Text Classification: Identify Personality with Myers-Briggs Personality Type Dataset

Mengtian Guo, Jiachen Liu, Jing Xu, Yujuan Fu
Instructor: Yuekai Sun

January 29, 2019

Abstract

Due to the increase in use of social media, reliable personality detection from text shows its importance. Our project focused on building a classifier to classify people into different Myers-Briggs Personality Type Index (MBTI) personality types based on their posts on social media. We used Naive Bayes, Support Vector Machines(SVM), Logistic Regression, Random Forest, K Nearest Neighbors, Linear Discriminant Analysis, and Neural networks to build different models. Logistic Regression and SVM showed relatively high accuracy, which was about 0.85. We also extracted important features in terms of personality classification based on our models.

1 Introduction

Personalized service provides a better user experience at a lower cost than the traditional, standardized service. It then becomes worthwhile to investigate into personal information. The extensive data from social media enables people to have insight into more personal information, such as one's personality. The resulted user information can be applied to commercial purposes, research, career counseling and so on. In this project, we used the Myers Briggs Type Indicator (MBTI), which is one of the most popular personality tests in the world, as an indicator for personality type[1].

We aimed at the potential for our classifier to be more accurate and reliable. We used different machine learning models to do the personality classification based on text classification and compared their performance based on accuracy and analysis of confusion matrix.

2 Relevant Works

Getting better understanding of human personality has long been researched. The typical approach to analyzing text is to count word usage based on close vocabulary. However, as addressed by H Andrew Schwartz in his paper, open-vocabulary analysis could provide more insights than traditional approach.

H Andrew Schwartz explored the relationship between personality, gender, and age in the language of social media[2]. The method used in the research, differential language analysis (DLA), has three main steps: extracting linguistic features from social media messages, correlating relative frequencies of those features with characteristics, visualization. In our project, we followed the same first two steps. Instead of focusing on the language feature analysis, we put more efforts on predicting personalities based on language features extracted. Although H Andrew Schwartz also used SVM to predict personality, the accuracy is only about 0.4. Our project tried to increase the accuracy of prediction by using other kinds of machine learning models and different ways of preprocessing.

3 Dataset and Features

Myers-Briggs Personality Type (MBTI) Dataset The MBTI Dataset from Kaggle [1] contains 8675 samples. There are two columns in the data set: type and posts. Each row contains a person's personality type and more than 50 posts. The data were collected from PersonalityCafe Forum , where over 135,668 people participate

and 9,722,480 posts were sent[6]. By drawing the distribution of target variables as shown in Figure 1, we found an unbalanced distribution among different personalities.

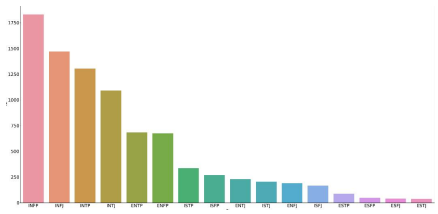

Figure 1: Distribution of target variables

Pre-processing

1. **Cleaning** The posts in our data set could contain websites, emoticons and punctuation which we were not taking as a factor for determining the personality of a person. So the first step of pre-processing is to clean data, including removing all irrelevant information, changing uppercase letters to lowercase and eliminating spaces between words to one.

2. **Stemming** After clearing the data, we normalized the sentences by stemming so that different representations of the same word come in the same feature. For example, "matrix" and "matrices" become the same word after stemming.

3. **Tokenization** First we got rid of stop words in English. Those stop words are very common words that appear in text but carry little meaning. Then we did vectorization to extract information from text. The technique we used is Bag of Words (BOW). First, we created our own dictionary which was all words occurring in our training data. After having successfully created the dictionary, we were able to return a numerical vector given an input text. In this step, we used Term Frequency (TF) instead of word counts and divided the number of occurrences by the the sequence length to avoid the biases caused by only counting how many times a word occurs in a text. Then we took the top 1600 frequently tokens composed of single words and two-word phrases. We find that considering two-word phrases as a token increases the accuracy since some word groups contain different meanings from single word.

Personality Labels The MBTI divides everyone into 16 distinct personality types across 4 axis and We had two ways of treating the personality labels:

- Introversion (I) - Extroversion (E)
- Intuition (N) - Sensing (S)
- Thinking (T) - Feeling (F)
- Judging (J) - Perceiving (P)

1. **Binary Encoding.** We represented each label as a vector $\vec{y_i}$ in $\mathbb{R}^4$ with binary entries, indicating personality types across 4 axis. In this case, we assumed that the four axis are independent from each other.

