**PROJECT REPORT**Project Term May-July 2021

**Personality Prediction AI**

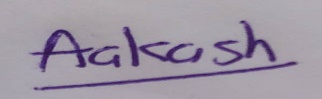
Submitted to:  
**LOVELY PROFESSIONAL UNIVERSITY  
PHAGWARA, PUNJAB**



SUBMITTED BY

**Aakash Maurya**

**11908728**

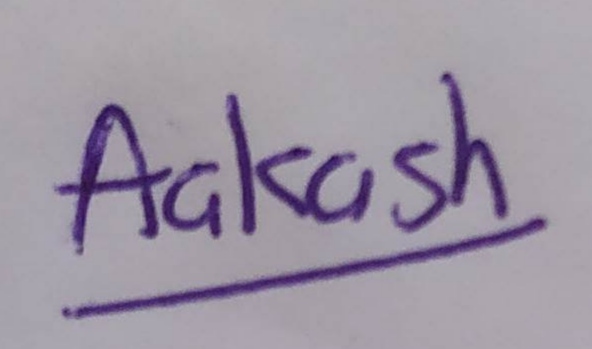
****

**DECLARATION**

We hereby declare that the project work entitled Personality Prediction AI is an authentic record of our own work carried out as requirements of Project for the award of B.Tech degree in from Lovely Professional University, Phagwara, under the guidance of Dr. Sagar Pande, during August to November 2021. All the information furnished in this project report is based on our own intensive work and is genuine.

**Aakash Maurya**

**11908728**



**CERTIFICATE**

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. They have completed this Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. No part of the work has ever been submitted for any other degree at any University. The Project is fit for the submission and partial fulfillment of the conditions for the award of B.Tech degree in Soft Computing  
from Lovely Professional University, Phagwara.

**Dr. Sagar Pande**

**School of Computer Science and Engineering,**

Lovely Professional University,

Phagwara, Punjab.

**TABLE OF CONTENTS**

Title Page i  
Declaration ii  
Certificate iii  
Table of Contents iv

INTRODUCTION  
MOTIVATION  
PROBLEM ANALYSIS  
DATASET ANALYSIS  
PROJECT DESIGN  
DATA FLOW DIAGRAM  
ADVANTAGE  
DISADVANTAGE  
STEPS  
TESTING  
APPLICATION  
CONCLUSION  
FUTURE SCOPE

SOURCE CODE

SNAPSHOTS

**Introduction**

Soft Computing is a set of algorithms, including neural networks, fuzzy logic, and genetic algorithms. These algorithms are tolerant of imprecision, uncertainty, partial truth and approximation. It is contrasted with Hard Computing Algorithms which finds provably correct and optimal solutions to problems.

**About Classification Algorithm**

Algorithms that help us segregate a given data set into different classes is known as classification algorithm. The models based on classification algorithms help in concluding input values that are given by the user during the training session of the model. The prediction of class labels is given by these models based on the data provided to them.

**Types of Naive Bayes Classification Algorithm**

* **Naïve Bayes**
  + Bernoulli Naive Bayes
  + Multinomial Naive Bayes
  + Gaussian Naive Bayes

**Motivation**

For this Project, My motivation for choosing such a topic comes because of my interest that I have in Psychology. I have researched a lot of sites to find suitable topic for the project, but I chose this which seems to have most affinity with me.

This topic doesn’t require any complicated files to import in the project. You need to do text classification on the dataset and train the model and predict using that model.

I have also researched for similar dataset and found one more popular dataset which is depended upon Big Five Trait ( OCEAN ), another method for determining about a person personality.

But, the reason I didn’t chose that dataset is because I am not familiar with that method of personality determination.

**Problem Analysis**

**Problem**

As the topic I have chosen is ‘Personality Prediction’, so o require to somehow predict someone personality with some input or clue.

**Given**

In this section, I will be taking account of resources that I have :  
The most important resources is Dataset, ‘MBTI Dataset’ which is downloaded from Kaggle.com. There were two most famous dataset:

1 : MBTI dataset  
2: OCEAN dataset

**Method**

After researching the dataset, I have learned that for this type of dataset, text classification is the most used Soft Computing method for creating program.

