of the Weibo	content
5	Has_video	whether the Weibo has video (yes:1 no:0)	video_url
6	Has_image	whether the Weibo has image (yes:1 no:0)	image_url
7	Is_origin	whether the Weibo is origin(yes:1 no: 0)	origin_weibo
8	0:6	was released from 0 to 6 O'clock(yes:1 no: 0)	created_at
9	6:9	was released from 6 to 9 O'clock(yes:1 no: 0)	created_at
10	9:12	was released from 9 to 12 O'clock(yes:1 no: 0)	created_at
11	12:14	was released from 12 to 14 O'clock(yes:1 no: 0)	created_at
12	14:18	was released from 14 to 18 O'clock(yes:1 no: 0)	created_at
13	18:24	was released from 18 to 24 O'clock(yes:1 no: 0)	created_at
14	2010	was released in 2010	created_at
15	2011	was released in 2011	created_at
16	2012	was released in 2012	created_at
17	2013	was released in 2013	created_at
18	2014	was released in 2014	created_at
19	2015	was released in 2015	created_at
20	2016	was released in 2016	created_at
21	2017	was released in 2017	created_at
22	2018	was released in 2018	created_at
23	2019	was released in 2019	created_at

Based on these data, we want to:
1) **cluster** the 24466 Weibo we have to 5-10 classes based on their contents
2) build a model to **predict** the popularity of a Weibo (reflected by the number of comment, likes, and repost) with the quantified tags from the raw data shown in Table.1 and the classes clustered from the first step
3) find the **5 most important tags** for the predictio

3.2 Clustering the news into different classes

After trails, we find that the prediction only based on tags in Table.1 is not accurate enough, so we want to cluster the news into different classes based on their contents and use these classes as additional tags to predict the popularity.

3.2.1 Text preprocessing

We use the **jieba package** to do the word segmentation. The document **"stop_words.txt"** contains the words that are automatically filtered out before or after processing natural language data in order to save storage space and improve search efficiency in information retrieval. These stop words are manually input and non-automatically generated, and the generated stop words will form a stop words table 9 (see Table.2) .

Table.2 Stop words list example(translated into English)

i	me	my	myself	we	our	ours	ourselves	you	your	yours	yourself	yourselves	he	him
his	himself	she	her	hers	herself	it	its	itself	they	them	their	theirs	themselves	what
which	who	whom	this	that	these	those	am	is	are	was	were	be	been	being
have	has	had	having	do	does	did	doing	a	an	the	and	but	if	or
because	as	until	while	of	at	by	for	with	about	against	between	into	through	during
before	after	above	below	to	from	up	down	in	out	on	off	over	under	again
further	then	once	here	there	when	where	why	how	all	any	both	each	few	more
most	other	some	such	no	nor	not	only	own	same	so	than	too	very	s
t	can	will	just	don	should	now	d	ll	m	o	re	ve	y	ain
aren	couldn	didn	doesn	hadn	hasn	haven	isn	ma	mightn	mustn	needn	shan	shouldn	wasn
weren	won	wouldn												

After the word segmentation and removing the stop words the data shows as follow (see Figure.2).

```
['观 世界 墨西哥 军方 一座 农场 中 毒贩 枪 战后 发现 72 具弃 尸 一名 自称 这起 惨案 幸存者 男子 称 死者 拒付 赎金 毒贩 枪杀 偷渡客',
 '观 世界 巴西 皮拉波 拉 邦 热 苏斯 小镇 提 亚特 河 污染 有毒 泡沫 浮满 河面 毒 泡沫 泛滥 导致 鱼类 人类 植物 建筑物 严重威胁 污染 沿河 居民 含有 洗涤剂 生活 清洁 用水 排放 提 亚特 河 工业废水 排放 sinaurl h457pD',
 '观 世界 匈牙利 铝厂 泄漏 有毒 废水 日 上午 流入 多瑙河 支流 莫雄 多瑙河 匈牙利 全国 灾难 管理 部门 发布 公报 说 日 上午 时 27 分 流入 莫雄 多瑙河 铝厂 废水 P H 值为 9.3 专家 监测 河水 污染 度 sinaurl h4xhlh',
 '观 世界 月 日 一艘 油料 驳船 埃及 南部 城市 阿斯旺 沉入 尼罗河 约 110 吨 柴油 泄漏 河中 埃及 各界 关注 埃及 水资源 灌溉 部 官员 日 受 污染 水域 超过 65 公里 阿斯旺 向北 延伸 至康 翁波 sinaurl h4ywNO',
 '观 世界 北京 开往 湛江 K157 次列车 节车厢 广西 脱轨 2010 年 月 日 凌晨 时许 北京 开往 湛江 k157 次 旅客列车 湘桂 线柳南 到达场 脱轨 事故 节车厢 脱轨 人员伤亡 影响 行车 事故 原因 调查 中 sinaurl h4gwmh',
 '观 世界 我够 黑 猜 黑客 2010 年 月 日 消息 维基 解密 风波 掀起 全球 网络战 黑客 组织 声称 召集 四千名 精英 攻击 维基 解密 机构 吃惊 sinaurl hbojE6',
 '观 世界 成都 大雾 137 辆车 追尾 浓雾 司机 难辨 东西 短短 数 小时 里 路段 地点 接连 发生 52 追尾 交通事故 交管部门 统计 共计 137 辆车 连环 追尾 程度 受损 事故 中 程度 受伤 话筒 sinaurl hbWCHn',
 '观 世界 月 日 时 49 分 深圳 地铁 号线 国贸 站 号 手扶 电梯 发生 运行 故障 逆向 下行 致使 24 名 乘客 受伤 事故 原因 调查 中 说 紧握 扶手',
```

Figure.2 Word segmentation

3.2.2 Text matrixing

In the matrixing process, we use Scikit-learn package performs TF-IDF calculation on text content and constructs 10*10 matrix(10 lines, 10 feature words).

TF-IDF is a statistical method used to assess the importance of a word to one of the documents in a document set or corpus.The importance of a word increases proportionally with the frequency of its occurrence in the document, but decreases inversely with the frequency of its occurrence in the corpus. The main idea of TF-IDF is that if a word appears in an article with high frequency and rarely in other articles, then this word or phrase is considered to have a good ability to distinguish categories and is suitable for classification.

Word frequency (TF) represents the frequency of entries (keywords) in the text.This number is normalized to term count to prevent it from skewing to long files.The same word may have a higher number of words in a long file than a short file, regardless of whether the word is important or not. For the

word t in a particular class, tf_ω can be expressed as: $\frac{The\ number\ of\ occurrences\ of\ an\ entry\ \omega\ in\ a\ class}{The\ number\ of\ entries\ in\ the\ class}$ IDF refers to the inverse document frequency:

$$IDF = \log\left(\frac{The\ total\ number\ of\ documents\ in\ the\ corpus}{The\ number\ of\ documents\ containing\ the\ entry\ \omega\ +1}\right)$$

When there is TF and IDF, multiply the two words to get the value of a word's TF-IDF. The larger the TF-IDF of a certain word in the article, the higher the importance of this word in this article in general. Therefore, by calculating the TF-IDF of each word in the article, the first few words in the ranking from large to small is the key word of this article.