2. **Decimal Encoding.** We represented the 16 personality types by decimal numbers from 0 to 15. It turned out that the decimal encoding has worse prediction and we chose to mainly focus on binary encoding.

4 Description of Problem

Classification of personality has drawn great interest not only from psychologists but researchers in other fields. Even when companies hire people, they take personality into consideration and try to group people with similar personality to cooperate together. However, currently available personality tests administered by trained psychologists only has hit rate around 0.5 and it is largely based on subjective surveys. That is, there is a high probability that taking the test for twice and get different result.

Text classification is a good way to solve the problem but current research by Alam. reported a 61.79% accuracy using multinomial Naive Bayes[13], which is not satisfactory at all. We tried to select different features and adjusted parameters to further improve accuracy.

5 Methods

In this project, we used 6 classification models including Naive Bayes, Support Vector Machine, Logistic Regression, Random Forest, K Nearest Neighbors, Linear Discriminant Analysis, to classify personality types. We made classification on the four perspectives of personality respectively, which means we classified each entry of the response variable to be 0 or 1 separately based on all the data. We used the function "MultiOutputClassifier" in our models to realize this[11].

5.1 Naive Bayes

Naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $x = (x_1, ..., x_n)$ representing some n features (independent variables), it assigns to this instance probabilities $p(C_k|x_1, ..., x_n)$ for each of K possible outcomes C_k, where k $\in \{1, ..., K\}$.
Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

using the chain rule, we can express the joint probability model as $p(C_k, x_1, ..., x_n) = p(c_k)p(x_1|C_k)p(x_2|C_k)...$ Thus, the joint model can be expressed as

$$p(C_k|x_1, ..., x_n)p(C_k, x_1, ..., x_n) = p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

Above is the naive Bayes probability model. The naive Bayes classifier combines this model with a decision rule–MAP decision rule. The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\widehat{y} = C_k$ for some k as follows:

$$\widehat{y} = argmax_k p(C_k)\prod_{i+1}^{n} p(x_i|C_k)$$

In practice, we set up a feature vector based on the pre-processed data and transform it into frequency table. After that we calculate the posterior probability of different types using Bayes equation to determine the classification result. The Naive Bayes classifier in sklearn is suitable for classification with discrete features (e.g., word counts for text classification)[11]. The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

5.2 Support Vector Machines

Support Vector Machines constructs a hyper-plane $w'\phi(x)+b = 0$, which can be used for classification. A good separation is achieved by the hyper-plane that has the largest distance to the support vector. Also we can add slack variable to classify inseparable data, thus the goal is to minimizing the cost function.

$$C(w) = \frac{2}{\|w\|^2} + c\sum_{i=1}^{n}\xi_i = \frac{1}{2}w'w + c\sum_{i=1}^{n}\xi_i$$

subjected to the constrains $y_i(w'\phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0$ where c is penalty parameter that balances classification errors vs. the complexity of the model and ξ_i is the so called slack-variable. We usually try to solve through a dual formulation problem.

$$min\frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n}\alpha_i$$

subjected to the constrains $\sum_{i=1}^{n} y_i\alpha_i, 0 \geq \alpha_i \geq C$ Where α_i are non-negative Lagrange multipliers and K(.) is a kernel function.
In practice, SVM works similar when apply to text classification but with more discrete features. The algorithm tries to find a optimal hyper-plane to separate different types based on the supporting vectors. To optimize our solution, we set the loss function to be 'hinge' loss in order to give a linear SVM[11]. Besides, we set the penalty to be the squared euclidean norm 'L2', which means we add squared magnitude of coefficient as penalty term to the loss function. Also we set 'alpha',which is the constant that multiplies the regularization term, to be 1e-4 which is our penalty on the loss. Finally we set the iteration stop criterion 'tol' to be 0.001, which means the iterations will stop when (loss > previousloss - tol).

