# PERSONALITY BASED RECOMMENDER SYSTEM

## COEN 281- Pattern Recognition and Data Mining

## Santa Clara University

**Shail Shah**

**Bhargav Maniyar**

**Pinak Ghate**

**Dhaval Pujara**

**Harshil Shah**

# ACKNOWLEDGEMENT

The success of any project depends on contribution of team and guidance of others. We take this as an opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project, especially Dr. Ming-Hwa Wang, for his guidance throughout the term.

# Abstract:

Social media is a place where users present themselves to the world, revealing personal details and insights into their lives. We are beginning to understand how some of this information can be utilized to improve the users' experiences with interfaces and with one another. In this paper, we are interested in the personality of users. Personality has been shown to be relevant to many types of interactions; it has been shown to be useful in predicting job satisfaction, professional and romantic relationship success, and even preference for different interfaces. Until now, to accurately gauge users' personalities, they needed to take a personality test. This made it impractical to use personality analysis in many social media domains. In this paper, we present a method by which a user's personality can be accurately predicted through the publicly available information on their Facebook profile. We will describe the type of data collected, our methods of analysis, and the results of predicting personality traits through machine learning. We then discuss the implications this has for social media design, interface design, and broader domains.

## Table of Contents

**9. Appendices**

# 1. Introduction:

Introduction Social networking on the web has grown dramatically over the last decade. In January 2005, a survey of social networking websites estimated that among all sites on the web there were roughly 115 million members. Just over five years later, Facebook alone has exceeded 500 million members. In the process of creating social networking profiles, users reveal a lot about themselves both in what they share and how they say it. Through self-description, status updates, photos, and interests, much of a user's personality comes out through their profile.

This paper attempts to bridge the gap between social media and personality research by using the information people reveal in their online profiles. Our core research question asks whether social media profiles can predict personality traits. If so, then there is an opportunity to integrate the many results on the implications of personality factors and behavior into the users' online experiences and to use social media profiles as a source of information to better understand individuals. For example, the friend suggestion system could be tailored to a user based on whether they are more introverted or extroverted.

Previous work has shown that the information in users' Facebook profiles is reflective of their actual personalities, not an "idealized" version of themselves. This, plus a broad user base makes Facebook an ideal platform for studying this connection.

We administered the Big Five Personality Inventory to 279 subjects through a Facebook application. In the process, we gathered all the public data from their Facebook profiles. This was aggregated, quantified, and passed through a text analysis tool to obtain a feature set. Using these statistics describing the Facebook profile of each user, we were able to develop a model that can predict personality on each of the five personality factors to within 11% of the actual values.

The ability to predict personality has implications in many areas. Existing research has shown connections between personality traits and success in both professional and personal relationships. Social media tools that seek to support these relationships could benefit from personality insights. Additionally, previous work on personality and interfaces showed that users are more receptive to and have greater trust in interfaces and information that is presented from the perspective of their own personality features (i.e.

introverts prefer messages presented from an introvert's perspective). If a user's personality can be predicted from their social media profile, online marketing and applications can use this to personalize their message and its presentation.

We begin by presenting background on the Big Five Personality index and related work on personality and social media. We then present our experimental setup and methods for analyzing and quantifying Facebook profile information. To understand the relationship between personality and social media profiles, we present results on correlations between each profile feature and personality factor. Based on this, we describe the machine learning techniques used for classification and show how we achieve large and significant improvements over baseline classification on each personality factor. We conclude with a discussion of the implications that this work has for social media websites and for organizations that may utilize social media to better understand the people with whom they interact.

## 1.1. Objective

Based on one's personality he/she may have interest in mingling with other personalities or may be with similar kind of personality so it varies from person to person. So, instead of considering only location or gender there are many other parameters that play important role.So the objective is to build a system to recommend or suggest a person to get connected with based on following:

1. Based on person's personality
2. Based on person's interest in getting mingle with other personality

To give user a perfect suggestion / recommendation of user profiles that he /she will surely like and would be a great fit for him/her.

## 1.2. What is the problem

The problem is to suggest a user profile of his/her interest that he would like or will interest him instead of randomly picking up profiles according to the location of the user and the users near him, or through the ELO score i.e rating his profile through an separate individual algorithm and not letting the users know their ELO score and just suggesting users that have a similar ELO score. In, which case the users can edit their profile in a

specific way so that their ELO score increases and they get matches according to that.

### 1.3. Why is this a project related to this class

In class we studied concepts like text mining, finding similarities among text with various algorithms. Our idea / project is to mine the data that is available on the social networking platforms and mine information from that data to try and find personality of the users. Using that derived information we will be able to suggest users having similar personality as him/her.

### 1.4. Why other approaches are not good

The currently present approaches present in the industry mainly use location based searching that happen in real time. They track the users when they open the application and use the user's location of where they are and the person near them, if they crossed paths recently or they are in a particular mile radius which again is determined by the company itself. User has no control whatsoever who he will see or the choices and the pool of users who he will be choosing from. He just sees the top user near his location.

Also, as the application tracks the user using his real time location the people can easily spoof their current location using a location spoofer and

can make the application believe that they are somewhere else when they really aren’t. The other flaw in this system is that if a user switched its GPS location in his phone off the application won’t be able to suggest any user.

Other approach that is used. Is to give a specific score to the user according to his/her profile. The algorithm gives a particular score that is also called as an ELO score to calculate attractiveness of the user. But, that score can be easily manipulated by modifying your profile to a better liking, completing every dialog box in the profile description. You will never be shown your ELO score and you can only see the profile of users who have a similar ELO score like yours for example if you’re a 7, you won’t be shown to 4’s but neither will you be shown to a 9.

## 1.5. Why our approach is better

As mentioned above the flaws in the present approach. In our approach we would not matter what location you are in and would not suggest users to match with other users on the basis of the location of the user but would suggest one on the basis of the personality of a user that would be predicted and thus would have a higher similarity ratio then what is currently present. Also, it wouldn't matter if a user is in this particular location and if it is in other as he would get the choices based on his personality that would be mined through his social media handles and as he likes or dislikes a user his list would be constantly updated. Thus, increasing the chance of getting a good match.

## 1.6. Statement of the problem

There are two problems in the current system of prediction:-

1) First being that the users only get matched with other users based on their real time location of where they are.
2) The users gets matches according to their ELO score which can be easily altered by making some specific changes to their profile.

## 1.7. Area or scope of investigation

The area of study or the scope of investigation mainly remains on the personality that we can derive from the data available to us that we will get from the different social media handles of the user and how the data relates to the user. If there are more relation of the user with it's profile from example: posts, photos, status updates, location updates. Thus, all the information matters when predicting the personality of the user.

Also, the other area of study will be the methods, algorithms used to clean the data that will be derived from the various handles and how we approach it.

Different machine learning algorithms will also be out area of interest as we are hoping to divide the data derived by personality into 5 different vectors also called as 'The big five model'. Thus, then comparing the vectors of two different users using cosine similarity. We would also have to study the different ML algorithms.

# 2. Theoretical bases and literature review

## 2.1. Definition of the problem

- Problem definition is  divided into two parts :
    - Predicting personality
    - Suggesting user profile on basis of their personality.

## 2.2. Theoretical background of the problem

We can say that what we post,upload and update on our social media platforms is an extended identity of our-self in the digital world. It is how we want to represent our self in a digital media. Thus it can be derived that what we portrait our-self in the digital world is an extended personality of our-self and can be related to us. Thus, proving that one's digital personality in the digital world is actually one's true personality in the real world. So, it can be said that what we get from the virtual world will help us to better understand the user and his behaviour/personality in day to day life. Most of the online suggestion or recommendation of user profile is either random or based on location/gender. Data set of status/tweets will be used to solve the problem.The  standard "Big Five model" in field of psychology is used to access personality. Every human possess five

personality traits with different magnitude. Presently the most used model is ‘NEO Personality Inventory Revised’ (NEO PI-R). The model suggests that any users personality can be divided and mapped into five characteristics. We will get the data from the users profile picture, tweets, status updates, location, images, tagged photos and many more and will try to derive the personality of the user into the five characteristics like Extraversion, Neuroticism, Agreeableness, Conscientiousness and Openness. All of these 5 characteristics display different personality of the human behaviour and based on how they interact with their social media. Below is the description of the 5 factors:

• Extraversion: active, friendly, talkative, energetic.

• Neuroticism: moody, anxious, hostile, irritable.

• Agreeableness: trusting, altruistic, sympathetic, warm.

• Conscientiousness: ambitious, efficient, organized.

• Openness: imaginative, curious, enthusiastic, idealistic.

Based on the magnitude of above personality we will get a personality vector. Based on the vector we can find similarity to make suggestion for a user / personality and to learn more or to give more suggestion that a person is interested we can use its liked / disliked history and can suggest a perfect match for him / her.

## 2.3. Related research to solve the problem

For predicting personality, different approaches can be used like predicting or inferring personality from status or tweets (text mining), from profile pictures, from network ( user connections). Poria et al. to exploit the features related to psycho-linguistics, emotive words and their frequencies analyzed at lexical level by utilizing popular dictionaries such as Linguistic Inquiry and Word Count (LIWC) and Medical Research Council (MRC) to infer personality, apart from it SenticNet, EmoSenticNet and ConceptNet to derive / infer features based on emotions.

## 2.4. Advantage/Disadvantage of those research

The Recommendation systems which currently exist make the recommendations based on the user's preferences. In our literature survey, there are 3 approaches for recommendation: Content based filtering, collaborative filtering and hybrid approach.

Content based approach involves creating a feature vector of items and users, calculating their similarities and use these similarities to make recommendations. Collaborative filtering approach uses past preferences of users to decide which items to recommend. Hybrid methods combine these approaches to make recommendations.

- **It benefits from large user bases.** Simply put, the more people are using the service, the better your recommendations will become, without doing additional development work or relying on subject area expertise.
    - **It's flexible across different domains.** Collaborative filtering approaches are well suited to highly diverse sets of items. Where content-based filters rely on metadata, collaborative filtering is based on real-life activity, allowing it to make connections between seemingly disparate items (like say, an outboard motor and a fishing rod) that nonetheless might be relevant to some set of users (in this case, people who like to fish).
    - **It produces more serendipitous recommendations.** When it comes to recommendations, accuracy isn't always the highest priority. Content-based filtering approaches tend to show users items that are very similar to items they've already liked, which can lead to filter bubble problems. By contrast, most users have interests that span different subsets, which in theory can result in more diverse (and interesting) recommendations.

- **It can capture more nuance around items.** Even a highly detailed content-based filtering system will only capture some of the features of a given item. By relying on actual human experience, collaborative filtering can sometimes recommend items that have a greater affinity with one another than a strict comparison of their attributes would suggest. Compared to collaborative filtering, there are some advantages and drawbacks of content-based filtering that we should understand.

The following are the advantages and the disadvantages of the research done:

**Advantages:-**

- The advantages being almost all the necessary data needed to predict the personality of the user is available through the APIs or on other websites.
- Also, because of the recent leaks(cambridge analytica) the data is easily available.
- User independence: collaborative filtering needs other users' rating to find the similarity between the users and then give the suggestion. Instead, content-based method

only have to analyze the items and user profile for recommendation.

- Transparency: collaborative method gives you the recommendation because some unknown users have the same taste like you, but content-based method can tell you they recommend you the items based on what features.
- No cold start: opposite to collaborative filtering, new items can be suggested before being rated by a substantial number of users.

