Myers-Briggs Personality Prediction

Nalisha Rathod(gs9440)
Sujata Gorai (hc6837)

A. Overview of the Problem

Personality is a way person respond to a particular situation. It is combination of characteristics that make an individual unique. Assessment of personality over the past two decades in various research has revealed that personality can be defined by 4 dimensions known as MBTI personality traits. In general, study of personality considered as a psychology research based on the survey or questionnaire. But this limits the research data to less number of persons. Hence there is a need of something through which we can increase the number of people involved in survey and to make the process automated. Data from Online Social Networking Sites provides a solution to this problem. It has emerged as one of the most ubiquitous means of communication today. It allows individual to find like-minded ones, whether it be for romantic or social purpose. It is also being used to maintain existing social connections. Online interactions generated more self-disclosures and fostered deeper personal questions than did face-to-face conversations. Now-adays people analyze person's social profile before considering as business partner or before dating.

A.1 Literature Review

Machine learning for predicting personalities has been a significant focus in Natural Language Processing research over the past few years. This is in part due to the availability of large corpora of online social interactions. (MBTI) Myers-Briggs Personality Type Dataset published by Kaggle competitions which have allowed researchers to gain access to datasets with 8600 training examples of post by different people around the world. In terms of methods, most research takes a text classification approach for personality prediction.

Carl Jungs theory of different people having different state of mind types states that random disparity in behavior is accounted for by the way people use judgement and perception and this is what Myers-Briggs Type Indicator (MBTI) is based on.
There are 16 personality types across four dimensions. Extraversion (E) vs Introversion (I) is a measure of how much an individual prefers their outer or inner world. Sensing (S) vs Intuition (N) differentiates those that process information through the five senses versus impressions through patterns. Thinking (T) vs Feeling (F) is a measure of preference for objective principles and facts versus weighing the points of view of others.
Finally, Judging (J) vs Perceiving (P) differentiates those that prefer planned and ordered life versus flexible and spontaneous. Note that these measures are not binary but rather on a continuum.

Mohammad Hossein Amirhosseini and Hassan Kazemian in March 2020 "**Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator**" [1] used natural language processing toolkit (NLTK) and XGBoost approach to discover personality type prediction based on MBTI personality type indicator. They have used Pandas, NumPy, re, Seaborn, Matplotlib and Sklearn.

Personality prediction of Twitter users with Logistic Regression Classifier learned using Stochastic Gradient Descent - Journals Iosr, Sharma Kanupriya, Kaur Amanpreet [2] Published 2015 – used Logistic Regression Classifier with parameter regularization using stochastic gradient descent.

Predicting Myers-Briggs Type Indicator with Text Classification (2017) – Rayne Hernandez and Ian Scott Knight [3] – various types of recurrent neural networks (RNN). After testing the SimpleRNN, GRU, LSTM, and Bidirectional LSTM options for recurrent layers in Keras, they found the LSTM option to give the best results.

B. Dataset Overview

Our main data set is a publicly available Kaggle data set containing 8675 rows of data[11]. Each row consists of two columns: (1) the MBTI personality type (e.g. INTJ, ESFP) of a given person, and (2) fifty of that persons social media posts. Since there are fifty posts included for every user, the number of data points is 430,000. This data comes from the users of personalitycafe.com, an online forum where users first take a questionnaire that sorts them into their MBTI type and then allows them to chat publicly with other users.

	type	posts
0	INFJ	http://www.youtube.com/watch?v=qsXHcwe3krw h...
1	ENTP	I'm finding the lack of me in these posts very...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired. That's another silly misconce...
5	INTJ	'18/37 @.@ Science is not perfect. No scien...
6	INFJ	'No, I can't draw on my own nails (haha). Thos...
7	INTJ	'I tend to build up a collection of things on ...
8	INFJ	I'm not sure, that's a good question. The dist...
9	INTP	https://www.youtube.com/watch?v=w8-egj0y8Qs ...