The TfidfVectorizer method is used to obtain the lexicon - eigenvector mapping relationship of news data. It converts the original text into a TF-IDF feature matrix, and the minimum frequency of words is equal to 2. The vectorization converter is then applied to the new test data. The TfidfVectorizer has three main attributes: max_df, max_features, min_df. *"max_df"* can be applied to the TF-IDF matrix to describe the highest occurrence rate of words in the document. Assuming that a word appears in 90% of documents, it may carry very little information.*"min_df"* could be an integer x.This means that the word must appear in more than x documents before it can be considered.That is, words appear in at least $1/x$ of documents. *"max_features"* limit the maximum number of words we can use.The model will preferentially select words with high word frequency to remain.Here is a screenshot of a sample line in the matrix. It means that the TF-IDF for the 0th string is 0.4492 for a word with dictionary number 355.

```
(0, 355)	0.44920187469971784
```

3.2.3 Text clustering

3.2.3.1 Basic principle

K - means algorithm needs to specify the number of clusters k in advance, the algorithm begins to randomly select k record point as the center, and then traverse each record of the entire data set , return each record to its nearest in the center of the cluster. Then replace the previous center point with the center point of the recorded mean value of each cluster. The algorithm iterates until convergence. Convergence refers to two aspects. One is that the cluster to which each record belongs does not change, and the other is that the optimization goal does not change much.

3.2.3.2 Optimize the clustering effect

The loss function of K-means is the squared error:

$$RSS_k = \sum_{x \in \omega_k} |x - u(\omega_k)|^2$$

$$RSS = \sum_{k=1}^{K} RSS_k$$

Where ωk is the kth cluster, $u(\omega_k)$ is the center of the kth cluster, RSS_k is the loss function of the kth cluster, RSS is the loss function of the whole.The objective of optimization is to select an appropriate record ownership scheme to minimize the overall loss function. The error function of K-means has a fatal defect, that is, as the number of clusters increases, the error function tends to 0. In the most extreme case, each record is a separate cluster. In this case, the error of the data record is 0. To minimize the

defect, we try different k values in increasing order, and draw the corresponding error value, and find a better k value by looking for the inflection point. To calculate the loss error we use the function: $\frac{k-means.score}{lens(the\quad number\quad of\quad the\quad document\quad lines)}$.

Here is a figure to show the relationship between n_features and error value

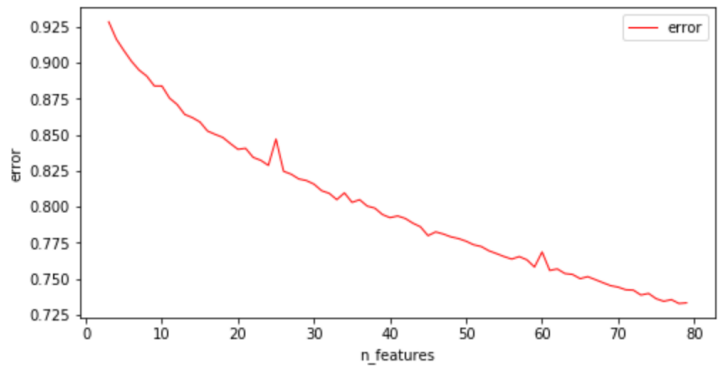


Figure.3 The relationship between n_features and error value

An obvious inflection point appears at $k = 10$ in the figure above, so $k = 10$ is selected as the number of center points. The following is the number of data sets of 10 clusters.

{6: 2216, 8: 8031, 0: 1496, 3: 3499, 5: 2599, 2: 1568, 4: 2305, 7: 1149, 1: 672, 9: 931}

When the initial center point is not well selected, only the local optimal point can be achieved, and the effect of the whole cluster will be relatively not ideal. To avoid this effect, there are several optimized means to choose the center point. Firstly, choose the points as the center points that are as far away from each other as possible.Secondly, carry out the preliminary clustering to output k clusters, and the central point of the cluster will be taken as the input of the central point of K-means. Besides, randomly select the center point to train K-means for several times to select the clustering result with the best effect. We adopt the last mean to optimize the clustering effect.

3.3 Prediction of the popularity

Based on the 24 tags in Tabel.1 and other tags generated from the classes clustered above, we want to make predictions of the Weibo's number of commnets, likes, and reposts into 4 levels: ***small, relatively small, relatively large, large***. First, we want to make predictions by the existing packages of four commonly used algorithms: **Naive Bayes algorithm, K-nearest Neighbor algorithm, Support Vector Machines algorithm, and Random Forest algorithm**. Then, we can choose the algorithm which gets the highest accuracy score and focus more on it.

3.3.1 Comparison of four different prediction models

First, we will give brief introductions of these four algorithms (VanderPlas, 2016).

- **Naive Bayes Classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. To estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric models for the features from the training set. Here we are using the multinomial event model.

- **K-nearest Neighbors algorithm** (K-NN) is a non-parametric method proposed by Thomas Cover used for classification and regression. In both cases, the input consists of the k closest training

examples in the feature space.The choice of k is important for this model; generally, larger values of k reduces effect of the noise on the classification,but make boundaries between classes less distinct. In this case, after trails, we choose k of 4.

- **Support-Vector Machines** (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier Random Forest algorithm
- **Random Forests** or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. In this case, we are using 200 estimators.

Using these four models, we made predictions of the number of comments, like, and reposts with these tags. The accuracy scores of each case are shown below (see Table.3)

Table.3 Accuracy scores of the four algorithms (using package)

	Number of comments	Number of likes	Number of reposts
Naive Bayes	0.4479	0.4626	0.4928
K-nearest neighbours	0.7370	0.8051	0.7613
Support Vector Machines	0.7413	0.7197	0.7734
Random Forest algorithm	0.7642	0.8167	0.7754

According to the table, we can find that the Random Forest algorithm suits this dataset most in general. The reason may be:

- for Naive Bayes algorithm (NB), it got much lower accuracy than the other three. It may result from that NB assumes all indexes are independent, but in this case, the correlations between these indexes are high
- for Support Vector Machine (SVM), though it has high accuracy scores, this algorithm’s efficiency is low when it comes to high-dimensional classification problems, since the classical SVM algorithm gives only binary classification, which is not suitable for this case.
- for K-Nearest Neighbour algorithm (KNN), it fails to give the intrinsic meaning of the data, which makes it hard to find the 5 most important tags affecting the predictions.As a result, we want to adjust the code of the normal code for Random Forest algorithm.

3.3.2 Coding random forest algorithm

Random forest is a non-parametric algorithm, which can be understood as an ensemble of simpler classifiers, **Decision trees** (VanderPlas, 2016). Therefore, in order to build a Random forest, we first look into the construction of Decision trees.