**Disadvantages:-**

- The disadvantage being to predict the data using all the user data will not be easily done and also many of the user data would need to be cleaned. There may be many anomalies in the data like missing information, fake informations.
- Also, many times it happens that as a user progresses through the different years in his life cycle, example: child, teen, adult. His personality changes thus, if we

have a user data available when he was a teen it may not be relevant when he becomes an adult and so on.

- Limited content analysis: if the content does not contain enough information to discriminate the items precisely, the recommendation will be not precisely at the end.
- Over-specialization: content-based method provides a limit degree of novelty, since it has to match up the features of profile and items. A totally perfect content-based filtering may suggest nothing "surprised."
- New user: when there's not enough information to build a solid profile for a user, the recommendation could not be provided correctly.
- We would need the latest data from each user to correctly predict his personality.

## 2.5. Your solution to solve the problem

Our solution to the problem is to suggest users who have similar personality and thus increasing the percentage of getting matched. For the system to successfully recommend a user based on his/her personality it would have to have data of the user to map to it can be done by collecting all the personality specific data that would be available from his/her social media handles. Thus, the system can start creating a personality of the user and thus map the users likes and dislikes.

After, that is done the system will randomly show people for a while and would start learning from the users likes and dislikes. So, it can map what the user likes and thus can start suggesting users who have the same characteristics in their personality. Thus, increasing the chance of getting a perfect match. As the systems would start learning from the users choices using machine learning algorithms like lstm, seq2seq to start differentiating the users what the user would like and what he would not like. Thus, mapping a users personality properly and also his/her choices would give us a better result of suggesting a user on which he is likely to like it.

## 2.6. Where our solution is different from others

The difference in the solutions from others is that currently every dating app uses location based matching to match users from one other the only difference being one gives more privacy control to one user than the other. Tinder uses your location to send you profiles of people in your area who fit the criteria you're looking for (specifically, age and gender). If you and the other person swipe right, a little screen pops up informing you there's a match (it's a thrilling moment), and you can introduce yourself. Bumble works largely the same way, with one major difference: on Bumble, only the woman can make the first move, which gives women more say in the dating process. It's why women say they feel safer on.

The difference in solution here being we would match the users based on their personality being similar or not and not just randomly suggesting users based on their location. So, by predicting the personality through their social media we would have a clear profile of a user of what he likes and what he doesn't.

Thus, we would be able to give a user a better prediction if who he is supposed to match and who rather than who he isn't. Also, after suggesting

a couple of random users at first to learn a user's needs we can now provide him with  a more realistic match then at the first as we would be able to learn from the users action on what he likes and dislikes. Thus, this are the basic difference between the different solutions.

## 2.7. Why our solution is better

Our solution is better than the one that is in use because of the following reasons:-

Our solution is to give the recommendations based on the personality of the users and thus increases the chance of getting a favourable match as it will recommend it using your personality. It will be able to do so by using the data you have given in your social media handles photos, posts, location, updates, statuses. All of this data contribute to your personality and will help us improve the match that you will get.

After, knowing your personality through your social media handles the system will try to learn what are your likes and dislikes when you are looking for the other user. It will do that by first suggesting some random users and will learn on the basis of that what are your likes and dislikes for example: you dislike a specific user because they show that they love cats.

The system will read that information and and will try to provide suggestions on the basis of that. All in all increasing the probability of suggesting a user that you are likely to be matched with the more often you use the application the more likely it is that the system learns about your likes and dislikes and help you in pairing with a perfect match.

Thus, these are the ways are solution is better than the ones currently in use.

# 3. Hypothesis

What we're trying to achieve here is that a user gets smart recommendations for the profile based on his personality vector.

In this project we plan to implement the models, SVM and word2vec to evaluate them to understand what factors are most predictive for finding profiles that a user would like using features set below.

We implemented Supervised Machine Learning models, the model takes a profile personality vector as input and will output the profiles according to the preferences of the user.

# 4. Methodology

## 4.1. How to generate/collect input data

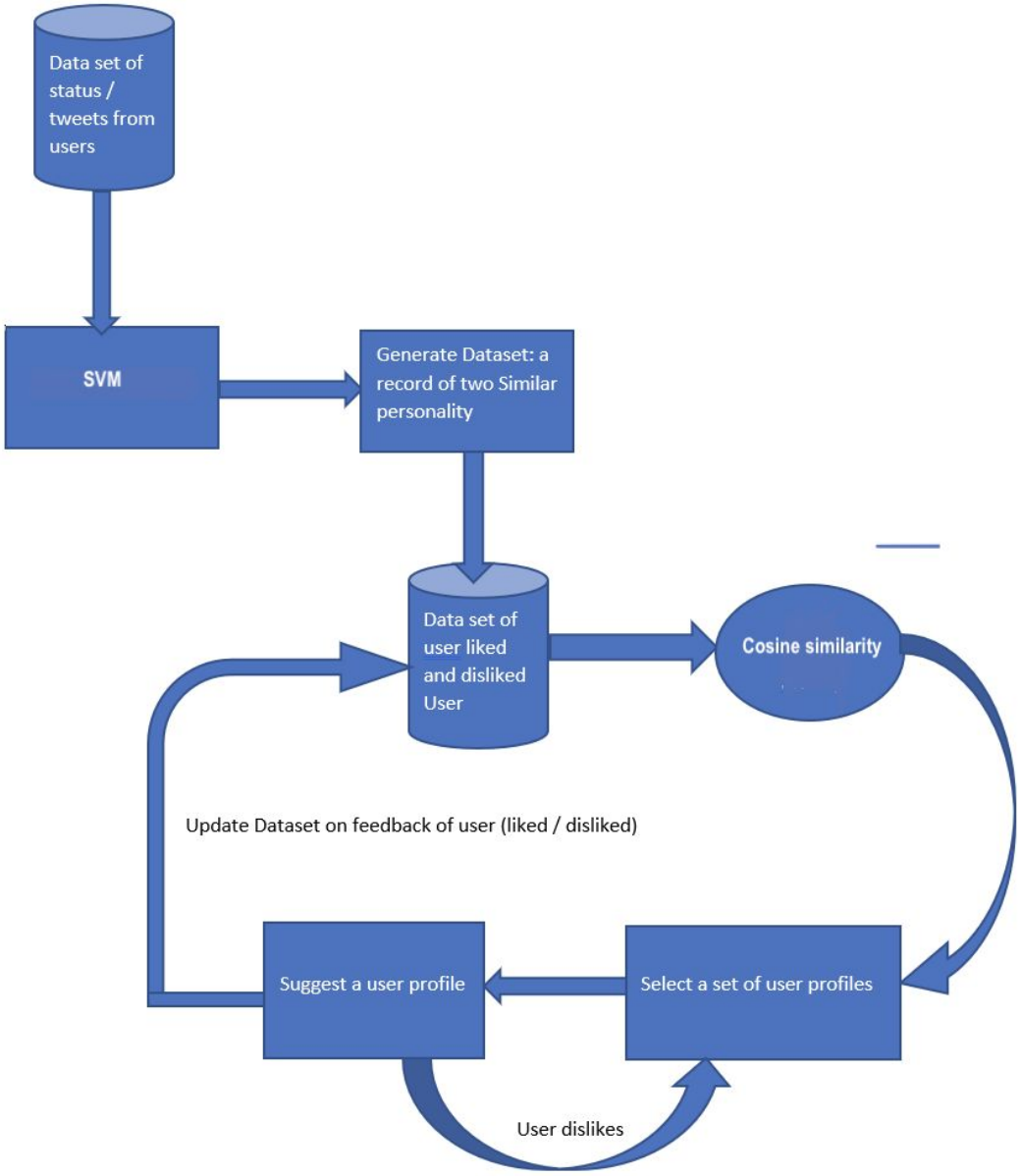
The data that we need to collect from the user about the information about his/her personality is available through an .csv file on sites like kaggle because of the research done on the user data on  a different topics. The data is of the facebook users collected through a survey and has many fields like status updates, location information, posts and much more. Though the data is available, the data is not properly cleaned or parsed and needs to be cleaned in order to get the necessary fields from the available data.

We would also have to generate some random user data in order to know display to the user at the beginning from which we will get to know about the likes and dislikes of the user from which we will get to know the personality of the user and the users with a following personality trait that he would be interested in. Thus, to get know this we would have to give the users a sample amount of profiles to choose from having different personality traits and this will help us knowing the user better.

These, are the following types of data that we need to generate/collect.

## 4.2. How to solve the problem

The following flow diagram will give the process in a nutshell.



### 4.2.1. Algorithm design

We can divide algorithm design into 2 phase :

1. Generate Personality Vector
2. Generate suggestion for User

## 1. Generate Personality Vector

**Method 1 :**

1. Iterate steps for every user

a. Count unique word and add frequency for each user.

b. Use Textblob to get sentimental analysis

c. Use SVM for text categorization

d. Add personality vector with Gender to ‘Personality DataSet’

2. Generate a pair of similar personality vector data set (Similarity Data Set) with similarity above threshold.

**Method 2:**

Word2vec:

Word2Vec is an effective solution to convert text into vector format, which leverages the context of the target words. Basically, we want to use the surrounding words to represent the target words with a

Neural Network whose hidden layer encodes the word representation.

There are two types of algorithms for Word2Vec, Skip-gram and Continuous Bag of Words (CBOW).

## Method 3

### 1. SVM (Support Vector Machine)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

We have 10 different personalities and to predict it we have used different kernel / types of SVM classifiers such as ‘linear’ , ‘polynomial’ and ‘radial’. Out of all 3 linear gave the best output as there was clear partition among data as shown in fig 3. But as we have to predict one class out of 10. SVM will use one to other classification for each of the class and allocated / predict highest scoring class as a suitable value.

If we plot SVM for our data, it will be 10 partitions of arbitrary shape and there will be a almost clear separation among the partitions.

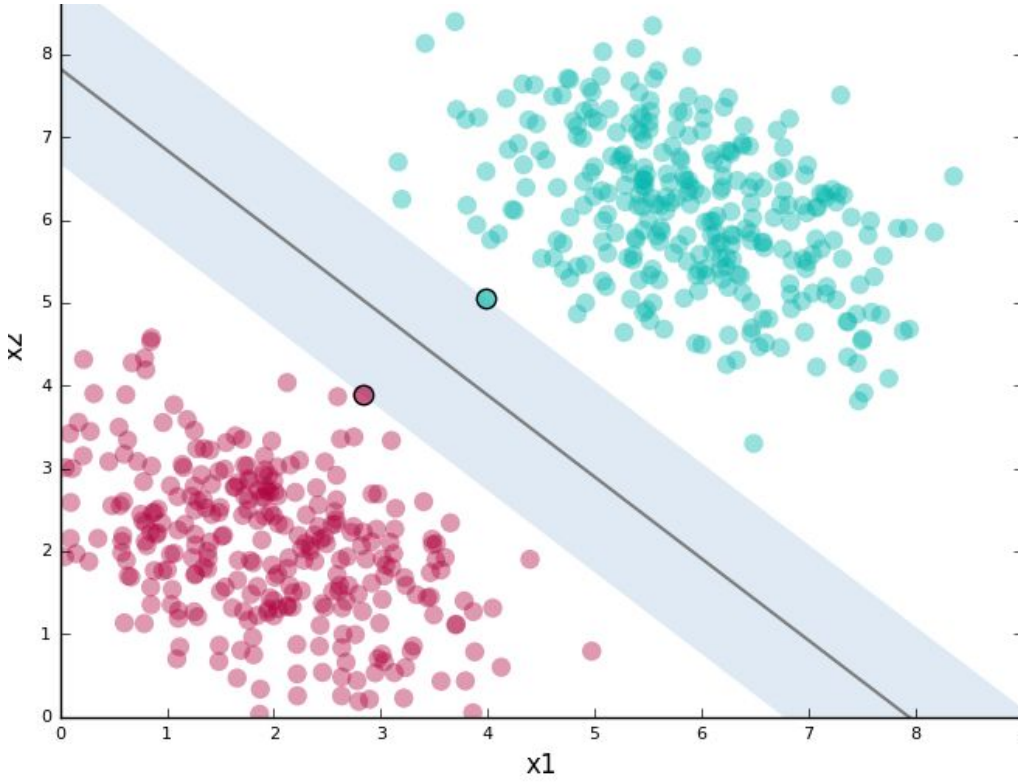
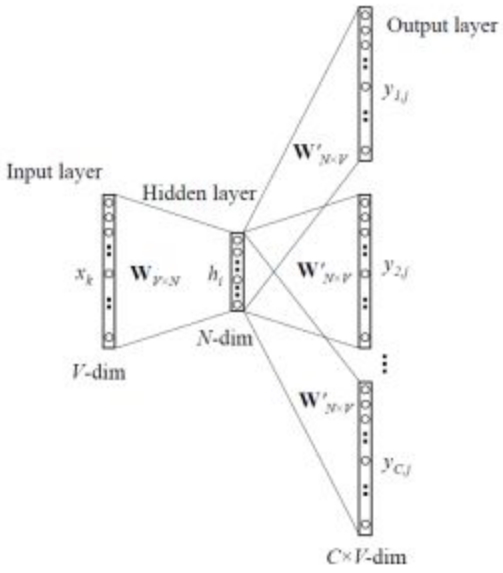