Fig 1 – First 10 rows of our dataset

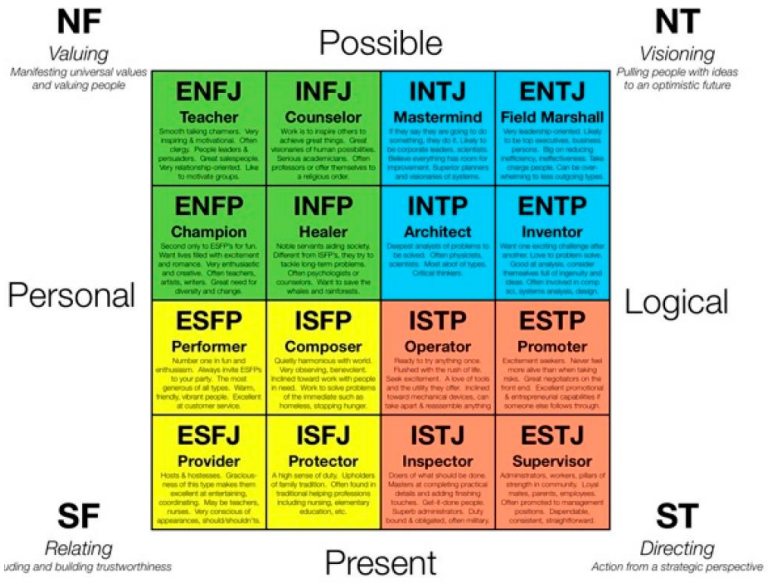

Fig 2 – 16 Personality Types

By combining the 4 Major personality type we can have a total of 16 different MBTI personality types. All 16 can be seen in Fig 2 along with a short description.

C. Methods

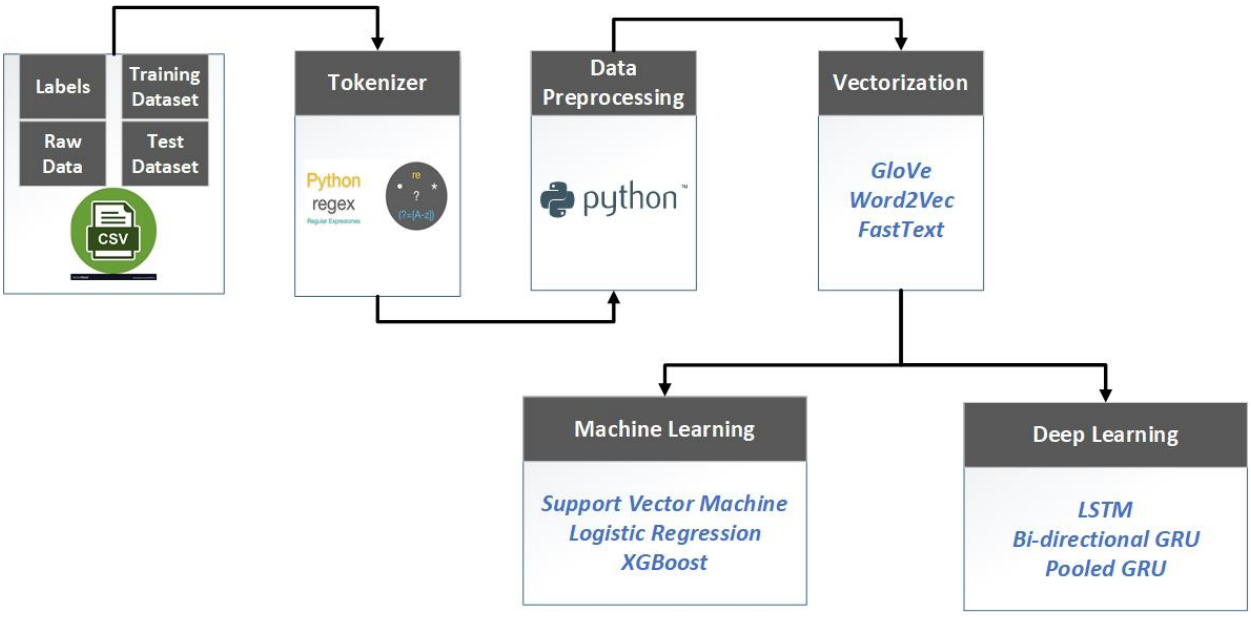

Fig 3-Our Work Strategy

In Figure 3, we can see our work strategy for building classifiers. We define our process as followings:

1. Import the Dataset
2. Break the Dataset
3. Tokenize the data
4. Data Pre-processing
5. Vectorization
6. Classification Models
   - Machine Learning Models
     - Support Vector Machine
     - Logistic Regression
     - XGBoost
   - Deep Neural Network Models
     - LSTM – Long Short-Term Memory
     - Bi-Directional GRU
     - Pooled GRU

C.1 Import the labeled Dataset

In each model, we have imported dataset using python. The dataset is on .csv format.[11].