3.3.2.1 Consturcting Decision Tree

To build a decision tree *(‘class DecisionTree()’)*, we do bagging first *(‘bootstrapping(self)’)*, through which a set number of samples are extracted from the database to construct a tree. The randomness of bootstrapping lowers the correlation between each tree.

For a set of samples, we use information entropy *('calculate_entropy(data)')*, indicating the uncertainty of random variables, to decide the order of splitting tags that goes in a tree. The overall entropy *((calculate_overall_entropy(data_below, data_above))* is the sum of information entropy, calculated as:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log(p(x_i))$$

, where $p(x_i)$ is the probability of choosing this class xi.

The bigger the entropy, the more unstable the random variable is, so the factor that gives the smallest entropy will be selected as the decisive factor in that step. For numerical data rather than *yes-no questions*, we do all the potential splitting conditions *('get_potential_splits(self,data)')* and then use the one giving smallest entropy to determine where to split the data *('determine_best_split(self,data, potential_splits)')*. Similar to above, the split that gives the smallest entropy will be selected, since it gives a more consistent subgroup.

A DecisionTree is built in the type of dictionary. For each split, both left and right nodes are linked to a subtree or reach final classification. Therefore, recursion is made to calculate the entropy again to determine the decisive factor of subtree as well as the best split, since subtrees are gone through the exact procedure as above *('decision_tree_algorithm(self, df, counter=0)')*. A tree is done when the division has already reached the initially set depth, or the sample in a group is not big enough to provide a useful split, or all the samples are already in the same class *('check_purity(data)')*. As the test case goes through a decision tree, it will return the classification of all pieces (see Figure.4).

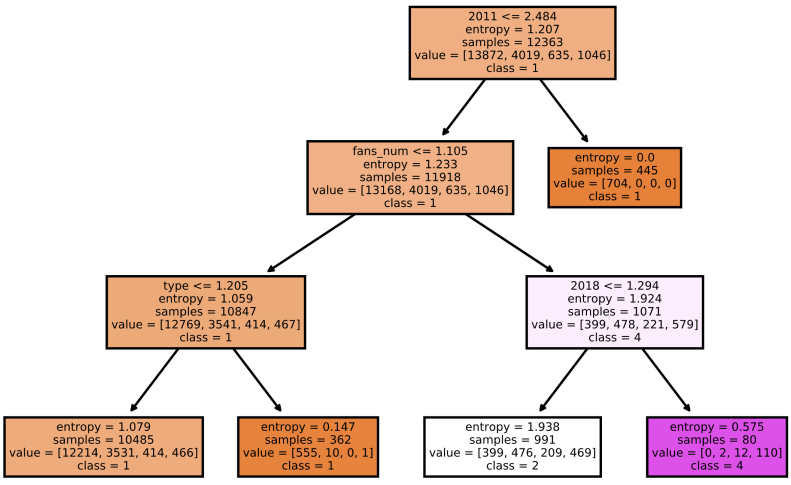


Figure 4. Sample Decision Tree

3.3.2.2 Assemble as Random Forest

Constructing a Random forest (*'RandomForest()'*) is similar to assembling the previously-built Decision trees (*'random_forest_algorithm(self)'*), and we build a list to stand for a forest. To determine the final prediction of a piece of testing sample, we take the most repeated classification result of all the decision trees in the forest as the ultimate result (*'random_forest_predictions(self)'*). The accuracy of the Random forest algorithm is tested by comparing the prediction outcome and the classification of original dataset (*'calculate_accuracy(predictions, labels)'*).

3.3.2.3 Prediction argument

Since we would like to predict three features (*comment number, like number, and repost number*), we adjust the code to fit our need so that we can directly choose the one we want to predict whitin the RandomForest algorithm by indicating the column in the arguemnts (*'y', in our code*).

4 Results

4.1 Result of clustering

In general, we cluster all news into 7 classes.

4.1.1 Distribution of clustering result

We cluster the text into 10 classes by using the k-means algorithm (see Figure.5).

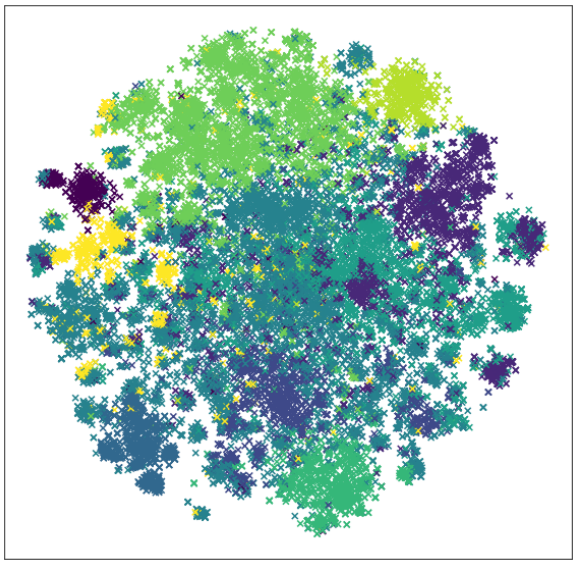


Figure.5 Visualization of clustering results

Then, we manually adjust the result into 7 classes. We give tags to describe each cluster. The clustering results and key words are as follows (see Figure.6):

Cluster 0 发生爆炸 爆炸 受伤 现场 死亡 事故 化工厂 伤亡 记者 原因
（爆炸事故 explosion accident）
Cluster 1 滑坡 山体 救援 现场 搜救 发生 人员 深圳 失踪 灾害
（地质灾害 geological disaster）
Cluster 2 中国 飞机 坠机 客机 坠毁 公民 航空 一架 遇难 乘客
（航空事故 aviation accident）
Cluster 3 爆炸 事故 天津 天津港 发生 死亡 滨海新区 现场 煤矿 仓库
（爆炸事故 explosion accident）
Cluster 4 地震 灾区 救援 鲁甸 芦山 救灾 云南 雅安 四川 抗震救灾
（地震灾害 earthquake disaster）
Cluster 5 嫌疑人 犯罪 宣判 被告人 死刑 杀人 警方 故意杀人罪 杀人案 一审
（刑事案件 criminal case）
Cluster 6 地震 发生 死亡 受伤 云南 房屋 遇难 四川 中国 倒塌
（地震灾害 earthquake disaster）
Cluster 7 视频 火灾 森林 扑救 火场 发生 消防 现场 消防员 爆炸
（森林火灾 forest fire）
Cluster 8 死亡 发生 cn http 事件 警方 记者 男子 受伤 事故
（刑事案件 criminal case）
Cluster 9 禽流感 h7n9 感染 病例 确诊 患者 例人 新增 病毒 死亡
（病毒流感 flu virus）

Figure.6 Manually adjusted clustering results and key words

The tags for the adjusted cluster are: explosion accident, geological disaster, aviation accident, earthquake disaster, criminal case, forest fire, flu virus.
The numebr of news in each class are shown in Table.4

Table.4 Number of news in each class

criminal case	10630
earthquake disaster	4521
explosion accident	4995
aviation accident	1568
forest fire	1149
flu virus	931
geological disaster	672

4.1.2 Example

For example, here is the content of one randomly selected news from our test set:

> 【青海杂多地震受灾牧民得到安置】10月17日15时14分，青海省玉树藏族自治州杂多县发生6.2级地震。此次地震导致全县3000余户牧民房屋不同程度受损。目前杂多县城区和各乡镇已分别设立安置点，对已排查出的受灾牧民全部进行了安置。#玉树6.2级地震 #
> (At 15:14 on October 17, a 6.2-magnitude earthquake hit Zado County, Yushu Tibetan Autonomous Prefecture, Qinghai Province.The earthquake caused damage to the houses of more than 3,000 herdsmen in the county.At present, zadoi county urban area and each township have set up their own resettlement sites, and all the affected herdsmen have been settled.# Yushu 6.2 Earthquake #)

Based on our model, it was assigned to be a news of type **earthquake disaster**.