Fig. 3

**Skip-gram**

For skip-gram model, the input is the target word, while the outputs are the words surrounding the target words. All the input and output data are of the same dimension and one-hot encoded. The network contains 1 hidden layer whose dimension is equal to the embedding size, which is smaller

than the input/ output vector size. At the end of the output layer, a softmax activation function is applied so that each element of the output vector describes how likely a specific word will appear in the context. The graph below visualizes the network structure.

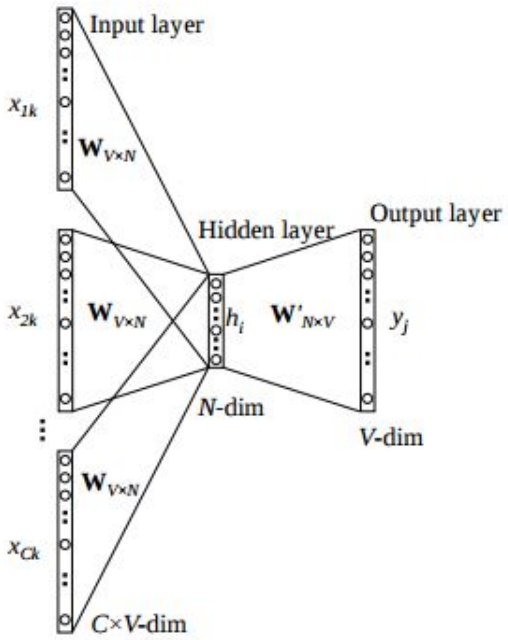



The word embedding for the target words can obtained by extracting the hidden layers after feeding the one-hot representation of that word into the network.

With skip-gram, the representation dimension decreases from the vocabulary size (V) to the length of the hidden layer (N).

**CBOW**

Continuous Bag of Words (CBOW) is very similar to the skip-gram model. The idea is that given a context, we want to know which word is most likely to appear in it.




The biggest difference between Skip-gram and CBOW is that the way the word vectors are generated. For CBOW, all the examples with the target word as target are fed into the networks, and taking the average of the extracted hidden layer. Skip-gram only feeds in the one and only one target word one-hot vector as input.

## 2. Generate Suggestion for User using LSTM

### Method 1:

1. From Similarity Data-Set, find a generic similar vector (sim-vector).
2. Find set of vectors from ‘Personality data set’ that are similar to (sim-vector) using LSTM.

### Method 2:

1. Find cosine similarity for a user and some set of other users as of now and later we can extend to find similarity that have opposite sex / in same area / city
2. Each user has minimum threshold value that he would like other person’s personality for eg. user 1 with threshold 20% will like all other the users/ profiles that have 20% similarity with his personality
3. Feed user with matching profile and update the database on its action like ‘Y’ or ‘N’ ( He liked it or not )

## Cosine Similarity

Cosine similarity is a measure of similarity between two

non-zero vectors of an inner product space that measures cosine of the angle between them.

For vectors A and B,the cosine similarity, is represented by dot product and magnitude as below:

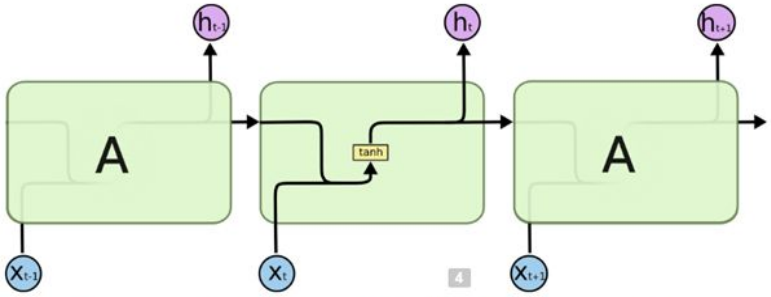
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$ , where $A_i$ and $B_i$ are components of vector $A$ and $B$ respectively.

So we find cosine similarity matrix between n*n users and use upper triangular matrix to compare with threshold and will sort the output in ascending order to get the most likely profile to be liked by user at first and will show him/her with its name and percentage match.

**LSTM Networks**

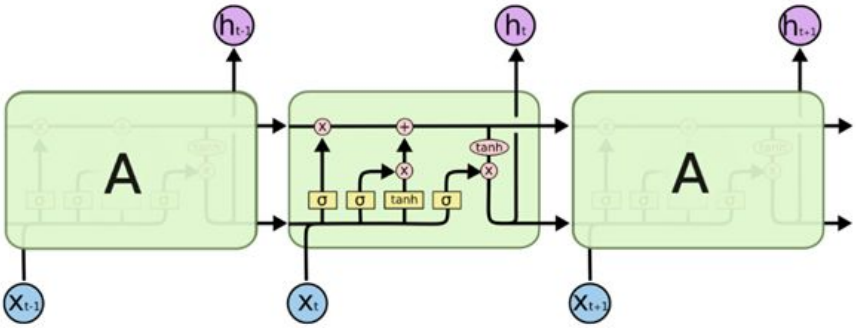
Long Short-Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tan(h) layer.



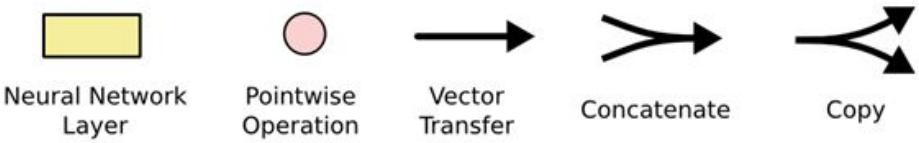

The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.




The repeating module in an LSTM contains four interacting layers.

The notions of LSTMs are as below.




In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

## Method: 4

### Random Forest

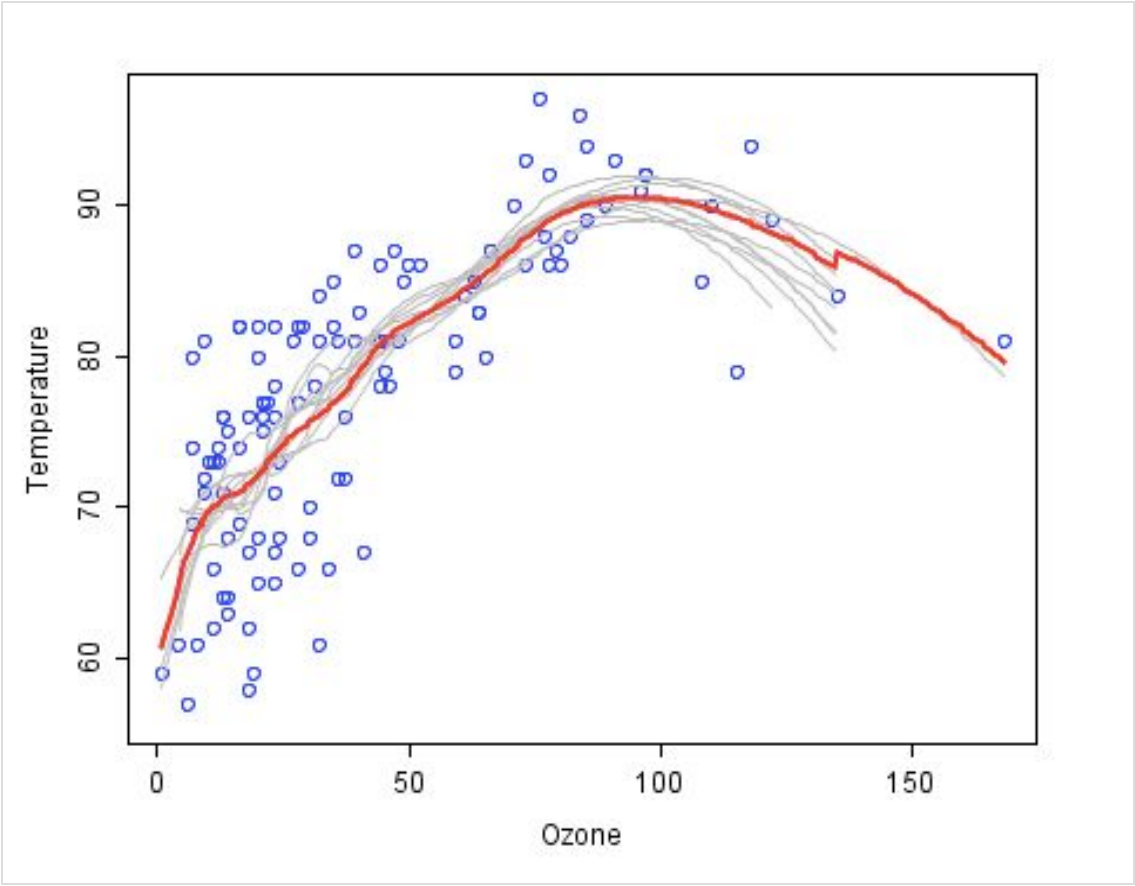
The random forest is an ensemble approach that can also be thought of as a form of nearest neighbor predictor.It works as an  divide-and-conquer approach used to improve performance. The main principle behind this method is that a group of weak learners can come together to form a strong learner. The figure below provides an example. Each classifier, individually, is a weak learner, while all the classifiers taken together are a strong learner.

The data to be modeled are the blue circles. We assume that they represent some underlying function plus noise. Each individual learner is shown as a gray curve. Each gray curve which is a weak learner is a fair approximation to the underlying data. The red curve which here represents as an strong learner can be seen to be a much better approximation to the underlying data.