C.2 Break the Dataset into 4 Major Classes

Below is the visualization of the data across all 16 MBTI types:

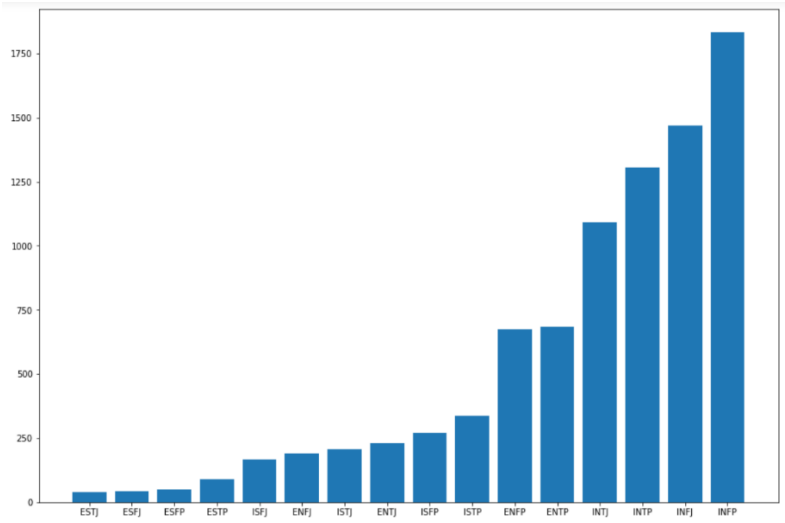

Fig 4: 16 Personality Types

As we can observe that the classes are highly imbalanced. The INFP has 1750 data whereas the ESTJ only has 39. Working with this dataset will produce unreliable results as the model will get biased to the class which has large amount of data.

We divided the Data into 4 major Classes E/I(Extraversion/Introversion), S/N(Sensing/Intuition), T/F(Thinking/Feeling and J/P(Judging/Perceiving) as is_E, is_S, is_T and is_J respectively. After splitting, each dataset has the same amount of data as in the original set i.e 8675.

	type	posts	is_E	is_S	is_T	is_J
0	INFJ	http://www.youtube.com/watch?v=qsXHcwe3krw h...	0	0	0	1
1	ENTP	I'm finding the lack of me in these posts very...	1	0	1	0
2	INTP	'Good one _____ https://www.youtube.com/wat...	0	0	1	0
3	INTJ	'Dear INTP, I enjoyed our conversation the o...	0	0	1	1
4	ENTJ	'You're fired. That's another silly misconce...	1	0	1	1
...	...	...	...	...	...	...
8670	ISFP	'https://www.youtube.com/watch?v=t8edHB_h908 ...	0	1	0	0
8671	ENFP	'So...if this thread already exists someplace ...	1	0	0	0
8672	INTP	'So many questions when i do these things. I ...	0	0	1	0
8673	INFP	'I am very conflicted right now when it comes ...	0	0	0	0
8674	INFP	'It has been too long since I have been on per...	0	0	0	0

Fig 5: Splitting the type into 4 major sections

This is the after the splitting of the dataset. If the person was defined as Extrovert in the original dataset, we classify it as 1 in the is_E dataset and if Introvert then as 0. Similarly, we classify the other classes of is_S, is_T and is_J. So, if a person is classified as ESFJ in the original dataset we break it as 1 in dataset is_E, 1 in dataset is_S, 0 in dataset is_T and 1 in dataset is_J.

After the splitting we have to check is they have dependency on each other i.e. if they are highly correlated.

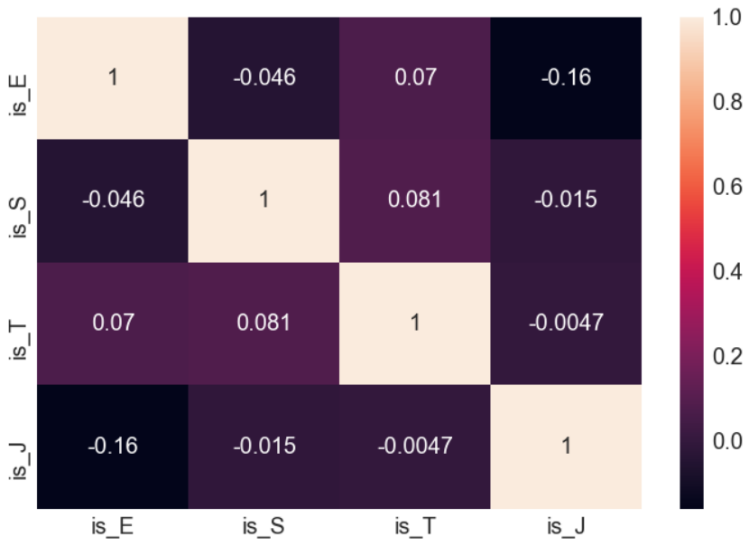

Fig 5: Correlation Plot

Above is the correlation plot for all 4 new attributes. The correlation values are really low and close to zero so we can deduce that they are not correlated.