4.2 Result of prediction

In general, we made predictiosn of a new's comment number, likes number, and reposts number into **small, relatively small, relatively large, large**.

4.2.1 Accuracy

According to the algorithm of random forest above, our adjusted code for random forest algorithm achieves satisfactory accuracy scores (see Table.5)

Table.5 Accuracy scores of adjusted code for random forest algorithm

Comments	0.763131
Likes	0.789904
Reposts	0.773963

Though the accuracy score is not as high as it was when we use the package of random forest, it is not

much lower and it is higher than 75%. In general, we made accurate predictions.

4.2.2 example

For example, here is the information of the above randomly selected news.

> 21907,856972,2016-10-19 12:40:03,http://ww2.sinaimg.cn/wap180/a782e4abjw1f8xcvvqnykj20p00godhl.jpg ,,,【青海杂多地震受灾牧民得到安置】10月17日15时14分，青海省玉树藏族自治州杂多县发生6.2级地震。此次地震导致全县3000余户牧民房屋不同程度受损。目前杂多县城区和各乡镇已分别设立安置点，对已排查出的受灾牧民全部进行了安置。#玉树6.2级地震# ,2810373291,67096486,6,93603,3,0,2,0,1,1,2016,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,6,0,0,0,1,0,0,0

Based on our random forest prediction model, the result is that
1) the comment number of it is **relatively large**
2) the like number of it is **relatively large**
3) the repost number of it is **large**.

4.3 Five most important tags for predictions

According to our algorithm, the top 5 tags that contribute to the like number in sequence are: *fans number, tweets number, the year 2013, hash number and the year 2012*. The top 5 tags that contribute to the comment number in sequence are: *fans number, tweets number, vip level, hash number and at number*. The top 5 tags that contribute to the repost number in sequence are: *fans number, tweets number, vip level, hash number and at number* (see Figure.7). In summary, **fans number, tweets number, vip level, hash number and at number** contribute a lot to the prediction model.

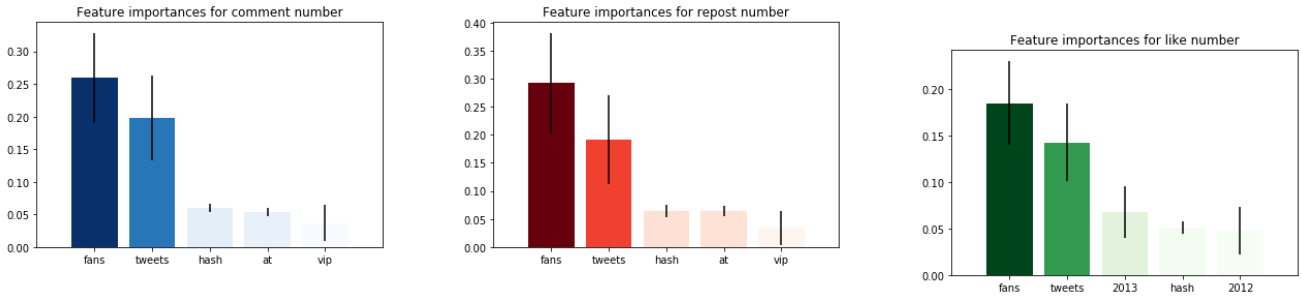


Figure.7 Five most important tags for predictions

Fans number can be understood or abstracted as a platform under the special browsing settings of weibo. Each user faces a different platform, and the number of fans is the size of this platform. Although is in the same network system, the user is the starting point of information transfer unfair, and this platform will effect as forward further affects the transmission of information in the form of index.

Tweets number partly concludes the accounts' state of operation. The accounts with high tweets number shows that it is under a well operation status which will attract more users' attention.

Hash number refers to the topic involved in the news. Topic is one of the most important interest pages in Weibo. Weibo users can enter the topic to publish and participate in the discussion. At the same time, the topic page will automatically include the relevant microblog with the inscription. Using topic module

can improve user searching term's matching accuracy. Hot topics can be recommended or even searched on the home page. Therefore, it contributes a lot in the prediction model.

News types also influence the transmission model (see Table.6 and Table.7). As for news related to viruses, it tends to share information on the number of confirmed and the cure cases. After reading this kind of news, users prefer to repost the news to share the information and knowledge to someone close. So the median of the repost number is 64. However, as for the news related to the explosion accident. People may be impressed by the heroism of the fireman or shocked by the enormity of the accident. So they prefer to give a like rather than repost the news.

Table.6 Data summary for news related to virus(left)

	comment_num	repost_num	like_num
count	931.000000	931.000000	931.000000
mean	83.045113	267.094522	43.308271
std	366.571579	985.592646	411.905732
min	0.000000	0.000000	0.000000
25%	10.000000	29.000000	2.000000
50%	21.000000	64.000000	7.000000
75%	64.000000	187.500000	16.000000
max	9293.000000	21580.000000	9459.000000

Table.7 Data summary for news related to explosion accidents(right)

	comment_num	repost_num	like_num
count	1149.000000	1149.000000	1149.000000
mean	265.514360	271.173194	1306.254134
std	1485.317496	1858.590974	8465.307491
min	0.000000	0.000000	0.000000
25%	7.000000	8.000000	13.000000
50%	28.000000	37.000000	96.000000
75%	127.000000	102.000000	361.000000
max	38204.000000	49195.000000	122849.000000

5 Conclusions

In all, we made clustering of the news type based on its content and built a model to predict the popularity of a Weibo. We also find the 5 most important tags for the prediction. Here is a conclusion of the strength and weakness of our model, and advice for the new media operation.

5.1 Strengths of the model

- Only use basic packages
  In our adjusted code for random forest algorithm, we only used the most basic packages: numpy and pandas, and achieved a relatively high accuracy score.
- Combain clustering with prediction
  Unlike other studies of prediction of news' popularity, we clustered the typr of the news first and use it as an index, which improves the accuracy score.
- Database matches model
  Most of the tags in our database are dummy variables, which suits the Random Forest algorithm.