Trees and Forests. The random forest starts with a standard machine learning technique called a decision tree which, in ensemble terms, is our

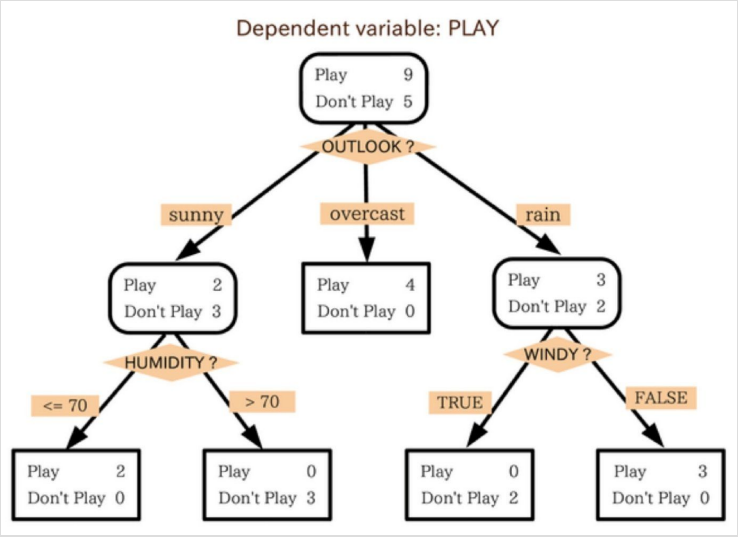
weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets.

Other example of this model,where the data is driven and made into trees it advises us, based u



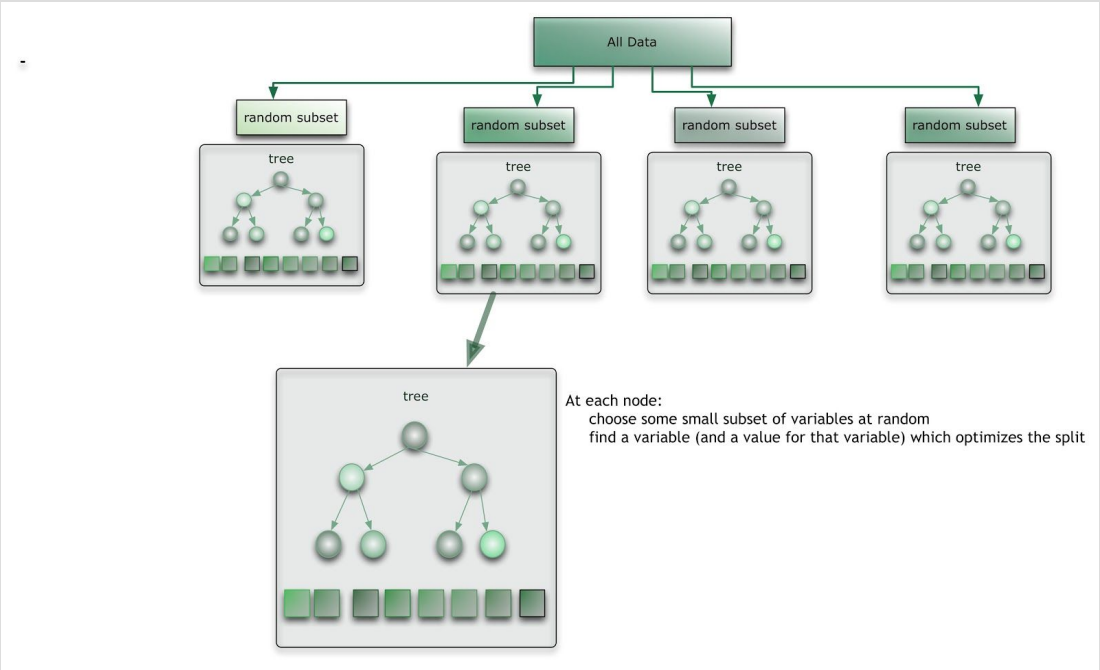

pon weather conditions, whether to play football. For example, if the outlook is

sunny and the humidity is less than or equal to 70, then it's probably of to play football.




The random forest (see figure below) takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms,

the trees are weak learners and the random forest is a strong learner.




Here is how such a system is trained; for some number of trees *T*:

1. Sample *N* cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
2. At each node:
    1. For some number m, predictor variables are selected at random from all the predictor variables.
    2. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.

3. At the next node, choose another m variables at random from all predictor variables and do the same.

Depending upon the value of m, there are three different possibilities

- Random splitter selection: m =1
- Breiman's bagger: m = total number of predictor variables
- Random forest: m << number of predictor variables. The alootihtam suggests three possible values for m: ½$\sqrt{m}$, $\sqrt{m}$, and 2$\sqrt{m}$.

Running a Random Forest. When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority.

It should be taken into thought that:

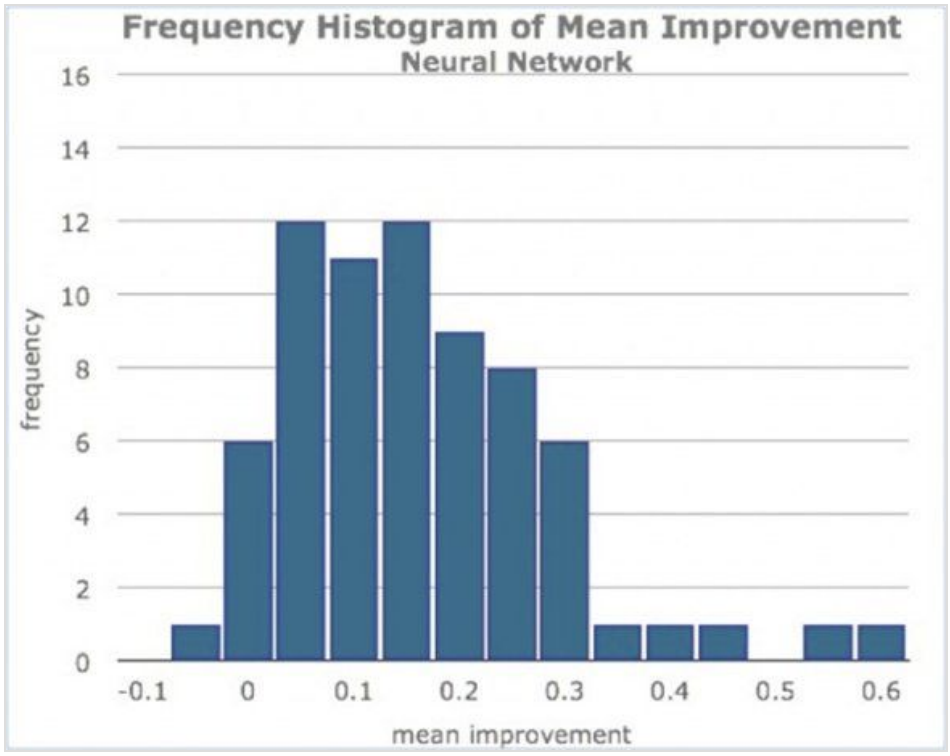
- With a large number of predictors, the eligible predictor set will be quite different from node to node.
- The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.

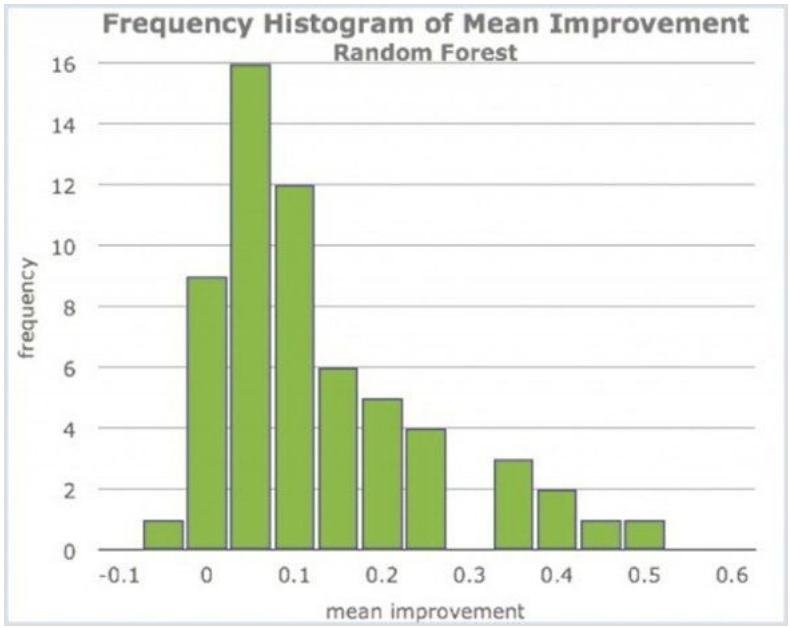
- As *m* goes down, both inter-tree correlation and the strength of individual trees go down. So some optimal value of *m* must be discovered.

Positives and negatives: Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. Random Forest weaknesses are that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy. Of course, the best test of any algorithm is how well it works upon your own data set.

Neither method can be said to be better than the other in all cases. It remains to be seen if there is any systematicity as to why and where one method is better than another.

This figure shows a frequency histogram of the mean precision improvement over chance for the 72 projects for the random forest:


Frequency Histogram of Mean Improvement
Neural Network
16
14
12
10
8
6
4
2
0
frequency
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
mean improvement


Frequency Histogram of Mean Improvement
Random Forest
16
14
12
10
8
6
4
2
0
frequency
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
mean improvement

## Method 5:

### Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

In logistic regression, the dependent variable is binary or i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).

The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the binary characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit transformation* of the probability of presence of the characteristic of interest:

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k$$

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values.

Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Sometimes logistic regressions are difficult to interpret;

Example:

How does the probability of getting lung cancer (yes vs. no) change for every additional pound a person is overweight and for every pack of cigarettes smoked per day?

Do body weight, calorie intake, fat intake, and age have an influence on the probability of having a heart attack (yes vs. no)?

**Binary logistic regression major assumptions:**

1. The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
2. There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
3. There should be no high correlations (multicollinearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Tabachnick and Fidell (2013) suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.

At the center of the logistic regression analysis is the task estimating the log odds of an event. Mathematically, logistic regression estimates a multiple linear regression function defined as:

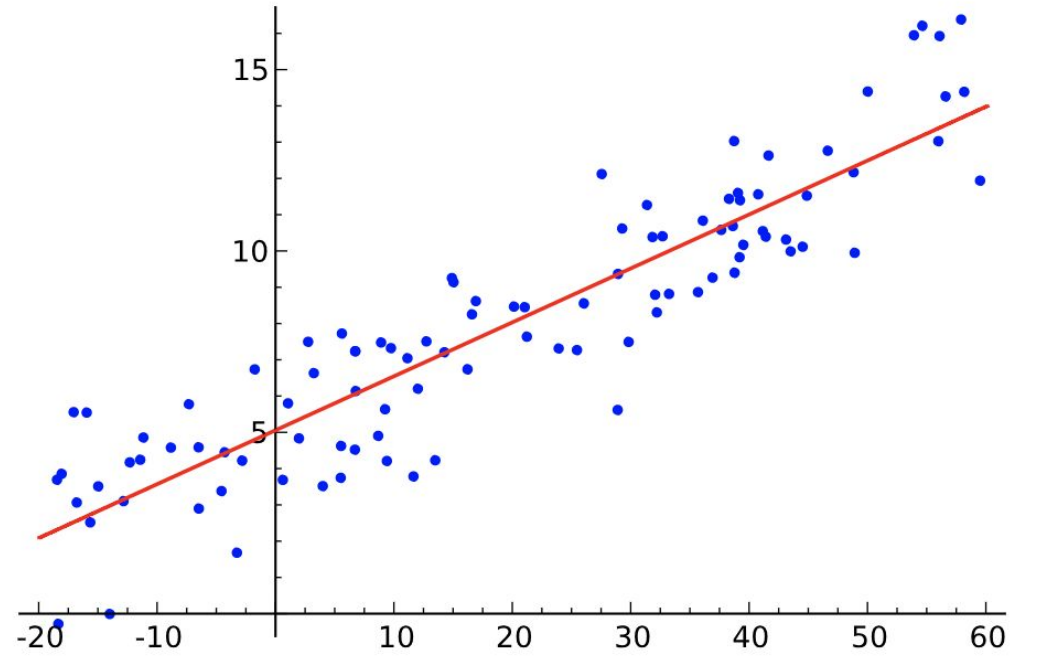
$$= \log\left(\frac{p(y=1)}{1-(p=1)}\right) = \beta_0 + \beta_1 x_{i2} + \beta_2 \cdot x_{i2} + \_ + \beta_p \cdot x_{in}$$

logit(p)

for i = 1…n .