5.3 Logistic Regression

Logistic regression is a multivariate analyzing technique which can form a multivariate relation between dependent variables and independent variables. A linear combination of predictors is used to fit a Logistic transformation of the probability of occurrence for each subject (π_i) as

$$Ln[\widehat{\pi}_i/(1 - \widehat{\pi}_i)] = \beta_0 + \beta_1 X_{1i} + ... + \beta_p X_{pi}$$

Regression coefficients are fitted by maximum likelihood estimation, and by solving the Logistic in order to get the probability of success for each subject is estimated as

$$\widehat{\pi}_i = \frac{e^{\beta_0+\beta_1 X_{1i}+...+\beta_p X_{pi}}}{1 + e^{\beta_0+\beta_1 X_{1i}+...+\beta_p X_{pi}}}$$

In our project, we used the function "LogisticRegressionCV" in sklearn, which is a logistic classifier with its own cross validation estimator[11]. To optimize the algorithm, we choose 'lbfgs' solver to handle multinomial loss. We set the penalty to be L2, which means we add squared magnitude of coefficient as penalty term to the loss function. Also, we set the

maximum iterations to be 10000, which is much larger than the default value in order to get more accurate classification. We set the cross-validation to be 5-fold which means we randomly split the data into 5 sets and take one group as the test data set while the remaining groups as a training data set. Finally we will fit a model on the training set and evaluate it on the test set.

5.4 Random Forest

A random forest is a estimator that fits a number of decision tree classifiers using random bootstrap samples of the original data sample and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is the same as the original input sample size and the samples are drawn with replacement.

$$\left.\begin{array}{c} D_1 \longrightarrow T_1 \\ \vdots \\ D_M \longrightarrow T_M \end{array}\right\} f(x) = \frac{1}{m}\sum_{i=1}^{m} T_i(x)$$

In our project, we used the function "RandomForestClassifier" and set the parameter "n_estimators" to be 100, which means there will be 100 decision trees and each time the data set feed into the model would be decided by bootstrap[11]. The number of features selected each time is default to be square root of the total number of features. The final result of classification would be derived by averaging the prediction of the 100 trees.

5.5 K Nearest Neighbors

K nearest neighbors(KNN) is a non-parametric algorithm which finds the closest K points for every point in the data and make predictions based on the average of those k points. Since KNN has no parameters, it makes no assumption about the distribution of the data set.

In this project, we used the function "KNeighborsClassifier" to fit the KNN model[11]. We set the number of K to be 5, and by default the nearest 5 points will have the same weight.

5.6 Linear Discriminant Analysis

For linear discriminant Analysis(LDA), predictions could be made based on Bayes' rule:

$$\begin{aligned} P(y=k|X) &= \frac{P(X|y=k)P(y=k)}{P(X)} \\ &= \frac{P(X|y=k)P(y=k)}{\sum_l P(X|y=l)P(y=l)} \end{aligned}$$

$P(X|y=k)$ is modeled as a multivariate Gaussian distribution:

$$P(X|y=k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} exp(-\frac{1}{2}(X-\mu_k)^t\Sigma_k^{-1}(X-\mu_k))$$

LDA makes predictions based on estimating P(y=k), μ_k and Σ_k from the training data.

$$y = argmax_y P(X|y)P(y)$$

In our project, we used the function "LinearDiscriminantAnalysis" to fit the model[11]. The model fits a Gaussian distribution to each class and assumes that all the classes share the same covariance matrix.

5.7 Neural Networks

Neural Networks(NN) is a multi-stage, multi-unit classifier inspired from the biological neuron cells as illustrated by figure. The neural net is formed by connecting the output of certain layer to the input of next layer and add weights on the inputs. Each layer of the neural net receives input and produce output based on the activation.

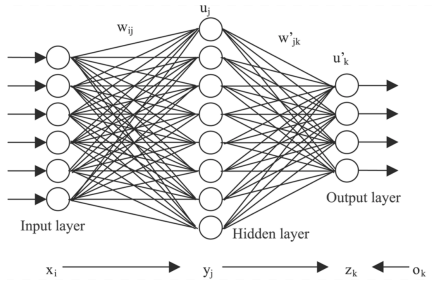

Figure 2: Neural Networks

In this project, we used keras to build a three-layer neural network. The first two layers have 512 neurons and the last layer has 4 neurons to produce four output for each perspective of personality. We chose the activation function to be rectifier for the first two layers and softmax for the last layer so that the output is in the form of softmax transformation. We also added Dropout to our model so that 0.3 of the input units are set to 0 during training, which helps to prevent over-fitting.