5.2 Weakness of the model

- Lack of consideration of 'emotional words'
  Both positive and negative emotions have a significant impact on information dissemination according to Xiong.et al. As a result, whether the content has emotional words will affect its popularity. However, due to the limited time, we fail to contain this part in our model. Future studies can focus more on this part.
- The huge time span
  The time span of the released date is huge in our dataset. Though we have proved that , especially for Weibo about news, the accumulation of comments, likes, and reposts of a Weibo is low after 7 days, there may still be an effect on the prediction.

- Long operation time
  Our code takes longer to operate.

5.3 Advice for the new media operation

Micro-blog is the latest and popular route in the field of news transmission. Its difference from traditional transmission routes lies in its stronger interactivity, wider possible audience and lower cost.According to the results of this study, the key to microblog in the field of news transmission is to attract more fans.In addition, the information represented by the content (like the hash number @number) has an important impact on the communication behavior on the platform of Weibo, and pictures also play an important positive role in forwarding.Accordingly, we can put forward the following strategy about micro blog operation in the news field.

- Increase the “findability”
  a) Use celebrity influencers, ask the celebrity to repost the microblog.Information released on weibo spreads virally in the form of fans' attention, thus having a very wide impact. Compared with ordinary weibo users, celebrity information is often spread more widely on Weibo, that is to say, celebrity effect can make the spread of an event geometrically amplified.Therefore, celebrity effect is a very effective means of microblog marketing and promotion, which makes the information of enterprise products or brands more attractive and convincing.
  b) Microblog should be updated with high frequency (increase the tweets number) to increase "exposure rate".

- Optimize content and pictures
  a) Improve the quality of the content
  b) Microblog marketing should convey as much information as possible and be accompanied by carefully selected pictures. Vivid pictures not only increase information density, but also increase vividness of the content, making it easier to induce trust and forwarding.

6 Individual Contribution

Group Member 1 Sissi Wang

- Database selection and clean
- Clustering model annotation
- KNN, Random Forest implementation (with package)
- Coding Random forest algorithm
- Data and code packaging
- Report writing (Data quantification, prediction with four packages, conclusion)\

Group Member 2 Tingyi Li

- Searching background
- Clustering model annotation
- Naïve Bayes, SVM implementation (with package)
- Coding Random forest algorithm
- Visualization of the data
- Report writing (introduction, background, adjust random forest code)

Group Member 3 Yijia Xue

- Database selection and clean
- Clustering model selection
- Clustering model implementation
- Coding Random forest algorithm
- Visualization of the data
- Report writing(clustering, 5 most important tags)

References

Cao Zairong 曹荣芳. (2015). *Weibozaixinwen xinwenchuanbozhongde yingxiang hefudaiwenti* 微博在新闻传播中的影响和附带问题 [The influence and collateral problems of Microblog in news communication]. *Kejichuanbo* 科技传播 [science and technology communication], (011), 123-124. Retrieved from: https://wenku.baidu.com/view/463ddea277eeaeaad1f34693daef5ef7bb0d1260.html (https://wenku.baidu.com/view/463ddea277eeaeaad1f34693daef5ef7bb0d1260.html)

Han Shaoqing 韩少卿. (2018). *Wangluoyuqing redianshijian chuanbodeshengmingzhouqi yanjiu* 网络舆情热点事件传播的生命周期研究[Research on the life cycle of network Public opinion hot event dissemination]. *DongnanChuanbo* 东南传播[Southeast communication], 170(10), 88-90. DOI：CNKI:SUN:DNCB.0.2018-10-028

Huang qimei 黄琦媚. (2017). *Congxinlangweibo kanweibodui xinwenchuanbodeyingxiang *从新浪微博看微博对新闻传播的影响[Influence of microblog on news communication from Sina weibo]. *Zhishiwenku* 知识文库[Knowledge Library] (23),194.

Malcolm, G. (2002). *The Tipping Point how Little Things Can Make a Big Difference.* Back Bay Books.

Python Data Science Handbook: *Essential Tools for Working with Data, by Jake VanderPlas, O'Reilly Media; 1 edition* (December 10, 2016), ISBN-13: 978-1491912058, full text and code freely available at https://jakevdp.github.io/PythonDataScienceHandbook/ (https://jakevdp.github.io/PythonDataScienceHandbook/).

Rongying, Z. (2014). Research on Influencing Factors of Information Dissemination in Micro-blog. *Information Studies: Theory & Application*, 3.

Sina (2020). Retrieved from: https://www.sina.com.cn (https://www.sina.com.cn),

Xiong, X., Li, Y., Qiao, S., Han, N., Wu, Y., Peng, J., & Li, B. (2018). An emotional contagion model for heterogeneous social media with multiple behaviors.* Physica A,* 490, 185-202. doi:10.1016/j.physa.2017.08.025

Zheng Kexin 郑珂歆. (2014).* Shejiao meiti dui xinwenchuanbo de yingxiang *社交媒体对新闻传播的影响[The influence of social media on news communication. Young journalists]. *Qingnianjizhe *青年记者[Young journalists], (22), 71-72. Retrieved from: http://www.cqvip.com/qk/82782x/2014022/662902261.html (http://www.cqvip.com/qk/82782x/2014022/662902261.html)

Appendix

In [1]:

```
'''
data processing (first step)
'''
import pandas as pd
#load the raw data
data = pd.read_csv('raw.csv')
#classify the weibo based on the hour it was released
def split_time(x):
    hour = x['created_at'].hour
    if hour<6:
        return 0
    if hour<9:
        return 1
    if hour<12:
        return 2
    if hour<14:
        return 3
    if hour<18:
        return 4
    return 5

# quantificate the indexes
data['created_at'] = pd.to_datetime(data['created_at'])
data['time'] = data.apply(lambda x:split_time(x),axis=1)
data['at_num'] = data.apply(lambda x:x['content'].count('@'),axis=1)
data['hash_num'] = data.apply(lambda x:x['content'].count('#'),axis=1)
data['has_video'] = data.apply(lambda x:0 if pd.isnull(x['video_url']) else 1,axis=1)
data['has_image'] = data.apply(lambda x:0 if pd.isnull(x['image_url']) else 1,axis=1)
data['is_origin'] = data.apply(lambda x:1 if pd.isnull(x['origin_weibo']) else 0,axis=1
data['year'] = data.apply(lambda x:pd.to_datetime(x['created_at']).year,axis=1)
data=data.drop('Unnamed: 0',axis=1)

# dimensionality reduction
data = data.join(pd.get_dummies(data.time))
data = data.join(pd.get_dummies(data.year))
data.to_csv('Weibo.csv')
```