**Overfitting.** When selecting the model for the logistic regression analysis, another important consideration is the model fit. Adding independent variables to a logistic regression model will always increase the amount of variance explained in the log odds (typically expressed as $R^2$). However, adding more and more variables to the model can result in overfitting, which reduces the generalizability of the model beyond the data on which the model is fit.

**Reporting the R2**. Numerous pseudo-R2 values have been developed for binary logistic regression. These should be interpreted with extreme caution as they have many computational issues which cause them to be artificially high or low. A better approach is to present any of the goodness of fit tests available; Hosmer-Lemeshow is a commonly used measure of goodness of fit based on the Chi-square test.


15
10
5
-20
-10
10
20
30
40
50
60

## Method 6:

## KNN

The *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric and instance based method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression.

When we say a technique is **non-parametric** , it means that it does not make any assumptions on the underlying data distribution. In other words, the model structure is determined from the data. If you think about it, it's pretty useful, because in the "real world", most of the data does not obey the typical theoretical assumptions made (as in linear regression models, for example). Therefore, KNN could and probably should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution data.

**Instance-based** learning means that our algorithm doesn't explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as "knowledge" for the prediction phase. Concretely, this means that only when a query to our database is made (i.e. when we

ask it to predict a label given an input), will the algorithm use the training instances to spit out an answer.

It is worth noting that the minimal training phase of KNN comes both at a *memory cost*, since we must store a potentially huge data set, as well as a *computational cost* during test time since classifying a given observation requires a run down of the whole data set. Practically speaking, this is undesirable since we usually want fast responses

### 4.2.2. Language used

1. Python 3.6

### 4.2.3. Tools used

1. Scikit-Learn
2. Hadoop 2.9.0
3. Map Reduce
4. Anaconda 3 with Jupyter
5. Numpy and Pandas
6. Textblob
7. Seaborn
8. Matplotlib
9. NLTK toolkit

# 5. Implementation

## 5.1 Code

```
# coding: utf-8
# <img src="TwitterData/flow.png"/>
# ### Read the input data files.
# In[1]:
import numpy as np
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
import sklearn.metrics.pairwise as sk
from textblob import TextBlob
# nltk.download()
from nltk.corpus import stopwords
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer

# In[4]:
#Read the tweets one by one and process it
import csv
user_id=[]
inpTweets = csv.reader(open('survey_dump_with_tweet_count',
'rt',encoding='utf8'), delimiter=',')
i = 0
```

```
for row in inpTweets:
    i+=1;
    if(i>1):
        user_id.append(row[0])
    if('1663416536' in row):
        print(i)
print(i);
# ### Pre-process Tweets

# In[6]:
class PreprocessTweets:

    def __init__(self):
        self.name = 'PreprocessTweets'

    #start process_tweet
    def processTweet(self, tweet):

        #Convert to lower case
        tweet = tweet.lower()
        #Convert www.* or https?://* to URL
        tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
        #Convert @username to AT_USER
        tweet = re.sub('@[^\s]+','AT_USER',tweet)
        #Remove additional white spaces
        tweet = re.sub('[\s]+', ' ', tweet)
        #Replace #word with word
        tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
        #trim
        tweet = tweet.strip('\'"')
        # Remove all Non-ASCII characters
        tweet = re.sub(r'[^\x00-\x7F]+',' ', tweet)
```

```
        return tweet
# In[7]:
class FilterStopWords:

    # stopWords = []
    def __init__(self):
        self.name = 'FilterStopWords'
        #initialize stopWords
        self.stopWords = []


    def getStopWordList(self, stopWordListFileName):
        #read the stopwords file and build a list
        stopWords = []
        stopWords.append('AT_USER')
        stopWords.append('URL')
        stopWords.append('[')
        stopWords.append('[')

        fp = open(stopWordListFileName, 'r',encoding='utf8')
        line = fp.readline()
        while line:
            word = line.strip()
            stopWords.append(word)
            line = fp.readline()
        fp.close()
        return stopWords

    def getFeatureVector(self, tweet, stopWords):
        featureVector = []
        #split tweet into words
        words = tweet.split()
        for w in words:
```

```
            #replace two or more with two occurrences
            #w = replaceTwoOrMore(w)
            #strip punctuation
            w = w.strip('\'"?,.')
            #check if the word stats with an alphabet
            val = re.search(r"^[a-zA-Z][a-zA-Z0-9]*$", w)
            #ignore if it is a stop word
            if(w in self.stopWords or val is None):
                continue
            else:
                featureVector.append(w.lower())
        return featureVector
# ### Feature Engineering

# In[8]:
class FeatureEngineering:
    def __init__(self):
        self.name = 'FeatureEngineering'
        self.featureList = []
        # self.sid = SentimentIntensityAnalyzer()
    #start extract_features
    def extract_features(self,tweet):
        tweet_words = set(tweet)
        features = {}
        for word in self.featureList:
            features['contains(%s)' % word] = (word in tweet_words)
        return features
## Create New Training set based on personality labels predicted from
Survey results
    def createNewTrainingSet(self, fileName):
        XTrain = []
        YTrain = []
        XTrainFeatures = []
```

```
        XTrainSentiment = []
        XTrainFreqTweets = []
        geo_latitude = []
        geo_longitude = []

        objFilterStopWords = FilterStopWords()
        objPreprocessTweets = PreprocessTweets()

        stopWords =
objFilterStopWords.getStopWordList('TwitterData/StopWords.txt')

        #Read the tweets one by one and process it
        inpTweets = csv.reader(open(fileName, 'r',encoding='utf8'),
delimiter=',')
        next(inpTweets)
        tweets = []
        i = 0
        for row in inpTweets:
            personality = row[5]
            tweet = row[1]
            cleanTweet = tweet.replace("""","""," ")
            cleanTweet = cleanTweet.replace("""," ")
            processedTweet =
objPreprocessTweets.processTweet(cleanTweet)

            XTrainFreqTweets.append(int(row[4]))
            wordsList = processedTweet.split()

            # Remove stop words
            filtered_words = [word for word in wordsList if word not in
stopwords.words('english')]
            filteredTweets = ' '.join(filtered_words)
```

```
            featureVector =
objFilterStopWords.getFeatureVector(processedTweet, stopWords)

            geo_latitude.append(float(row[2]))
            geo_longitude.append(float(row[3]))

            blob = TextBlob(processedTweet)
            sentiment = 0
            for sentence in blob.sentences:
                sentiment += sentence.sentiment.polarity

            totSentiment = sentiment/ len(blob.sentences)

            XTrainSentiment.append(totSentiment)

            XTrainFeatures.append(filteredTweets)

            YTrain.append(personality)

        return XTrain, YTrain, XTrainFeatures, XTrainSentiment,
XTrainFreqTweets, geo_latitude, geo_longitude
# In[9]:
objFeatureEngineering = FeatureEngineering()
fileName = 'TwitterData/survey_dump_with_tweet_count'
XTrain, YTrain, XTrainFeatures, XTrainSentiment, XTrainFreqTweets,
geo_latitude, geo_longitude =
objFeatureEngineering.createNewTrainingSet(fileName)
# ### Get Feature vector
# In[10]:
newYTrain = []

for item in YTrain:
    temp = item.replace('[', '')
```

```
    temp = temp.replace('\"', '')
    newItem = temp.replace(']', '')
    newYTrain.append(newItem)

YTrain = newYTrain
# ### Map the class labels to numbers

# In[11]:
def mapLabels(className):
    if className == 'Conscientiousness':
        return 0
    elif className == 'Extrovert':
        return 1
    elif className == 'Agreeable':
        return 2
    elif className == 'Empathetic':
        return 3
    elif className == 'Novelty Seeking':
        return 4
    elif className == 'Perfectionist':
        return 5
    elif className == 'Rigid':
        return 6
    elif className == 'Impulsive':
        return 7
    elif className == 'Psychopath':
        return 8
    elif className == 'Obsessive':
        return 9
    #elif className == None:
        #return 10
    else:
        pass
```

```
YTrain = [mapLabels(x) for x in YTrain]
# In[12]:
XTrain = np.array(XTrainFeatures)
YTrain = np.array(YTrain)
# In[13]:
YTrain
# ### Split Train and Test data
# In[14]:
n=60
trainSamples = XTrain[0:n]
YtrainSamples = YTrain[0:n]

testSamples = XTrain[n:]
YtestSamples = YTrain[n:]

trainSentimentSamples = np.array(XTrainSentiment[0:n])
testSentimentSamples = np.array(XTrainSentiment[n:])
trainFreqTweetSamples = np.array(XTrainFreqTweets[0:n])
testFreqTweetSamples = np.array(XTrainFreqTweets[n:])
# ### Bag of Words as Features
# In[15]:
vectorizer = CountVectorizer()
XTr = vectorizer.fit_transform(trainSamples)

trainBagVector = XTr.toarray()
XTe = vectorizer.transform(testSamples)
testBagVector = XTe.toarray()



# In[16]:
XEv = XTe
# ### Stack or concatenate all features together
```

```
# In[17]:
XTrainWordFeatures = trainBagVector #trainNGramVector

temp = np.column_stack((XTrainWordFeatures, trainSentimentSamples))

XTrainAllFeatures =  np.column_stack((temp, trainFreqTweetSamples))

XTestWordFeatures = testBagVector #testNGramVector
temp =  np.column_stack((XTestWordFeatures, testSentimentSamples))

XTestAllFeatures =  np.column_stack((temp, testFreqTweetSamples))

# ### Write Predicted Output Labels to File
# In[18]:
def writePredictedLabelFile(YPred):
    f = open("Predictions.csv","w")
    f.write("Id,Label" + "\n")
    for i in range(len(YPred)):
        f.write(str(i) + "," + str(np.around(YPred[i],decimals=2))+ "\n")
    f.close()
# In[19]:
train = XTrainAllFeature
YTrain = YtrainSamples
YTest = YtestSamples
# In[20]:
from sklearn.neighbors import KNeighborsClassifier

train = XTrainAllFeatures
test = XTestAllFeatures

params = {'neighbours':10}
neighbours = params['neighbours']
neigh = KNeighborsClassifier(n_neighbors=neighbours)
```

```
YPredKNN = neigh.fit(train, YTrain).predict(test)
# In[21]:
from sklearn.ensemble import RandomForestClassifier
params = {'trees':150, 'criterion':'entropy','random_state':None}
trees = params['trees']
crit = params['criterion']
seed = params['random_state']
clf =
RandomForestClassifier(n_estimators=trees,criterion=crit,random_state=se
ed)
clf.fit(train, YTrain)
YPredRF = clf.predict(test)
# In[22]:
#LogReg = linear_model.LogisticRegression(solver = 'sag', multi_class =
'multinomial',penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None)
LogReg = linear_model.LogisticRegression()
penalty = ['l1', 'l2']
C = np.logspace(0, 4, 10)
hyperparameters = dict(C=C, penalty=penalty)
clf = GridSearchCV(LogReg, hyperparameters)

clf.fit(train, YTrain)
YPredLR = clf.predict(test)
# In[23]:
from sklearn import svm
params = {'kernel':'linear'}
ker = params['kernel']
YPred = svm.SVC(kernel=ker, probability = True).fit(train,
YTrain).decision_function(test)
# In[24]:
YPred  # [7, 8, 6, 4, 9, 7, 9, 6, 1, 6, 7, 2, 4, 2, 2, 7, 9, 7, 9, 7, 8, 6,
#9, 4, 8, 9, 1, 3, 9, 9, 8, 7, 7, 9, 9, 1, 1, 1, 8, 9, 7, 8, 1, 6,
```

```
 #      8, 5, 8, 6, 3, 1, 4, 9, 3, 9, 7, 7, 9, 9, 3, 7, 3, 9, 4, 8, 7, 1,
  #     9, 9, 1, 9, 1, 3, 3, 2, 3, 1, 1, 0, 4, 7, 8, 6, 4, 9, 7, 9, 6, 1,
   #    6, 7, 2, 4, 2, 2, 7, 9, 7, 9, 7, 8, 6, 9, 4, 8, 9, 1, 3, 9, 9, 8,
    #   7, 7, 9,9,
     #  1, 1, 1, 8, 9, 7, 8, 1, 6, 8,5, 8,
      # 6, 3, 1, 4, 9, 3, 9, 7, 7, 9, 9, 3, 7, 3, 9, 4, 8, 7, 1, 9, 9, 1,
       #9, 1, 3, 3, 2, 3, 1, 1, 0, 4]"'
# In[25]:
dfYPred=YPred
# In[26]:
df=pd.DataFrame(data=dfYPred)
# In[27]:
dfYPred
# In[28]:
pers=['Conscientiousness','Extrovert','Agreeable','Emphathetic','Nov
Seekng','Perfectionist','Rigid','Impulsive','Psychopath','Obsessive']
# In[29]:
df.columns=pers
# In[30]:
df['dom_pers']=df.idxmax(axis=1)
# In[31]:
df['thres']= np.random.randint(10, 100, df.shape[0])
# In[32]:
df['user_id']=user_id[n:]
# In[33]:
df.set_index('user_id')
df.head(10)
# In[34]:
def cosine_sim(df):
    for index, row in df.iterrows():
        print((row[1:10]))
# In[35]:
df1=pd.DataFrame(data=sk.cosine_similarity(YPred,YPred))
```

```
# In[36]:
df1=df1*100
# In[37]:
df1.columns=user_id[n:]
df1['user_id']=user_id[n:]
# In[38]:
#df_1=df1.loc[df1['user_id'] == '1516255956']
# In[39]:
#userid or input
userid='1414166594'
# In[40]:
df_d=df.loc[df['user_id'] == userid]

labels = tuple(pers)#'Extrovert', 'Agreeable', 'Empathetic','Novelty
Seeking','Perfectionist','Rigid','Impulsive','Psychopath','Obsessive'
sizes = df_d.iloc[0,:10]
explode = (0.2,0,0.2,0,0.2,0,0.2,0,0.2,0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
# In[41]:
df_d=df1.loc[df1['user_id'] == userid]
df_2=df_d[(df_d<100) & (df_d>df.loc[df['user_id'] == userid]['thres'].iloc[0])]
df_2=df_2[df_2!='NaN']
df_2.dropna(axis=1,inplace=True)
df_2=df_2.iloc[:,:-1]
df_2=df_2.transpose()
df_2.columns=[1]
df_2=df_2.sort_values(1)
# In[42]:
df_2.head()
```

```
# In[43]:
from IPython import display
def plotSimilarityGraph(userid2):

    user_personality_df=df.loc[(df['user_id'] == userid) |
(df['user_id']==userid2 )]

    del user_personality_df['thres']
    del user_personality_df['dom_pers']

    df_melt = user_personality_df.melt('user_id', var_name='Personalities',
value_name='Magnitudes')
    df_melt

    barG=sns.factorplot(data=df_melt,
kind='bar',hue='user_id',x='Personalities',y='Magnitudes',palette='Set1')
    barG.set_xticklabels(rotation=90)
    display.display(barG)
# In[45]:
for index,row in df_2.itertuples():
    plotSimilarityGraph(index)
    inp=input("Do you want another suggestion? (y/n): ")
    if(inp=='n'):
        break;
```