6 Conclusion

6.1 Binary Encoding vs. Decimal Encoding

Model	Accuracy for Binary(%)	Accuracy for Decimal(%)
Naive Bayes	78.11	37.23
SVM	84.32	16.74
Logistic Regression	85.49	59.62
Random Forest	80.33	55.97
K Nearest Neighbors	79.38	33.07
Linear Discriminant Analysis	80.48	55.44
Neural Networks	54.28	55.32

Table 1: Prediction Accuracy(%) of Binary Encoding and Decimal Encoding

The binary encoding performs far better than decimal encoding. Therefore, we decided to use binary encoding.

In the binary encoding case, we assumed that the four axes of personality types are independent. Then we combined the results from four binary classification problems. In the decimal encoding case, we treated 16 personality types instead of a combination of 4. In this situation, we ignored the correlation within the 16 personality types.

In both cases, we choose features with a maximum length of two words, giving a better result than one-word case.

6.2 Comparison of Models

As we can see from our result shown in Table.1, all our models give us a good prediction accuracy of more than 79%. Among the seven models, Support Vector Machines and Logistic Regression generate the best result, which is worthwhile for a further investigation.

Figure 3, Figure 4, Figure 5 below show the confusion matrix for each model. The x axis shows the predicted label and the y axis shows the true label. The numbers on the diagonal show the correctly predicted personality labels and others stand for mislabeled samples. We separated the models into three pairs based on their performances on the confusion matrices.

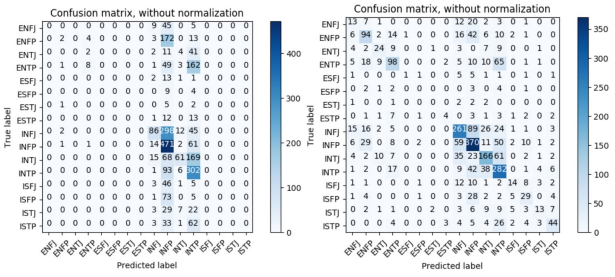

Figure 3: Naive Bayes (left) and Support Vector Machines (right)

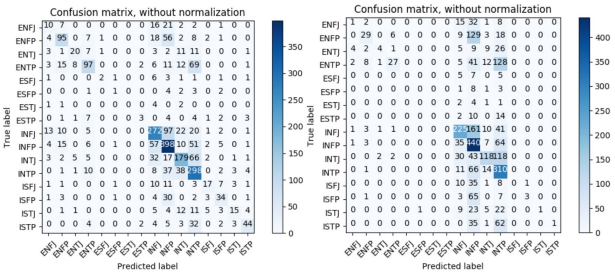

Figure 4: Logistic Regression (left) and Random Forest (right)

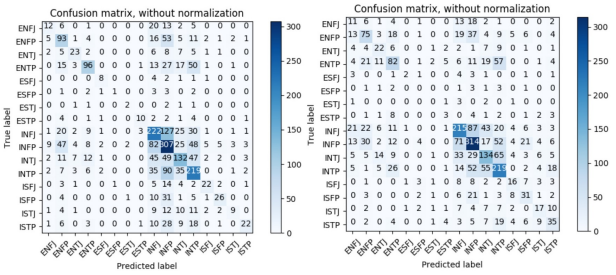

Figure 5: K Nearest Neighbors (left) and Linear Discriminant Analysis (right)

Support Vector Machines and Logistic Regression: In this project, we used the default mode for the SGDClassifier function, giving a linear model for SVM. Both models are linear, assuming features in our dictionary are independent and linearly separable. The processes of deriving parameters are similar. The two models usually perform very close to each other. However, the intuition behind the two algorithms are different:

• Support Vector Machines generates a hyper-plane which is able to maximize the minimum distance of data points to the margin by minimizing hinge loss.

• Logistic Regression comes up with a hyper-plane which is able to maximize the probability of data falling in one side, namely one class of the label by

minimizing logistic loss.
The two models perform very similar to each other, which is reasonable because their loss functions are very similar to each other. But since the loss function of Logistic Regression diverges faster than that of SVM, it is more sensitive to outliers and tends to perform better compared to SVM when the data set is small. This may explain why Logistic Regression works better in this case, since many personality types have small number of observations.