In [ ]:

```
import sys
!{sys.executable} -m pip install jieba
```

In [ ]:

```
'''
cluster of different news type
For this part, the result may be different every time clustering was run, so the result m
'''

import jieba
import pickle
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans, MiniBatchKMeans
import codecs
from sklearn.manifold import TSNE

#put the content column into a csv file
y = data['content']
y.to_csv('content.csv')
#drop the index of content.csv
data = pd.read_csv('content.csv')
data.columns = ['index','content']
data = data.drop(['index'], axis=1)
data.to_csv('content1.csv')

def getStopWords():
    """
    creat a list of 'stopwords'
    as the base to split the content into list of words
    """
    stopwords = []
    #import the txt file
    for word in open("stopwords.txt", "r",encoding='utf-8'):
        stopwords.append(word.strip())
    return stopwords

def load_articles():
    """
    split the content into words
    """
    #get the content and stopwords
    stop_words=getStopWords()
    data = pd.read_csv('content1.csv')
    #show the process
    print("clustering now ...")
    #cut the content to words and put words that are not in the stopwords list into one
    data['content'] = data['content'].apply(
        lambda x: " ".join([word for word in jieba.cut(str(x)) if word not in stop_wor

    #put these words (one index for one Weibo) into a list
    articles = []
    for  content in data['content'].tolist():
        article =  content
        articles.append(article)
    return articles

def transform(articles, n_features=1000):
    """
```

```
    get the data of tf-idf of each words
    """
    #get the vectorized result of Tfidf of the words in each Weibo
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=n_features, min_df=5, use_idf=
    X = vectorizer.fit_transform(articles)
    return X, vectorizer


def train(X, vectorizer, true_k=10, mini_batch=False, show_label=False):
    """
    cluster the data with k-means algrithm
    """
    #MiniBatch K-means algorithm's main idea is to use small random batches of data of a
    #MiniBatch K-means is used when the dataset is large, to reduce the time and memory c
    if mini_batch:
        k_means = MiniBatchKMeans(n_clusters=true_k, init='k-means++', n_init=1,
                                  init_size=1000, batch_size=1000, verbose=False)
    #when the dataset is not large, just use the K-means algrithm
    else:
        k_means = KMeans(n_clusters=true_k, init='k-means++', max_iter=300, n_init=1,
                         verbose=False)
    k_means.fit(X)

    # show the distribution of the classes
    def plot1(something):
        tsne = TSNE(n_components=2)
        decomposition_data = tsne.fit_transform(something)
        x = []
        y = []
        for i in decomposition_data:
            x.append(i[0])
            y.append(i[1])
        fig = plt.figure(figsize=(10, 10))
        ax = plt.axes()
        plt.scatter(x, y, c=k_means.labels_, marker="x")
        plt.xticks(())
        plt.yticks(())
        plt.show()

    if show_label:
        print("Top terms per cluster:")
        #find the key words
        order_centroids = k_means.cluster_centers_.argsort()[:, ::-1]
        terms = vectorizer.get_feature_names()
        for i in range(true_k):
            print("Cluster %d" % i, end='')
            #show the most important 10 key words
            for ind in order_centroids[i, :10]:
                print(' %s' % terms[ind], end='')
            print()

    #put the results of clustering of each Weibo in the list
    result = list(k_means.predict(X))
    plot1(X.toarray())
    datanew = pd.read_csv('Weibo.csv')
    resultpd = pd.Series(list(k_means.predict(X)))
    datanew['type'] = resultpd
    #join the result of the clustering to the dataframe
    datanew = datanew.join(pd.get_dummies(datanew.type))
    datanew.to_csv('Weibo&cluster.csv')
    #show the result
```

```
    print('Cluster distribution:')
    print(dict([(i, result.count(i)) for i in result]))
    #return the purity of k-means algrithm
    return -k_means.score(X)


def plot_params():
    """
    test and find the best parameter
    """
    #get all words in the content
    articles = load_articles()
    print("%d docments" % len(articles))
    #use tf-idf to show the distance of two contents
    X, vectorizer = transform(articles, n_features=500)
    true_ks = []
    scores = []
    #try k from 3 to 80 to find how many clusters should be used based on the purity
    for i in range(3, 80, 1):
        score = train(X, vectorizer, true_k=i) / len(articles)
        true_ks.append(i)
        scores.append(score)
    #plot a figure to show the trend
    plt.figure(figsize=(8, 4))
    plt.plot(true_ks, scores, label="error", color="red", linewidth=1)
    plt.xlabel("n_features")
    plt.ylabel("error")
    plt.legend()
    plt.show()

def out():
    """
    show the reslut of clustering with the parameter of highest purity
    """
    articles = load_articles()
    X, vectorizer = transform(articles, n_features=500)
    score = train(X, vectorizer, true_k=10, show_label=True) / len(articles)

out()
```

In [ ]:

```
#make adaption of the clustering by hand
dataclu=pd.read_csv('Weibo&cluster.csv')
dataclu['爆炸事故']=dataclu['0.1']+dataclu['3.1']
dataclu['地质灾害']=dataclu['1.1']
dataclu['航空事故']=dataclu['2.1']
dataclu['地震灾害']=dataclu['4.1']+dataclu['6']
dataclu['刑事案件']=dataclu['5.1']+dataclu['8']
dataclu['森林火灾']=dataclu['7']
dataclu['病毒流感']=dataclu['9']
dataclu=dataclu.drop(['0.1','1.1','2.1','3.1','4.1','5.1','6','7','8','9','Unnamed: 0', '
#rename the columns
dataclu.rename(columns={'0':'0:6','1':'6:9','2':'9:12','3':'12:14','4':'14:18','5':'18:24
dataclu.to_csv('dataclustered.csv')
```

In [ ]:

```
'''
Multinomial Naive Bayes--predicttion of comment_num,like_num,repost_num
'''
#set the factors used for prediction
import pandas as pd
data = pd.read_csv('dataclustered.csv')
df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
X = df
# import the packages needed
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

list_accuracy=list()
#set the lables of comment_num
bins = [-1, 50, 100, 1000, 136265]
data['comment_num'] = pd.cut(data['comment_num'], bins,labels=[1,2,3,4],right=True)
y = data['comment_num']
#predict the number of comment
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
model = MultinomialNB().fit(X_train, y_train)
predicted = model.predict(X_test)
list_accuracy.append(accuracy_score(y_test, predicted))

#set the lables of like_num
bins = [-1,75 , 500, 1000, 306161]
data['like_num'] = pd.cut(data['like_num'], bins,labels=[1,2,3,4],right=True)
y = data['like_num']
#predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
model = MultinomialNB().fit(X_train, y_train)
predicted = model.predict(X_test)
list_accuracy.append(accuracy_score(y_test, predicted))

#set the lables of like_num
bins = [-1, 100, 500, 1000,  565454]
data['repost_num'] = pd.cut(data['repost_num'], bins,labels=[1,2,3,4],right=True)
y = data['repost_num']
#predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
model = MultinomialNB().fit(X_train, y_train)
predicted = model.predict(X_test)
list_accuracy.append(accuracy_score(y_test, predicted))

print(list_accuracy)
```