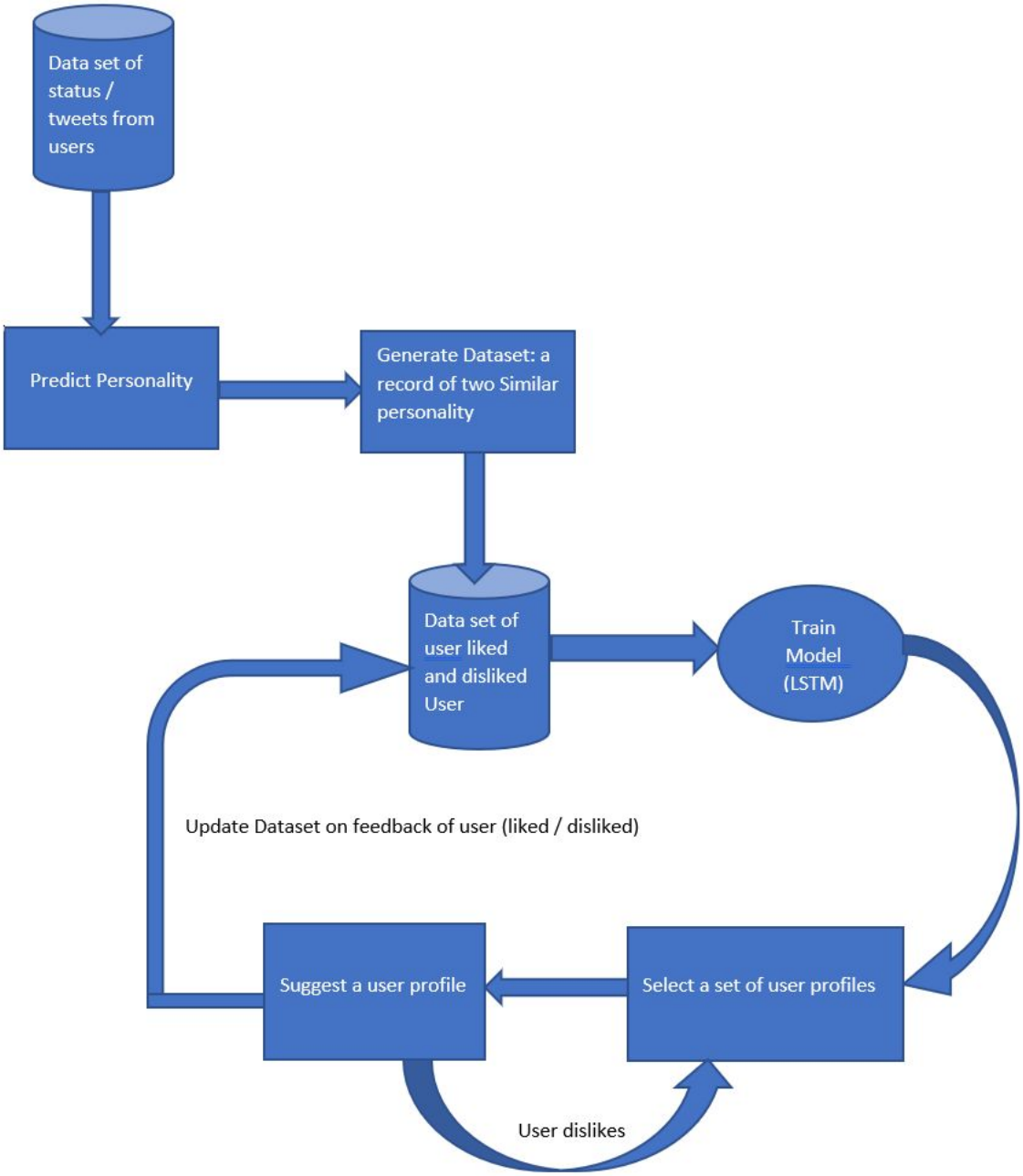
## Mapper.py

```
#!/usr/bin/python

import sys


def main(argv):
    word2count = {}
    line = sys.stdin.readline()
    try:
        while line:
            list = line.split(",")
            print('%s\t%s' % (list[1],list[5]))
            line = sys.stdin.readline()
    except "end of file":
        return None
if __name__=="__main__":
    main(sys.argv)
```

## Reducer.py

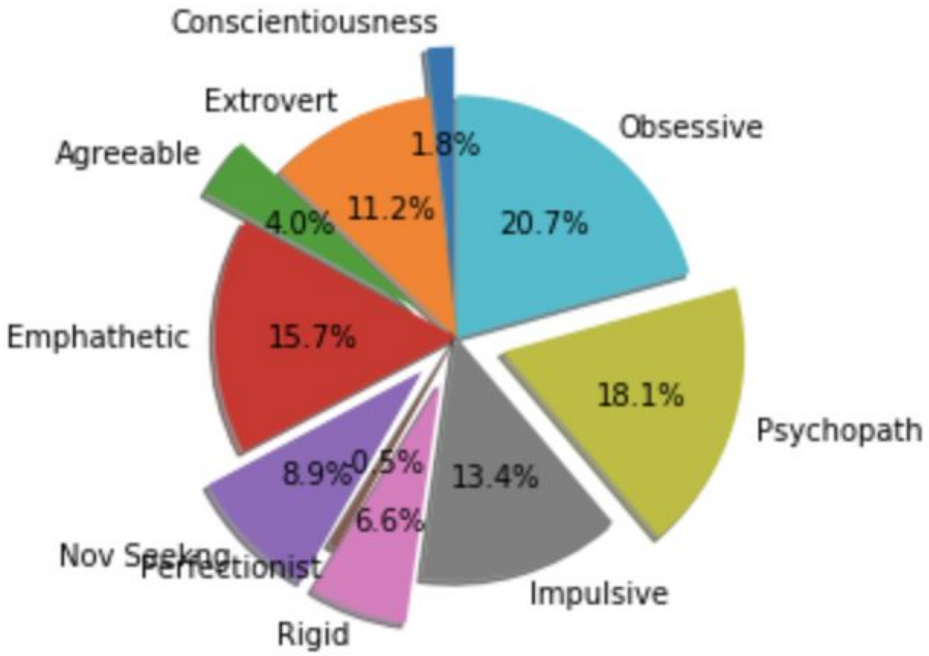
```
#!/usr/bin/python
from operator import itemgetter
from collections import defaultdict
import sys
word2count = {}
l = []
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word = line.split('\t')
    if(len(word)==2):
        if(word[0] in word2count):
            word2count[word[0]].append(word[1])
        else:
            word2count[word[0]] = [word[1]]
for word in word2count.keys():
    print '%s\t%s' % (word,word2count[word])
```