Naive Bayes and Random Forest: From the confusion matrix, we found that both models tend to generate predicted labels on more frequently occurred classes in the training set. That's because these two models are very sensitive to whether the data set is balanced.
• Naive Bayes: The classifier predicts the class y to be the one that maximize $P(X|y)P(y)$. For the classes with large amount of observations, P(y) is estimated to be higher. Therefore, it is more likely for the classifier to classify a new observation into such classes.
• Random Forest: Each decision tree is built on a bag, which is drawn from a uniform random sample from the data. During the process, the model assumes that the probabilities that each class is drawn are the same. But this is not true for unbalanced data set. The result is biased when averaging biased tree results from unbalanced classes. We picked the parameter for $n\_estimators$, the number of trees as 100. Changing this parameter will not make improvement since the sample are drawn from the unbalanced data set.

K Nearest Neighbors and Linear Discriminant Analysis: From the confusion matrix, both models tend to generate more disperse predicted labels among all possible personality types.
• K Nearest Neighbors: Since this model makes no assumption about the data distribution, we can safely apply it to our data set. The result is obtained by inspecting the nearest 5 points. The unbalanced data set can influence the accuracy. For the classes with small number of observations, the observations of such classes are limited and may not be representative for the class. For the major classes, their observations are widely distributed. It is very likely that an observation from the minor class is surrounded by observations from major classes. This can be solved by shuffling the data set several times and averaging the result. The value of k is also worth inspecting. When the predicted labels are more disperse than the actual result, the data is possibly slightly underfitted. A possible way to fix it is to decrease the value of k.
• Linear Discriminant Analysis: The model makes the assumption that the data in each class follows normal distribution and shares the same covariance matrix. This assumption is not necessarily true, which can be a possible source of error.

Neural Networks: Compared to the other models, the performance of Neural Networks is not very good, which may due to the inappropriate coefficients we chose. In order to get higher accuracy, we could do some experiment in terms of the coefficients in the future. But for this project, we choose to focus on more basic models.

6.3 Confusion Model Analysis

A confusion matrix is a summary of prediction results on a classification problem. It can tell us the ways the classification model is confused when making predictions. We did some calculation to help us understand the model that performs best, the Logistic Regression model.

Definition of Terms

- Positive (P): Observation is positive
- Negative (N): Observation is not positive
- True Positive (TP): Observation is positive, and is predicted to be positive
- False Negative (FN): Observation is positive, but is predicted to be negative
- True Negative (TN): Observation is negative, and is predicted to be negative
- False Positive (FP): Observation is negative, but is predicted to be positive

Classification Accuracy is given by the relation

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall is defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples.

$$Recall = \frac{TP}{TP + FN}$$

Precision is defined as the ratio of the total number of correctly classified positive examples divide to the total number of predicted positive examples.

$$Precision = \frac{TP}{TP + FP}$$

F-measure will always be nearer to the smaller value of Precision or Recall.

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

For the logistic regression model,

	Accuracy	Recall	Precision	F-measure
ENFJ	0.95	0.17	0.62	0.27
ENFP	0.90	0.49	0.56	0.52
ENTJ	0.96	0.33	0.65	0.44
ENTP	0.89	0.43	0.50	0.46
ESFJ	0.99	0.12	0.20	0.15
ESFP	0.99	0.08	-	-
ESTJ	0.99	0.00	0.43	0.00
ESTP	0.98	0.12	0.61	0.19
INFJ	0.81	0.61	0.57	0.59
INFP	0.77	0.72	0.60	0.66
INTJ	0.86	0.57	0.51	0.54
INTP	0.79	0.74	0.52	0.61
ISFJ	0.96	0.31	0.55	0.40
ISFP	0.95	0.43	0.60	0.50
ISTJ	0.96	0.25	0.70	0.36
ISTP	0.95	0.44	1.00	0.62

Table 2: Confusion Matrix Analysis for Logistic Regression

The first 8 and the last 4 classes, which have small observations, have high accuracy in terms of prediction, while the classes with more observations have low accuracy. That may be because when there are small number of observations and they are very close to each other, it is easy to separate them from the other classes. But when the class has a lot of observations, its distribution is more complex and it is difficult to decide a boundary. For the classes with small observations, although the accuracy is very high, we cannot trust the predicted result since the data we have may not be representative for a bigger population. Another possible reason is that, since the accuracy also considers the correctly classified negative cases. For a class with very small amount of observations, the accuracy of classifying this class could still be very high even though the model predicts negative for very observation since majority of the data is truly negative. We therefore cannot trust all the accuracy. The accuracy of our model in terms of predicting the classes with big observations is about 0.8, which is more representative than the overall accuracy.