After referencing to many text classification algorithm templates, I used the Multinomial Naïve Bayes, as I find it easy to code and use.

This algorithm can be found in SciPy.

**Dataset Analysis**

**Dataset**

As previously mentioned, the dataset I am using is ‘MBTI Dataset’ from Kaggle.com.

It contains a total of 8674 rows, means 8674 individual posts and their personality type.

This dataset 2 columns:  
1 – Type  
2 – Posts

|  |  |  |
| --- | --- | --- |
|  | Type | Posts |
| 1 | *[Integer]* | *[String]* |
| …... |
| 8674 |

**Dataset Modification**

For making the model easy to train, I did some modification in the dataset before cleaning and training the model. So the resultant dataset looked like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Type | Posts | IvE | NvS | FvT | PvJ |
| 1 | *[Integer]* | *[String]* | *[Character]*  *( I, E )* | *[Character]*  *( N, S )* | *[Character]*  *( F, T )* | *[Character]*  *( P, J )* |
| …... |
| 8674 |

**Data Flow Diagram**

Users

Enter Test Input

MBTI Dataset

Model

**Advantage**

**Naïve Bayes Classification**

* This algorithm works very fast and can easily predict the class of a test dataset.
* You can use it to solve multi-class prediction problems as it’s quite useful with them.
* Naive Bayes classifier performs better than other models with less training data.

**Disadvantage**

**Naïve Bayes Classification**

* If your test data set has a categorical variable of a category that wasn’t present in the training data set, the Naive Bayes model will assign it zero probability and won’t be able to make any predictions in this regard. This phenomenon is called ‘Zero Frequency,
* It assumes that all the features are independent. While it might sound great in theory, in real life, you’ll hardly find a set of independent features.

**Steps**

**Dataset Modification**

To make the model easier, I have made some modification in the dataset. The column ‘Type’, gives us the type of personality between 16 personality trait.

If the model was made with that column the accuracy of the model would decrease, So I decided to make 4 different model, where each model deal with 1 of the 4 trait.

It makes the model simple and also increases the accuracy.

**Text Preprocessing**

Text Preprocessing is a method to remove noise ( useless character, in this case ) from the dataset. This is a basic process, when dealing with text information.

This process removes punctuation marks, web links, emoticons, numbers, etc. The result the cleaning is only words which doesn’t contain punctuation marks, symbols or numbers.

**CountVectoriser**

It is a function that counts the number of elements/words and how many times they have appeared in a document/text

In this project, this function gives every word a unique number and then calculates the number of times that word appeared in all documents.

**Dataset Splitting**

After cleaning the dataset, the next requirement to create model is training data and testing data.

**Training data** is the initial dataset used to train machine learning algorithms. Models create and define their rules using this dataset.

**Testing data** is another dataset used to test machine learning algorithms. Trained models are tested using this dataset.

So, I used train\_stest\_split() from SciPy to divide the MBTI dataset in two parts, one for training and other for testing with default train test ratio.

**Training the Model**

This is the easiest part to code, All you need to do is to declare model as variable and pass the dataset with fit\_transform() method.

The model automatically trains itself with the given training dataset.

**Creating a Simple GUI**

After training and testing the model, I have created a simple Graphical User Interface. In this GUI you enter your intended text and click on ‘Predict’ button, using the text the model predict each of the personality trait pair and gives result in probability and string version.

**Testing**

**Accuracy**

The Accuracy of a machine learning classification algorithm is one way to assess how often model classifies a data point correctly.

***Accuracy = No. of Correct Prediction / No. of Total Prediction***

**Tips for Improving Accuracy**

There are many ways to improve the model accuracy. As I have used CountVectoriser() and MultinomialNB(), I will be presenting few ways to improve accuracy

**max\_features**This parameter determines the number of feature ( or unique words, in this case ) to be used during the training of the model. It selected the most common features for the sparse matrix. This prevent the noise in the matrix, and also reduce the size of matrix by removing rarer features from the matrix.

**stopwords**Stopwords is a term used in Natural Language Processing, these are the most common words that doesn’t mean anything, words like : the, is, are, etc  
Category like Noun which doesn’t tell us anything about the writer.