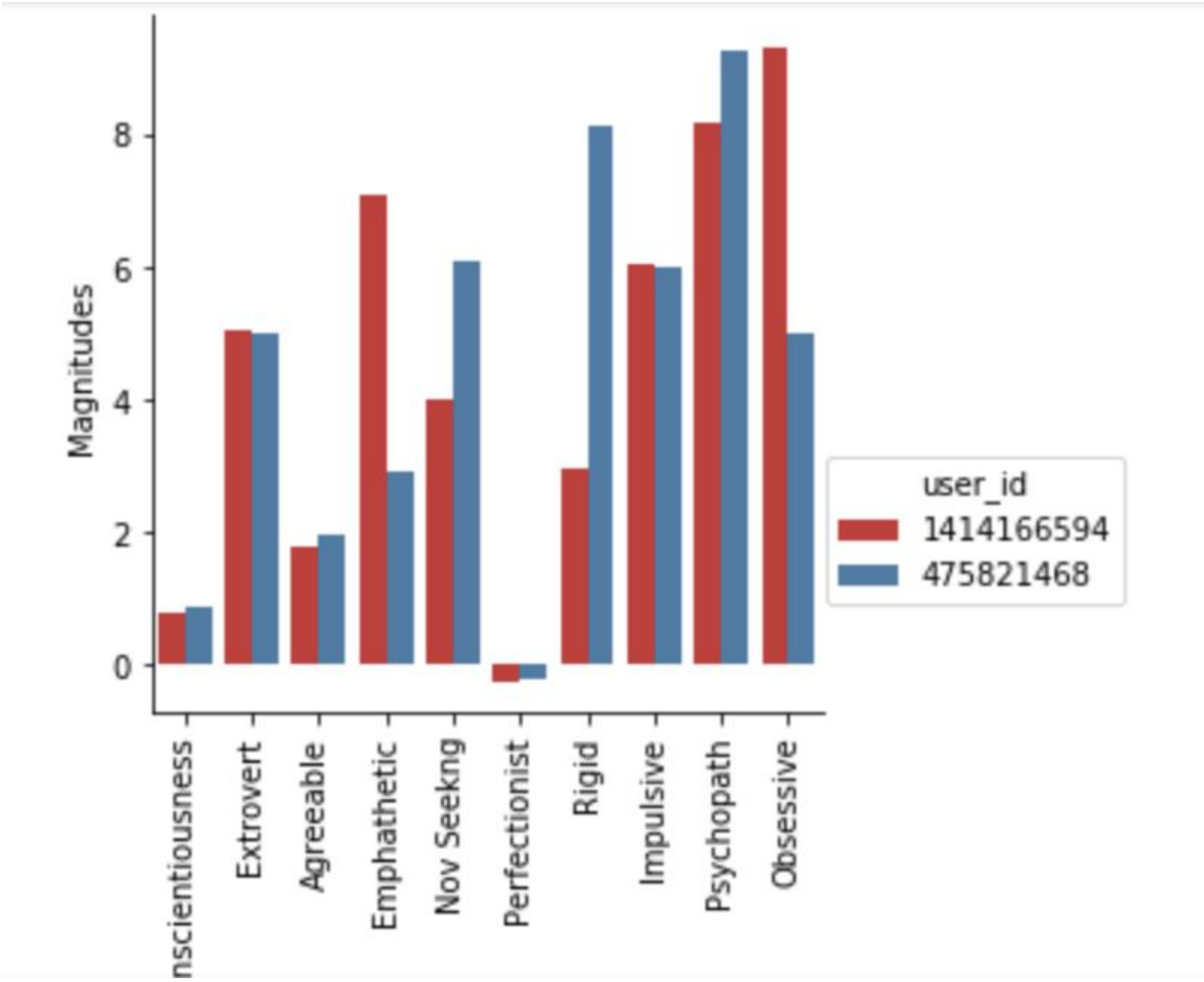
## 5.2 Design document and flowchart



# 6. Data analysis and discussion

## 6.1 Output generation




Magnitudes
8
6
4
2
0
nscientiousness
Extrovert
Agreeable
Emphathetic
Nov Seekng
Perfectionist
Rigid
Impulsive
Psychopath
Obsessive
user_id
1414166594
475821468

## 6.2 Output Analysis

As, you can see from the generated pie chart above this is a pie chart of a person and his personality that was derived from his twitter data. We can see that there are some major personality trends in this chart and from that we can derive his personality and also after that recommend a person having a very similar personality like this user by creating their charts and comparing. Thus, we can get find out personality of any user if we have his/her data and

Above is the pair of the two users that were recommended to one another based on their personality that was predicted with the help of the different modeling algorithms.

You can see from the above chart that the similarities the two person possess because of their personality and how we compared them using their individual personality vectors.

## 6.3 Output against hypothesis

We implemented Supervised Machine Learning models, the model takes a profile personality vector as input and will output the profiles according to the preferences of the user.

Our hypothesis was that we give a user smart recommendation for his profile based on his personality vector and we have been able to achieve it so far.

It was done using the models that we had decided during the starting of this project and the evaluation came out to be true as we have been successful in getting the personality vector from the user.

# 7. Conclusions and recommendations

## 7.1 Summary and conclusions

The performance of Linear SVMs and logistic regression are comparable in practice. SVMs with nonlinear kernel are used if the data won't be linearly separable (or need to be more robust to outliers than LR will normally tolerate). Otherwise, we try logistic regression first and see how you do with that simpler model. If logistic regression fails, we try an SVM with a non-linear kernel like a RBF.

SVM fits a function (hyperplane) that attempts to separate two classes of data that could be of multiple dimensions.

SVM could have difficulty when the classes are not separable or there is not enough margin to fit a (n-dimensions - 1) hyperplane between the two classes.

However, in our case, the number of features is very large (5230). As the classes are linearly separable, using SVM with a linear kernel seems a viable option.

Random Forests generally needs larger number of instances to work its randomization concept well and generalize to the novel data. In addition, in one way or another, random forests works with combination of some kind of soft linear boundaries at the decision surface thus I believe that this is still below the success of max margin SVM non linear boundaries.

Thus if you have small amount of data compared to possible variations of the instances than SVM is better choice.

## 7.2 Recommendations for future studies

For further studies and recommendations we can use the same set of data and try using other distribution models than the ones we have tried on this dataset.

By doing so we can know how the other type compares to this data models and also we can extent that by also changing the data set and trying the different models on different dataset.

Thus, by doing so, we can come to know about the feasibility of the data with the different models and how successfully it can compute and predict the personality of the user and after doing that the important thing to also take care of is that how well it can predict the right person to match it with.

Thus, in the future studies we can try using different neural models to get the best match case scenario on different data sets and can thus come to a conclusion that a particular model is the best for this type of recommender system.

# 8. Bibliography


- [1] Chris Sweeney Liu Liu Sean Arietta Jason Lawrence. "HIPI: A Hadoop Image Processing Interface for Image Based MapReduce Tasks." University of Virginia.
- “Mining of Massive Datasets,2nd Edition”, by Anand Rajaraman, Jure Leskovec, Jeffrey D. Ullman, ISBN: 978-1107077232, Cambridge 2014.
- ACM Digital Library -Assessing personality using demographic information from social media data, https://dl.acm.org/citation.cfm?id=2789187.2789201
- Analyzing Personality through Social Media Profile Picture Choice, http://wwbp.org/papers/persimages16icwsm.pdf
- Point-of-Interest Recommendation for Location Promotion in Location Based Social Network, https://ieeexplore.ieee.org/document/7962475/
- Adaptive Location Recommendation Algorithm based on Location

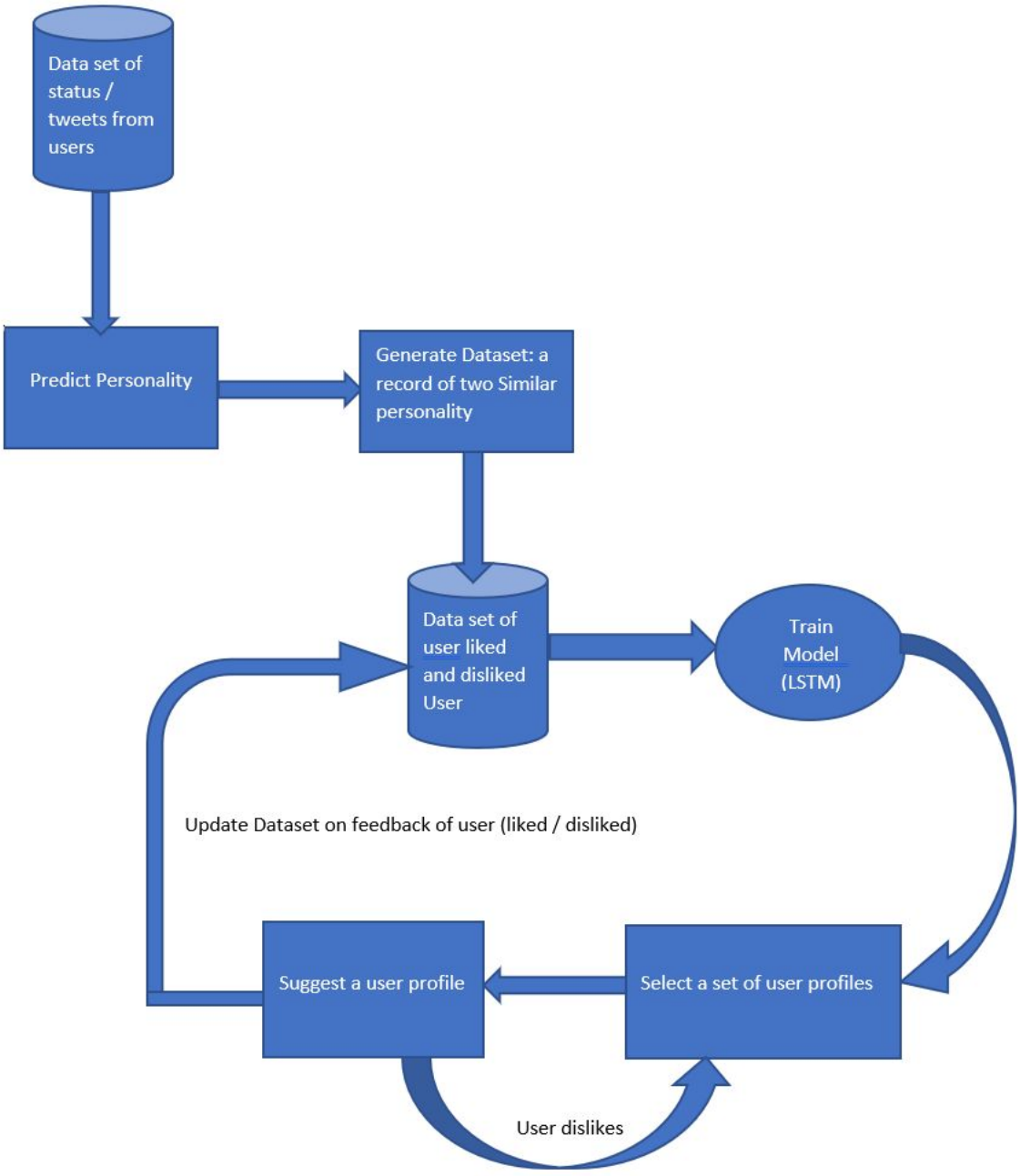
Based Social Networks, https://ieeexplore.ieee.org/document/7250231/

- Knowledge-driven Approach to Predict Personality Traits by Leveraging Social Media Data, https://ieeexplore.ieee.org/document/7817065/
- Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/
- https://www.medcalc.org/manual/logistic_regression.php
- http://www.statisticssolutions.com/what-is-logistic-regression/
- https://en.wikipedia.org/wiki/Support_vector_machine
- https://blog.statsbot.co/support-vector-machines-tutorial-c1618e635e93
- https://en.wikipedia.org/wiki/Cosine_similarity
- http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/
- https://pandas.pydata.org/
- https://www.anaconda.com/