The recall shows the sensitivity of the model, which is whether the model could classify an observation to positive when it is actually positive. The recalls for 'ESFJ', 'ESFP', 'ESTJ', 'ESTP' are relatively low compared to the other types. That's because the class of 'ES' has very small number of observations and the observations used in training set is not representative for the whole class. It is therefore hard to decide an observation to be in this class during prediction. Combined with the analysis of accuracy and observe that these four classes have high accuracy, we could conclude that the main reason why the four classes have high accuracy is their small number of observations. On the other hand, the classes with more observations do better in recall, which is reasonable.

The result of precision shows that when our model predicts an observation to be a certain class, only about 0.5 of the time the observation is true in that class. Therefore, our model could only be used as reference when target group is different.

Taking both precision and recall into consideration, we have F-measure ($F_{0.5}$), which is the harmonic average of the precision and recall. In most cases, we have a trade-off between recall and precision, which means optimizing sometimes increase one and disfavoring the other. The F-measure gives us a way to equally treat precision and recall. The results above shows an average F-measure of around 0.6 considering classes with large observations. For further study, there is still chance to optimize this model.

6.4 Important Features Extraction and Verification

In this project, we used TFIDF to reflect how important a word is to a document in a collection. We output the most important features, which means those with higher TFIDF, in different personalities to verify that it conforms to the MBTI classification standard. Following are the keywords for 'ENFJ' 'ENFP' 'ISTJ' 'ISTP':

# 'ENFJ':	# 'ENFP':	# 'ISTJ':	# 'ISTP':
Most correlated unigrams:			
ex	hahaha	girlfriend	sports
dated	lol	sensing	fucking
lol	crazy	wink	shit
hugs	haha	political	car
	XD	direct	fuck
Most correlated bigrams:			
don need	oh god	know don	don need
personality types	feel like	oh god	feel like
make feel	just wanted	think people	don care
sent iphone	best friends	little bit	just want
using tapatalk	makes feel	feel like	pretty good
	maybe just	don like	best friends
		welcome forum	don need

Table 3: Most important features for EN- IS- types

These sentence fragments imply the users' personality. Based on the most important features, we can describe people who are 'ENFJ' 'ENFP' as enthusiastic, inspiring and being able to mesmerize their listeners. Also, we can describe 'ISTJ' 'ISFJ' as practical, fact-minded and always ready to defend their loved ones. Combining with the figures below[??] , the result is very close to the professional explanation, which validates our prediction.

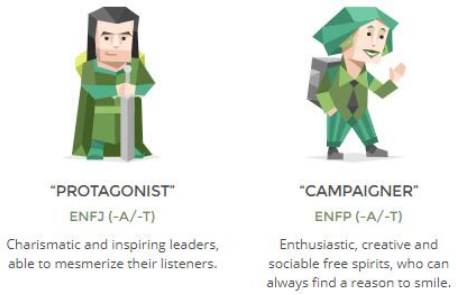

Figure 6: Diplomats personality type

Figure 7: Sentinels personality type

6.5 Speculation from the data set

Our data were extracted from *the last 50 things people have posted*[1] on the PersonalityCafe Forum, where posts are mainly about personality types. Therefore, the data we have is slightly different from various discussions online.

The data set is unbalanced. The number of samples from a specific personality type can be derived from summing up the corresponding rows in the confusion matrix in Figure.4. A large percent of personality types is distributed among the Four types, 'INFJ', 'INFP', 'INTJ', 'INTP'. Namely, there are more introversion-toward and intuition-toward people in the data set. Compared to the top 4 personality types within the United States from the website 16 Personalities with a sample of 22,678,145 people shown in table4 below[14], the result is different in the axis of *Introversion* and *Extroversion*. Since our data are randomly sampled, we can speculate that the website PersonalityCafe Forum is more popular to introvert people. More generally, introvert people may care more about their personality types. This shows that the data set we use may not be representative and our model may not be accurate in other cases.

Personality Types	Percentage
INFP	12.08%
ENFP	8.31%
INFJ	5.67%
ENFP	5.16%

Table 4: Top 4 personality types in the United States.