**min\_df**This parameter defines a minimum threshold line for any feature to be present in sparse matrix.

Eg. min\_df = 2  
This means that the feature should be appearing 2 times is in any document for it to remain in the matrix.

Eg. min\_df = 0.25  
This means that the feature should be appearing 25% of times when aggregating all document

**Result of Improvements**

I have tested the model and made modification many times to get better accuracy. After testing many times, and tweaking many lines.

|  |  |  |
| --- | --- | --- |
| *Models* | *before improv.* | *after improv.* |
| *IEmodel* | 0.75 | 0.81 |
| *NSModel* | 0.78 | 0.86 |
| *ftmodel* | 0.74 | 0.81 |
| *pjmodel* | 0.70 | 0.78 |

**Precaution**

After some hands on testing, I have found that longer the input text the better the accuracy, and we should avoid using stopwords, like this, is am, etc. These words doesn’t improve the accuracy of the models.

**Application**

Although, the goal of this project was primarily focused on College Project.

Still, this project takes string input and predict the personality traits of person. It doesn’t matter the source of the text, as long as it is text free of any symbols and numbers, it can predict the personality trait with up to 86%.

We can use this model at many places such as Work distribution, Advertisements Recommendation, etc.

**Conclusion**

The conclusion that I have come to, regarding this project is that :

This project is simple to understand and when compared to the amount of coding and time spend on the project it gives quite a good result when used with the interface.

With some additional training, this project can be used for many purposes.

**Future Scope/Ideas**

**Implementation in Games**

We can also implement this kind of system in online multiplayer games, whether it is for making game paid offers, calculating damage counters, giving special kind of damage, having different status effect, etc.

There are many uses of this kind of program if viewed from creation point of view.

**Work Distribution**

People works efficiently when proper work is given to proper person, and this stays true not only in terms of skills but also personality. People who are more extrovert are more efficient when it comes to meeting and negotiation than the people who are introvert, given that skills are same.

This program can predict personality traits and may be used to distribute work according the person’s skill and personality trait.

**Advertisements Recommendation**

Getting to know the personality of someone can be quite helpful when negotiating or making an offer or advertisements. If we can choose the favorable choice, the probability of success increases.

**SOURCE CODE**

# importing source files

import string

import re

import numpy as np

import pandas as pd

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.naive\_bayes import MultinomialNB

Dataset = pd.read\_csv('C:\\Users\\Aakash\\Downloads\\Personality Prediction AI\\MBTI.csv')

# adding separate field for each pair

Dataset['IvE'] = Dataset.Type

Dataset['NvS'] = Dataset.Type

Dataset['FvT'] = Dataset.Type

Dataset['PvJ'] = Dataset.Type

for i, t in enumerate(Dataset.Type):

if 'I' in t:

Dataset.IvE[i] = 'I'

elif 'E' in t:

Dataset.IvE[i] = 'E'

if 'N' in t:

Dataset.NvS[i] = 'N'

elif 'S' in t:

Dataset.NvS[i] = 'S'

if 'F' in t:

Dataset.FvT[i] = 'F'

elif 'T' in t:

Dataset.FvT[i] = 'T'

if 'P' in t:

Dataset.PvJ[i] = 'P'

elif 'J' in t:

Dataset.PvJ[i] = 'J'

Posts = Dataset.Posts.values

IE = Dataset.IvE.values

NS = Dataset.NvS.values

FT = Dataset.FvT.values

PJ = Dataset.PvJ.values

# method for cleaning text

def PreProcessor(Texts):

Lemmatizer = WordNetLemmatizer()

Punctuation = list(string.punctuation)

StopWords = stopwords.words('english')

Processed\_Documents = []

for i, document in enumerate(Texts):

print('{0}/{1}'.format(i + 1, len(Texts)))

# tokenization

Tokens = regex.findall(document)

# skipping useless tokens

t\_regex = re.compile(r"[^a-zA-Z]")

document = []

for token in Tokens:

token = np.array(token)

token = np.unique(token[token != ''])

if len(token) > 0:

token = token[0].lower()

else:

continue

if re.search(t\_regex, token) == None and token not in StopWords:

token = Lemmatizer.lemmatize(token)

document.append(token)

document = ' '.join(document)