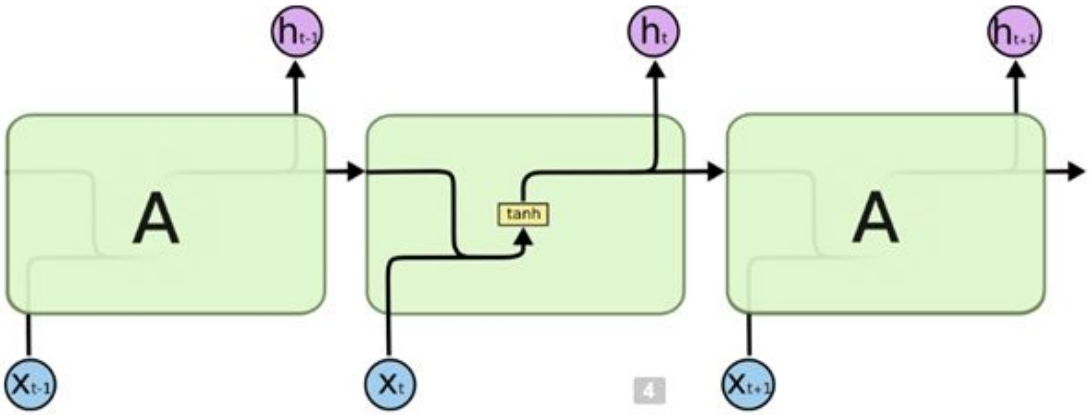
- https://seaborn.pydata.org/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# 9. Appendices

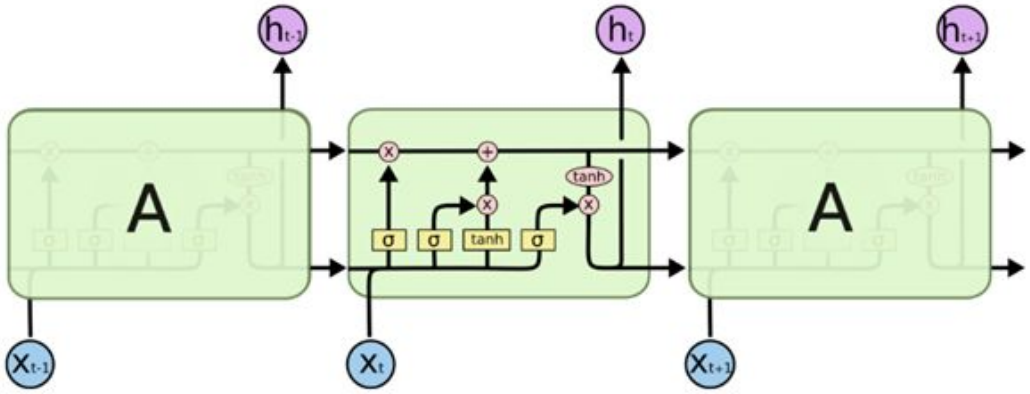
## 9.1 Program flowchart



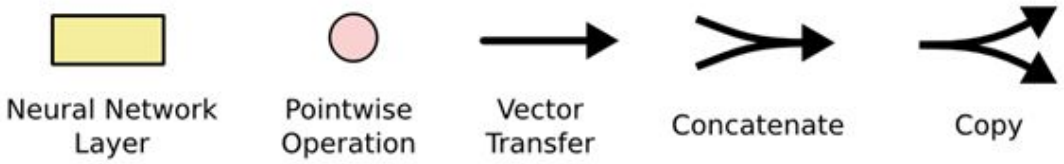
## 9.2 Long Short Term Memory Networks



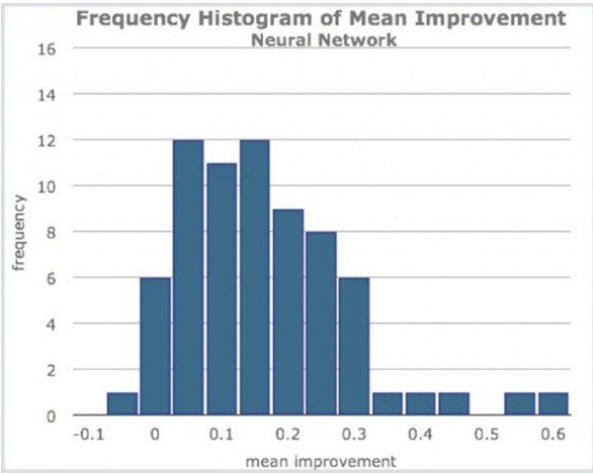

The repeating module in a standard RNN contains a single layer.

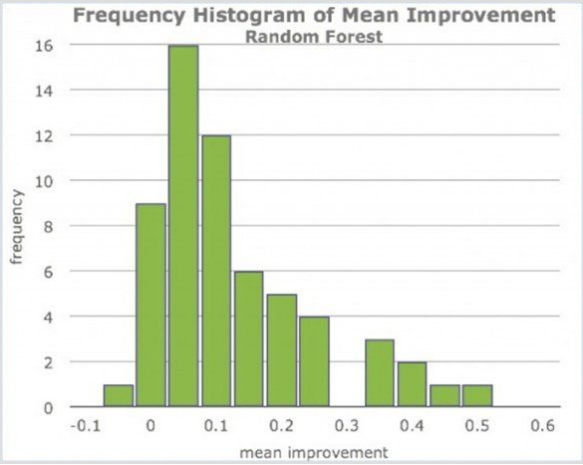


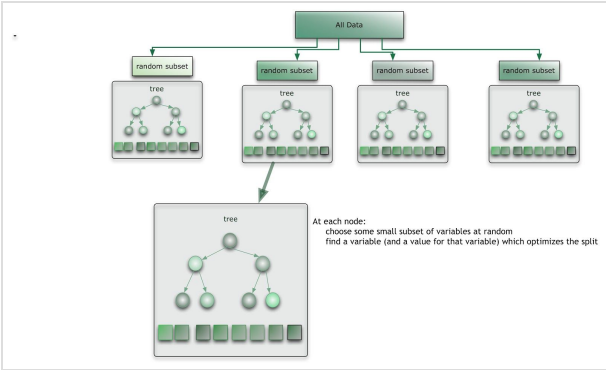

The repeating module in an LSTM contains four interacting layers.

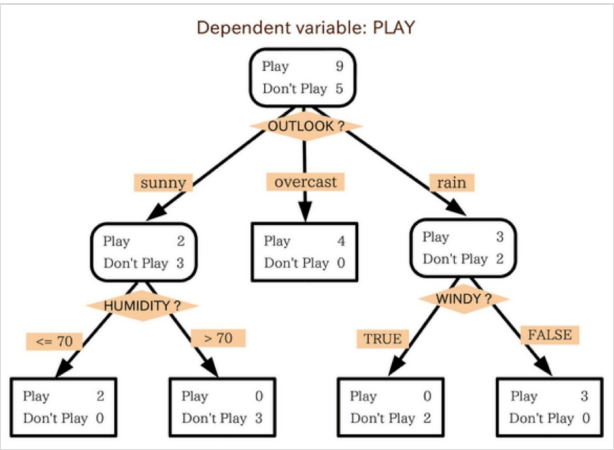


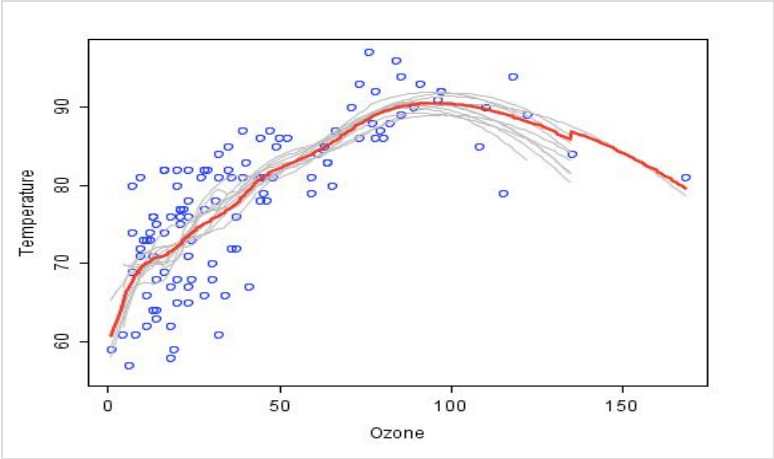

## 9.3 Random Forest Model



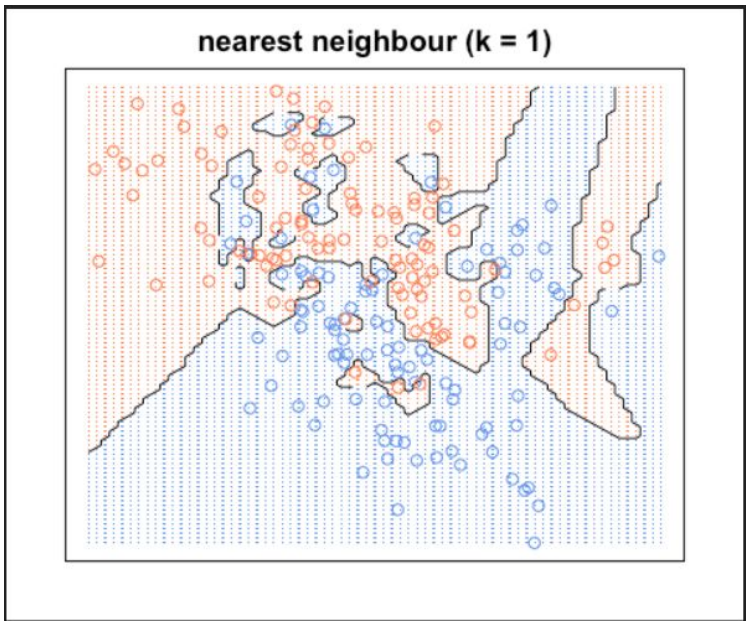



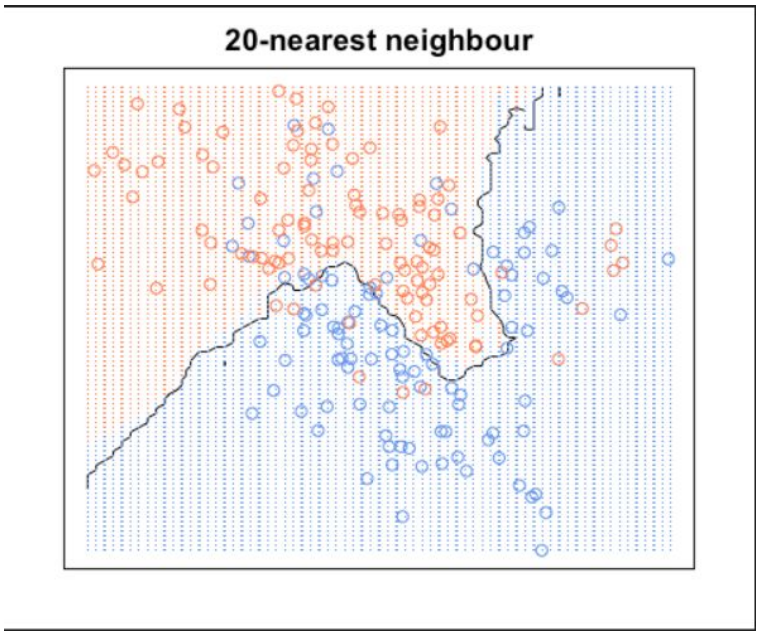




Temperature
60
70
80
90
0
50
100
150
Ozone

## 9.4 KNN



THANK YOU!