Visualizing the new Datasets:

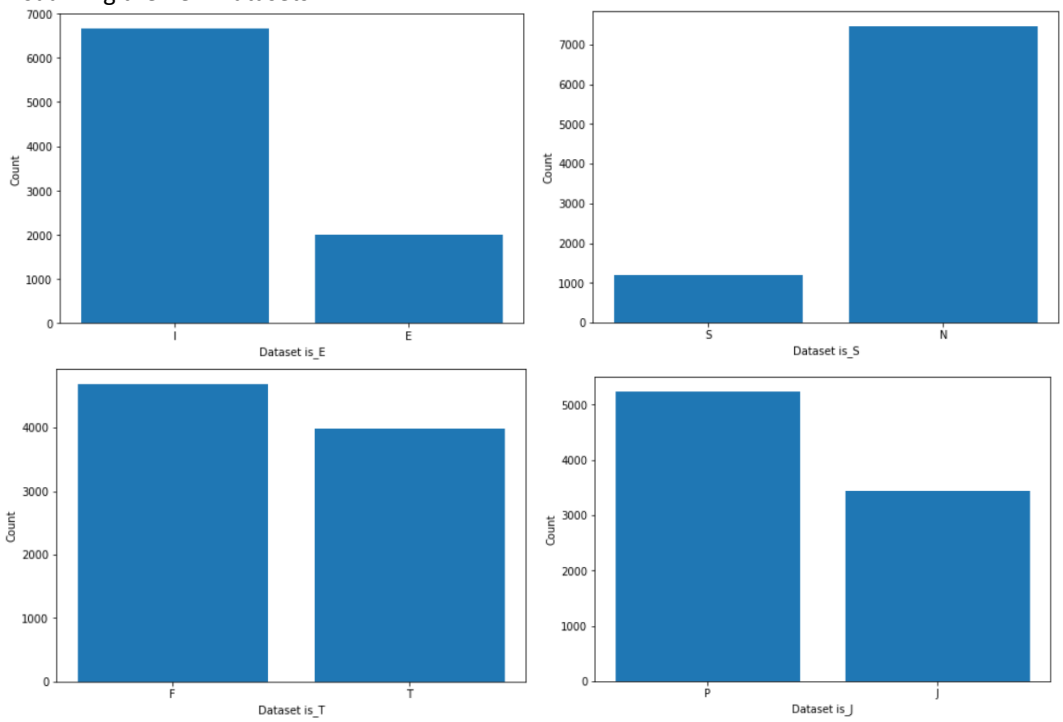

Fig 6: Dateset overview after splitting

As we can see from the plots that is_E and is_S ia still imbalanced whereas is_T and is_J are more or less balanced. So, we decided to experiment a imbalanced dataset and another balanced one and observe the outcome – for this we selected is_E as our imbalanced class and is_T as the balanced one for Machine Learning Models and is_S ans is_T for Deep Learning Models.

Then we divided both of the dataset into 3 parts – Training, Validation and Testing set into 60:20:20 ratio respectively.

C.2 Tokenize the Dataset

We use Tokenizer from regex library in python. The code snippet is given below:

```
# Tokenization - this is taken from SpaCy

re_tok = re.compile(f'([{string.punctuation}“”¨«»®´·º½¾¿¡§£€‘’])')

def tokenize(s):
    tokens = re_tok.sub(r' \1 ', s).split()
    return tokens
```

C.3 Data Pre-processing

We have done several methods to preprocess the comments before passing to the training model. Below are the cleaning we performed:

1. Remove 3 pipe ||| characters which were used to separate the 50 posts from a user
2. Make the sentence into lowercase.
3. Remove punctuation.
4. Remove words containing numbers.
5. Remove URLs.
6. Remove Mentions (@) and Hashtag (#)
7. Remove Stop words and the MBTI types that were mentioned in the post as it doesnot serve any purpose of classifying the personality.
8. Lemmatization using WordNetLemmatizer

C.4 Vectorization

In word vector representations, each word is represented by a vector which is concatenated or averaged with other word vectors in a context to form a resulting vector which is used to predict other words in the same context. These vectors allow capture word analogies, semantic associations and other hidden information about a language. In previous research, word vector representations have proved to boost the efficiency and accuracy of classification models. However, inconsistent performances are observed in some application contexts [7].