7 Summary

In this project, we focused on the relationship between personality and online posts. Using text classification, we built models to classify people into different personalities across 4 axis. Using stemming, tokenization, Bag of Words, and different personality labels, we translated the text data into the form that could be used in sklearn models. We used Naive Bayes, Support Vector Machines, Logistic Regression, Random Forest, K Nearest Neighbors, Linear Discriminant Analysis, and Neural Networks to build models and compared their performance. The SVM and Logistic Regression showed an accuracy of about 0.85 in predicting personality type based on posts. Therefore, out model could predict personality pretty

accurately. However, since the data set is unbalanced, we could only predict personality types 'INFJ', 'INFP', 'INTJ', 'INTP' well. And since the data set is not representative for general population, our model may not be useful in general cases. In actual application, it is reasonable to use information from target population and try to get balanced data.

Code

```
import pandas as pd
import numpy as np
import itertools
import re
import imp
from stemming.porter2 import stem
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import pickle
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout
from keras.utils import to_categorical
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.multioutput import MultiOutputClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
import xgboost as xgb
import seaborn as sns
from itertools import groupby

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

```
'''
Input the dataset
'''
data = pd.read_csv('mbti_1.csv')
types = np.unique(np.array(data['type']))
labelCoder = LabelEncoder()
inty = labelCoder.fit_transform(data['type'])
data['posts']=[re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'
                                                , ' ', sentence) for sentence in data['posts']
                                                ]
data['posts']=[re.sub("[^a-zA-Z]"," ",sentence) for sentence in data['posts']]
data['posts']=[re.sub(' +',' ',sentence).lower() for sentence in data['posts']]
data['posts']=[[stem(word) for word in sentence.split(" ")] for sentence in data['posts']]
data['posts']=[" ".join(sentence) for sentence in data['posts']]
vectorizer = CountVectorizer(ngram_range=(1, 2), max_features=1600, max_df=0.5)
''', max_features=1500, max_df=0.5'''
X = vectorizer.fit_transform(data['posts'])
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X)

'''
Author: Depture
Availability: https://www.kaggle.com/depture/multiclass-and-multi-output-classification
'''
per_list = {'I':0, 'E':1, 'N':0, 'S':1, 'F':0, 'T':1, 'J':0, 'P':1}
per_list_pair = [{0:'I', 1:'E'}, {0:'N', 1:'S'}, {0:'F', 1:'T'}, {0:'J', 1:'P'}]

def tran_per(p):
    return [per_list[l] for l in p]

def tran_back(p):
    s = ""
    for i, l in enumerate(p):
        s += per_list_pair[i][l]
    return s

bin_per = np.array([tran_per(p) for p in data.type])
print("Binarize MBTI list: \n%s" % bin_per)

x_train, x_test, y_train, y_test = train_test_split(tfidf, bin_per, test_size=0.3,
                                                random_state=2018)
'''
Test accuracy on different classifiers
'''
nb = MultinomialNB()
multiNb = MultiOutputClassifier(nb, n_jobs=-1).fit(x_train, y_train)
nbpredict = multiNb.predict(x_test)
nbAcc = np.mean(nbpredict == y_test)
print(nbAcc)
sgd = SGDClassifier(loss = 'hinge', penalty='l2', alpha=1e-4, tol=0.001)
multiSGD = MultiOutputClassifier(sgd, n_jobs=-1).fit(x_train, y_train)
sgdpredict = multiSGD.predict(x_test)
sgdAcc = np.mean(sgdpredict == y_test)
print(sgdAcc)
score=multiSGD.score(x_test, y_test)
print(score)
cv_fold=5
log = LogisticRegressionCV(cv=cv_fold, solver='lbfgs', max_iter=10000)
multiLog = MultiOutputClassifier(log, n_jobs=-1).fit(x_train, y_train)
logpredict = multiLog.predict(x_test)
logAcc = np.mean(logpredict == y_test)
print(logAcc)
score=multiLog.score(x_test, y_test)
print(score)
rf = RandomForestClassifier(n_estimators=100)
```