In [ ]:

```
'''
KNN--predicttion of comment_num,like_num,repost_num
'''
#set the factors used for prediction
import pandas as pd
data = pd.read_csv('dataclustered.csv')
df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
X = df
# import the packages needed
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

list_accuracy=list()
#set the lables of comment_num
bins = [-1, 50, 100, 1000, 136265]
data['comment_num'] = pd.cut(data['comment_num'], bins,labels=[1,2,3,4],right=True)
y = data['comment_num']
# predict the number of commments
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train);
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of like_num
bins = [-1,75 , 500, 1000, 306161]
data['like_num'] = pd.cut(data['like_num'], bins,labels=[1,2,3,4],right=True)
y = data['like_num']
# predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train);
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of repost_num
bins = [-1, 100, 500, 1000,  565454]
data['repost_num'] = pd.cut(data['repost_num'], bins,labels=[1,2,3,4],right=True)
y = data['repost_num']
# predict the number of reposts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train);
```

```
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

print(list_accuracy)
```

In [ ]:

```
'''
SVM--predicttion of comment_num,like_num,repost_num
'''
#set the factors used for prediction
import pandas as pd
data = pd.read_csv('dataclustered.csv')
df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
X = df

#import the packages needed
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

list_accuracy=list()
#set the lables of comment_num
bins = [-1, 50, 100, 1000, 136265]
data['comment_num'] = pd.cut(data['comment_num'], bins,labels=[1,2,3,4],right=True)
y = data['comment_num']
#predict the number of comments
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
svclassifier = SVC(kernel='rbf',gamma='scale')
svclassifier.fit(X_train, y_train);
y_pred = svclassifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of like_num
bins = [-1,75 , 500, 1000, 306161]
data['like_num'] = pd.cut(data['like_num'], bins,labels=[1,2,3,4],right=True)
y = data['like_num']
#predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
svclassifier = SVC(kernel='rbf',gamma='scale')
svclassifier.fit(X_train, y_train);
y_pred = svclassifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of repost_num
bins = [-1, 100, 500, 1000,  565454]
data['repost_num'] = pd.cut(data['repost_num'], bins,labels=[1,2,3,4],right=True)
y = data['repost_num']
#use the SVM package to predict the number of reposts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
svclassifier = SVC(kernel='rbf',gamma='scale')
svclassifier.fit(X_train, y_train);
y_pred = svclassifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

print(list_accuracy)
```

In [ ]:

```
'''
Random Forest--predicttion of comment_num,like_num,repost_num
'''
#set the factors used for prediction
import pandas as pd
data = pd.read_csv('dataclustered.csv')
df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
X = df
#import packages needed
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

list_accuracy=list()
#set the lables of comment_num
bins = [-1, 50, 100, 1000, 136265]
data['comment_num'] = pd.cut(data['comment_num'], bins,labels=[1,2,3,4],right=True)
y = data['comment_num']
#predict the number of comments
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = RandomForestClassifier(n_estimators=200, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of like_num
bins = [-1,75 , 500, 1000, 306161]
data['like_num'] = pd.cut(data['like_num'], bins,labels=[1,2,3,4],right=True)
y = data['like_num']
#predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = RandomForestClassifier(n_estimators=200, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

#set the lables of repost_number
bins = [-1, 100, 500, 1000,  565454]
data['repost_num'] = pd.cut(data['repost_num'], bins,labels=[1,2,3,4],right=True)
y = data['repost_num']
#predict the number of likes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = RandomForestClassifier(n_estimators=200, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
list_accuracy.append(accuracy_score(y_test, y_pred))

```

```
58 print(list_accuracy)
```

In [ ]:

```
'''
Adjusted Random Forest algorithm
'''
import pandas as pd
import numpy as np
import random

def train_test_split(df, y, test_size): ##Split the train and test data
    df['label'] = y
    test_size = round(test_size * len(df))
    indices = df.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)
    test_df = df.loc[test_indices]
    train_df = df.drop(test_indices)
    return train_df, test_df

def calculate_accuracy(predictions, labels): ##Calculate accuracy of prediction
    predictions_correct = (predictions == labels)
    accuracy = predictions_correct.mean()
    return accuracy

def check_purity(data): ##Check whether all the data are in the same class
    label_column = data[:, -1]
    unique_classes = np.unique(label_column)
    if len(unique_classes) == 1: # this statement is satisfied only when the np.unique g
        return True
    else:
        return False

def classify_data(data): ##Classify the data according to different class in a factor
    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)
    index = counts_unique_classes.argmax() # take the class that appears the most
    classification = unique_classes[index]
    return classification

def calculate_entropy(data): ##Calculate the entropy of a situation
    label_column = data[:, -1]
    counts = np.unique(label_column, return_counts=True)[1]
    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))
    return entropy

def calculate_overall_entropy(data_below, data_above): ##Add up the total entropy of a n
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n
    overall_entropy = (p_data_below * calculate_entropy(data_below)
                       + p_data_above * calculate_entropy(data_above))
    return overall_entropy

def split_data(data, split_column, split_value): ##Split a node
    split_column_values = data[:, split_column]
    # determine whether it belongs to the below or the above using fancy indexing
    data_below = data[split_column_values <= split_value]
    data_above = data[split_column_values > split_value]
    return data_below, data_above
```

```
58
59  class RandomForest():
60      def __init__(self, df, y, n_trees,n_bootstrap, n_features, dt_max_depth, min_samples
61          self.df, self.y, self.n_trees, self.n_bootstrap, self.n_features  = df, y, n_tre
62          self.dt_max_depth, self.min_samples = dt_max_depth, min_samples
63          self.train_df, self.test_df = train_test_split(df, y, test_size=0.2)
64          self.trees, self.forest = self.random_forest_algorithm()
65
66      def bootstrapping(self): ## Bagging (bootstrap aggregating)
67          # randomly select n_bootstrap number of pieces of data to build a tree later
68          bootstrap_indices = np.random.randint(low=0, high=len(self.train_df), size=self.
69          df_bootstrapped = self.train_df.iloc[bootstrap_indices]
70          return df_bootstrapped
71
72      def random_forest_algorithm(self): ##Build random forest (list)
73          trees =[]
74          forest = []
75          for i in range(self.n_trees):
76              df_bootstrapped = self.bootstrapping()
77              tree = DecisionTree(df_bootstrapped, self.n_features, self.dt_max_depth, sel
78              trees.append(tree)
79              forest.append(tree.decision_tree_algorithm(tree.df))
80          return trees, forest
81
82      def random_forest_predictions(self): ##Go through all the trees and return the class
83          df_predictions = {}
84          for i in range(len(self.trees)):
85              column_name = "tree_{}".format(i)
86              predictions = self.trees[i].decision_tree_predictions(self.test_df, tree=sel
87              df_predictions[column_name] = predictions
88          df_predictions = pd.DataFrame(df_predictions)
89          random_forest_predictions = df_predictions.mode(axis=1)[0]
90          return random_forest_predictions
91
92
93  class DecisionTree():
94      def __init__(self, df, n_features, max_depth, min_samples,random_subspace=None):
95          self.df, self.n_features, self.max_depth, self.min_samples = df, n_features, max_
96          self.random_subspace = random_subspace
97
98      def get_potential_splits(self,data): ##Potential splits
99          potential_splits = {}
100         column_indices = list(range(data.shape[1] - 1))  # excluding the last column (la
101         if self.random_subspace and self.random_subspace <= len(column_indices):
102             # randomly select k columns from column_indices
103             column_indices = random.sample(population=column_indices, k=self.random_subs
104         for column_index in column_indices:
105             values = data[:, column_index]
106             unique_values = np.unique(values)
107             potential_splits[column_index] = unique_values
108         return potential_splits
109
110     def determine_best_split(self,data, potential_splits): ##Choose the split with lowes
111         overall_entropy = 102102
112         # go through all the potential splits
113         for column_index in potential_splits:
114             for value in potential_splits[column_index]:
115                 data_below, data_above = split_data(data, split_column=column_index, spl
116                 current_overall_entropy = calculate_overall_entropy(data_below, data_abo
117                 # compare total entropy and select
118                 if current_overall_entropy <= overall_entropy:
```

```
                    overall_entropy = current_overall_entropy
                    best_split_column = column_index
                    best_split_value = value
        return best_split_column, best_split_value

    def decision_tree_algorithm(self, df, counter=0): ##Build a decision tree by diction
        # data preparations
        if counter == 0:
            global COLUMN_HEADERS
            COLUMN_HEADERS = self.df.columns
            data = self.df.values
        else:
            data = df
        # all the conditions that indicates the end of splitting a tree
        if (check_purity(data)) or (len(data) < self.min_samples) or (counter == self.m
            classification = classify_data(data)
            return classification
        # recursion
        else:
            counter += 1 # in case it reaches the man_depth
            # helper functions
            potential_splits = self.get_potential_splits(data)
            split_column, split_value = self.determine_best_split(data, potential_splits)
            data_below, data_above = split_data(data, split_column, split_value)
            # return if data is already empty
            if len(data_below) == 0 or len(data_above) == 0:
                classification = classify_data(data)
                return classification
            # determine question
            feature_name = COLUMN_HEADERS[split_column]
            # if print the tree, the sturcture would be like this
            question = "{} <- {}".format(feature_name, split_value)
            # instantiate sub-tree
            sub_tree = {question: []}
            # find answers (recursion)
            yes_answer = self.decision_tree_algorithm(data_below, counter)
            no_answer = self.decision_tree_algorithm(data_above, counter)
            # If the answers are the same, then there is no point in asking the question.
            if yes_answer == no_answer:
                sub_tree = yes_answer
            else:
                sub_tree[question].append(yes_answer)
                sub_tree[question].append(no_answer)
            return sub_tree

    def predict_example(self,example, tree): ##Prediction that goes down a tree
        # print(tree)
        question = list(tree.keys())[0]
        feature_name, comparison_operator, value = question.split(" ")
        # splitting queistion
        if example[feature_name] <= float(value):
            answer = tree[question][0]
        else:
            answer = tree[question][1]
        # return if it is already not a dictionary (which means it reaches the end)
        if type(answer) != dict:
            return answer
        # recursion
        else:
            residual_tree = answer
            return self.predict_example(example, residual_tree)
```

```
180
181     # All examples of the test data
182     def decision_tree_predictions(self,test_df, tree):
183         predictions = test_df.apply(self.predict_example, args=(tree,), axis=1)
184         return predictions
```