# skipping

if len(document) >= 0:

Processed\_Documents.append(document)

print()

return np.array(Processed\_Documents)

# cleaning and lemmatizing the dataset

Posts = PreProcessor(Posts)

# training the model

CountVec = CountVectorizer().fit(Posts)

X = CountVec.transform(Posts)

xIETrain, xIETest, yIETrain, yIETest = train\_test\_split(X, IE)

xNSTrain, xNSTest, yNSTrain, yNSTest = train\_test\_split(X, NS)

xFTTrain, xFTTest, yFTTrain, yFTTest = train\_test\_split(X, FT)

xPJTrain, xPJTest, yPJTrain, yPJTest = train\_test\_split(X, PJ)

IEModel = MultinomialNB().fit(xIETrain, yIETrain)

NSModel = MultinomialNB().fit(xNSTrain, yNSTrain)

FTModel = MultinomialNB().fit(xFTTrain, yFTTrain)

PJModel = MultinomialNB().fit(xPJTrain, yPJTrain)

# creating GUI window

import tkinter as TK

class GUI(TK.Frame):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent)

self.Base = TK.Frame(self, bg='#EEE')

self.Base.pack(fill='both', expand=True)

LabelList = [TK.Label(self.Base, bg='#DDD', text="Introvert VS Extrovert"), TK.Label(self.Base, bg='#DDD', text="Intuition VS Sensing"), TK.Label(self.Base, bg='#DDD', text="Feeling VS Thinking"), TK.Label(self.Base, bg='#DDD', text="Perceiving VS Judgement")]

self.Label = TK.Label(self.Base, bg='#EEE', text="Enter Your Text:")

self.Text = TK.Entry(self.Base, bg='#DDD', bd=0)

self.Button = TK.Button(self.Base, bg='#DDD', width=3, bd=0, text="Predict", command=self.Predict)

self.Model = [[TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD')], [TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD')], [TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD'), TK.Label(self.Base, bg='#DDD')]]

for i in range(8):

self.Base.rowconfigure(i, weight=1)

for i in range(4):

self.Base.columnconfigure(i, weight=1)

self.Label.grid(row=0, column=0, columnspan=2, padx=3, pady=3, sticky='nsw')

self.Text.grid(row=1, rowspan=3, column=0, columnspan=4, padx=3, pady=3, sticky='nsew')

self.Button.grid(row=0, column=3, padx=3, pady=3, sticky='nsew')

for i in range(4):

LabelList[i].grid(row=i+4, column=0, padx=3, pady=3, sticky='nsew')

for j in range(3):

self.Model[j][i].grid(row=i+4, column=j+1, padx=3, pady=3, sticky='nsew')

def Predict(self):

if(self.Text.get()!=""):

Result = CountVec.transform([self.Text.get()])

self.Model[1][0]['text'] = str(IEModel.predict\_proba(Result)[0][0])[:4]

self.Model[1][1]['text'] = str(NSModel.predict\_proba(Result)[0][0])[:4]

self.Model[1][2]['text'] = str(FTModel.predict\_proba(Result)[0][0])[:4]

self.Model[1][3]['text'] = str(PJModel.predict\_proba(Result)[0][0])[:4]

self.Model[2][0]['text'] = str(IEModel.predict\_proba(Result)[0][1])[:4]

self.Model[2][1]['text'] = str(NSModel.predict\_proba(Result)[0][1])[:4]

self.Model[2][2]['text'] = str(FTModel.predict\_proba(Result)[0][1])[:4]

self.Model[2][3]['text'] = str(PJModel.predict\_proba(Result)[0][1])[:4]

class Interface(TK.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.Screen = None

self.Switch(GUI)

def Switch(self, FrameClass):

NewFrame = FrameClass(self)

if self.Screen is not None:

self.Screen.destroy()

self.Screen = NewFrame

self.Screen.pack(fill='both', expand=True)

App = Interface()

App.title("Personality Prediction AI")

App.geometry("600x300+600+100")

App.resizable(width=False, height=False)

App.mainloop()

**SNAPSHOTS**