```
multiRf = MultiOutputClassifier(rf, n_jobs=-1).fit(x_train, y_train)
rfpredict = multiRf.predict(x_test)
rfAcc = np.mean(rfpredict == y_test)
print(rfAcc)
knn = KNeighborsClassifier(n_neighbors = 5)
multiKnn = MultiOutputClassifier(knn, n_jobs=-1).fit(x_train, y_train)
knnpredict = multiKnn.predict(x_test)
knnAcc = np.mean(knnpredict == y_test)
print(knnAcc)

'''
Plot confusion matrix
'''
lad = LinearDiscriminantAnalysis()
multilad = MultiOutputClassifier(lad, n_jobs=-1).fit(x_train.toarray(), y_train)
ladpredict = multilad.predict(x_test.toarray())
ldAcc = np.mean(ladpredict == y_test)
print(ldAcc)
plt.figure()
y_test = [tran_back(t) for t in y_test]
predict = [tran_back(t) for t in nbpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

predict = [tran_back(t) for t in sgdpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

predict = [tran_back(t) for t in logpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

predict = [tran_back(t) for t in rfpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

predict = [tran_back(t) for t in knnpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

predict = [tran_back(t) for t in ladpredict]
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

model = Sequential()
model.add(Dense(512, input_shape=(1500,)))
model.add(Activation('relu'))
```

```
model.add(Dropout(0.3))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(4))
model.add(Activation('softmax'))
model.summary()
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
histroy= model.fit(x_train, y_train,
                    epochs=2,
                    verbose=1,
                    validation_split=0.1,
        batch_size=32)
score = model.evaluate(x_test, y_test, batch_size=32, verbose=1)
predict = model.predict(x_test)
predict = [np.argmax(r) for r in predict]
y_test = [np.argmax(r) for r in y_test]
print("test accruacy:", score[1])
cnf_matrix = confusion_matrix(y_test, predict)
np.set_printoptions(precision=2)
plot_confusion_matrix(cnf_matrix, classes=types,
                      title='Confusion matrix, without normalization')
plt.show()

'''
Important features for 16-type personality
'''
from io import StringIO
data['id'] = data['type'].factorize()[0]
category_id_df = data[['type', 'id']].drop_duplicates().sort_values('id')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['id', 'type']].values)
data.head()

labels = data.id


from sklearn.feature_selection import chi2
N = 10
for type, id in sorted(category_to_id.items()):
  features_chi2 = chi2(X, labels == id)
  indices = np.argsort(features_chi2[0])
  feature_names = np.array(vectorizer.get_feature_names())[indices]
  unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
  bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
  print("# '{}':".format(type))
  print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-N:])))
  print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-N:])))
```

References

[1] J, Mitchell. *(MBTI) Myers-Briggs Personality Type Dataset.* Kaggle, 22 Sept. 2017, www.kaggle.com/datasnaek/mbti-type/home.

[2] Schwartz, H. Andrew, et al. *Personality, Gender, and Age in the Language of Social Media: The Open-Vocabulary Approach.* PloS one 8.9 (2013): e73791

[3] Boukkouri, Hicham EL. *Text Classification: The First Step Toward NLP Mastery.* Medium, 18 June 2018, medium.com/data-from-the-trenches/text-classification-the-first-step-toward-nlp-mastery-f5f95d525d73.

[4] Medlock, Ben W. *Investigating classification for natural language processing tasks.* No. UCAM-CL-TR-721. University of Cambridge, Computer Laboratory, 2008.

[5] Shaikh, Javed. *Machine Learning, NLP: Text Classification Using Scikit-Learn, Python and NLTK.* Medium, 23 July 2017, towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a.

[6] *PersonalityCafe Forum, www.personalitycafe.com/forum/.*

[7] Templeton, Graham. *Artificial neural networks are changing the world. What are they?* Medium, 12 October 2015, http://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they

[8] Sharma, Abhishek. *Confusion Matrix in Machine Learning,* https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

[9] *https://scikit-learn.org/stable/*

[10] *https://www.16personalities.com/personality-types*

[11] Scikit-learn, Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. *Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12,2825–2830, 2011*

[12] Keras, Chollet, François and others, 2015, https://keras.io

[13] F.Alam, E.A.Stephnov and G.Riccardi. *Personality traits recognition on social network-facebook* Proc of Workshop on Computational Personality Recognition, AAAI Press, Melon Park, CA, 2013, pp. 69.

[14] *United States Personality Profile. 16Personalities, www.16personalities.com/country-profiles/united-states.*