In [ ]:

```
1  '''
2  Use our own Random Forest algorithm for prediction
3  '''
4  #set the dataFrame to predict comment_num
5  data = pd.read_csv('dataclustered.csv')
6  df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
7  bins = [-1, 50, 100, 1000, 136265]
8  data['comment'] = pd.cut(data['comment_num'], bins,labels=[1,2,3,4],right=True)
9  # Build random forest class
10 My_Random_Forest = RandomForest(df, data['comment'], n_trees=200,n_bootstrap=100, n_featu
11 predictions = My_Random_Forest.random_forest_predictions()
12 accuracy = calculate_accuracy(predictions,My_Random_Forest.test_df.label)
13 print(accuracy)
14
15 #set the dataFrame to predict like_num
16 data = pd.read_csv('dataclustered.csv')
17 df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
18 bins = [-1,75 , 500, 1000, 306161]
19 data['like'] = pd.cut(data['like_num'], bins,labels=[1,2,3,4],right=True)
20 # Build random forest class
21 My_Random_Forest = RandomForest(df, data['like'], n_trees=200,n_bootstrap=100, n_features
22 predictions = My_Random_Forest.random_forest_predictions()
23 accuracy = calculate_accuracy(predictions,My_Random_Forest.test_df.label)
24 print(accuracy)
25
26 #set the dataFrame to predict repost_num
27 data = pd.read_csv('dataclustered.csv')
28 df=data.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','created_at','image
29 bins =  [-1, 100, 500, 1000,  565454]
30 data['repost'] = pd.cut(data['repost_num'], bins,labels=[1,2,3,4],right=True)
31 # Build random forest class
32 My_Random_Forest = RandomForest(df, data['repost'], n_trees=200,n_bootstrap=100, n_featu
33 predictions = My_Random_Forest.random_forest_predictions()
34 accuracy = calculate_accuracy(predictions,My_Random_Forest.test_df.label)
35 print(accuracy)
```

In [ ]:

```
'''
Find and visualize the most important 5 features for the prediction model
'''
from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('dataclustered.csv')
data=df.drop(['Unnamed: 0','id','comment_num','repost_num','like_num','type','created_at'
bins = [-1, 50, 100, 1000, 136265]
df['comment_num'] = pd.cut(df['comment_num'], bins,labels=[1,2,3,4],right=True)
X = data
bins = [-1, 100, 500, 1000,  565454]
df['repost_num'] = pd.cut(df['repost_num'], bins,labels=[1,2,3,4],right=True)
bins = [-1,75 , 500, 1000, 306161]
df['like_num'] = pd.cut(df['like_num'], bins,labels=[1,2,3,4],right=True)
print(X)
#y = df['comment_num']
#y = df['repost_num']
#y = df['like_num']

# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=100,
                              random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

df = pd.read_csv('dataclustered.csv')
df = df.drop(['Unnamed: 0','id'],axis=1)
df[df['type']==9].describe()

# Plot the feature importances of the forest
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.collections import LineCollection

%matplotlib inline
map_vir = cm.get_cmap(name='Greens')
n=5
norm = plt.Normalize(importances[indices][:n].min(), importances[indices][:n].max())
norm_y = norm(importances[indices][:n])
color = map_vir(norm_y)
plt.figure()
plt.title("Feature importances for like number")
```

```
#plt.title("Feature importances for comment number")
#plt.title("Feature importances for repost number")

plt.bar(range(5), importances[indices][:5],
       color=color, yerr=std[indices][:5], align="center")
#plt.xticks(range(5), ['fans','tweets','hash','at','vip'])
#plt.xticks(range(5), ['fans','tweets','2013','hash','2012'])
#plt.xticks(range(5), ['fans','tweets','hash','at','vip'])
#plt.xticks(range(5), indices)
plt.xlim([-1, 5])
plt.show();
```