In this project, we explore three popular embedding models, namely, Word2Vec [8], GloVe [9] and FastText [10], combined with various classification models and tried to identify which combination of models work based for personality classification.

Here is a summary on how the models differ in principles.

Word2Vec was developed by Google in 2013 [8]. It is a group of related models based on two-layer neural networks that are trained to reconstruct linguistic contexts of words. Two model architectures can be used: continuous bag-of-words (CBOW) or continuous skipgram(SG). In CBOW architecture, the model predicts the current word from a window of surrounding context words. As in other bag-of-words approaches, the order of context words does not influence prediction. In the continuous SG architecture, the model uses the current word to define the surrounding window of context words. The SG architecture weights nearby context words more heavily than more distant context words.

GloVe (Global Vectors for Word Representation) was developed in 2014 by the Stanford University [9]. It allows the user to obtain word vector representations by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

FastText is an extension of Word2Vec proposed by Facebook in 2017 [10]. It is based on the SG model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram and words are represented as the sum of these vector representations.

Pre-trained word embeddings on large training sets are publicly available, such as those produced for Word2Vec, GloVe or Wiki word vectors for FastText.

For our project, we took the following steps for vectorization of user comments in machine learning models:

- Loaded the GloVe, Word2Vec aFastText vectors in a dictionary called embedded index.
- Defined a function called sent2vec which creates a normalized vector for a whole sentence.
- Created sentence vectors using the sent2vec function for training, validation and testing datasets and fed these vectorized datasets to the machine learning classification models.

```
#Word to vec embedding loading
from tqdm import tqdm

embeddings_index_word2vec = {}
f = open('C:/Sujata/CompScience/Winter 2021/Data Mining/Project Final/GoogleNews-vectors-negative300.txt', encoding='utf
for line in tqdm(f):
    values = line.split()
    word = values[0]
    try:
        coefs = np.asarray(values[1:], dtype='float32')
    except:
        continue
    embeddings_index_word2vec[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index_word2vec))
```

```
3000001it [05:14, 9540.41it/s]

Found 3000000 word vectors.
```

Fig 7: The code snippet for Word2Vec Word Embedding

```
def sent2vec(s, embeddings_index):
    words = str(s)
    words = tokenize(words)
    words = remove_stopwords(words)
    words = [w for w in words if w.isalpha()]
    M = []
    for w in words:
        w = wrl.lemmatize(w)
        try:
            M.append(embeddings_index[w])
        except:
            continue
    M = np.array(M)
    v = M.sum(axis=0)
    if type(v) != np.ndarray:
        return np.zeros(300)
    return v / np.sqrt((v ** 2).sum())
```

Fig 8: Code for Word Embeddings

We have also done the word embeddings for our deep learning models. The code snippet is of embeddings on deep learning models is shown in Figure 9.

```
def get_coefs(word, *arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.rstrip().rsplit(' ')) for o in open(EMBEDDING_FILE))

word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.zeros((nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

Fig 9: Code for Word Embedding for Deep Learning

C.5 Classification Models

We have used three conventional machine learning models and three deep neural network models to test performance. The model architectures are described in next sections.

C.5.1 Machine Learning Models

In NLP literature, linear classifiers have always stood as strong baselines for text classification problems. These state-of-the-art models have proved their suitability and their robustness when they are combined with right features. In this research, we have used traditional linear classification models like Support Vector Machine (SVM), Logistic Regression and XGBoost classifiers combined with GloVe, Word2Vec and FastText word embedding approaches. Nine models were explored for the Personality Prediction task. Each of the three word embedding models is coupled with each of the three linear classification models and their training (5 fold cross-validation) accuracy and testing accuracy are compared to find the best-performing model.

1. Support Vector Machine: A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. SVMs have proved to be helpful in text and hypertext categorization in document classification.

The first approach we took in this project was to implement the SVM algorithm, to provide sort of a baseline that we can work with. The steps taken in the implementation are as follows:

For is_E – Imbalanced Data

_ The normalized word embeddings for each sentence in the vectorized training dataset was fed as input into the scikit-learn's SGDClassifier model with 'Stochastic Gradient Descent' as optimizer, class weight as 'balanced' and 'hinge' as loss function, to make sure that the classifier works as a Support Vector Machine.

_ 5-fold cross-validation technique was used to determine the training accuracy score.

_ balanced_accuracy,F1 score and Recall as the scoring parameter (as it is imbalanced)

_ After fitting the vectorized training dataset into the SGDClassifier model, we calculated all the 3 scores using the vectorized validation dataset. We can see the code snippet of SVM model using 'hinge' loss in figure below:

```
# SVM Done for is_E class
col = ['is_E']
preds = np.zeros((X_validate_IE_clean.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    print('fit '+ class_name)
    classifier = SGDClassifier(loss='hinge', max_iter=1000, epsilon=0.001, n_jobs=-1, class_weight='balanced')

    cv_score = np.mean(cross_val_score(classifier, xtrain_IE_word2vec, Y_train_IE[class_name], cv=5,
                                       scoring='balanced_accuracy'))
    print('CV score for class {} is {}'.format(class_name, cv_score))

    classifier.fit(xtrain_IE_word2vec, Y_train_IE[class_name])

    val_score = classifier.score(xvalid_IE_word2vec, Y_validate_IE[class_name])
    print('Validation score for class {} is {}'.format(class_name, val_score))
```

Fig 10: SVM Model code with Word2Vec embedding

Figure 10 shows the code snippet for our SVM model with Word2Vec embeddings and balanced_accuracy as scoring parameter.

For is_T – Balanced Data

Same parameters were used for SVM for the Balanced class only the scoring parameter was changed accuracy and roc_auc(Area under the curve).

```
# SVM Done for is_T class
col = ['is_T']
preds = np.zeros((X_validate_FT.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    print('fit '+ class_name)
    classifier = SGDClassifier(loss='hinge', max_iter=1000, epsilon=0.001, n_jobs=-1, class_weight='balanced')

    cv_score = np.mean(cross_val_score(classifier, xtrain_FT_glove, Y_train_FT[class_name], cv=5, scoring='accuracy'))
    print('CV score for class {} is {}'.format(class_name, cv_score))

    classifier.fit(xtrain_FT_glove, Y_train_FT[class_name])

    val_score = classifier.score(xvalid_FT_glove, Y_validate_FT[class_name])
    print('Validation score for class {} is {}'.format(class_name, val_score))
```

Fig 11: Code for SVM Model for balanced dataset with GloVe embedding

This same procedure was repeated for the other two types of word embedding models used in this research, i.e. GloVe and FastText.

2. Logistic Regression: Logistic Regression is another very commonly used supervised machine learning model for two-group classification problems and has been widely used for text classification and sentiment analysis in the literature. Next, we implemented the Logistic Regression Classifier to compare its performance with SVM and see if it works better for predicting personalities.

The steps taken in the implementation are similar to the SVM implementation and are as follows:

For is_E – Imbalanced Data

- The normalized word embeddings for each sentence in the vectorized training dataset was fed as input into the scikit-learn's LogisticRegression classification model with 'Stochastic Average Gradient descent (sag)' as optimizer and class weight as 'balanced'.
- 5-fold cross-validation technique was used to determine the training accuracy score.
- balanced_accuracy, F1 score and Recall as the scoring parameter (as it is imbalanced
- After fitting the vectorized training dataset into the LogisticRegression classifier, we calculated all the 3 scores using the vectorized validation dataset.

```
# is_E logistic regression
col = ['is_E']
preds = np.zeros((X_validate_IE_clean.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    print('fit '+ class_name)
    classifier = LogisticRegression(C=0.1, solver='sag', class_weight='balanced', max_iter=1000)

    cv_score = np.mean(cross_val_score(classifier, xtrain_IE_fasttext, Y_train_IE[class_name], cv=5,
                                       scoring='f1'))
    print('CV score for class {} is {}'.format(class_name, cv_score))

    classifier.fit(xtrain_IE_fasttext, Y_train_IE[class_name])

    val_score = classifier.score(xvalid_IE_fasttext, Y_validate_IE[class_name])
    print('Validation score for class {} is {}'.format(class_name, val_score))
```

Fig 12: Code for Logistic regression for imbalanced class with FastText embedding

Figure 12 shows the code snippet for our LR model with FastText embeddings. This same procedure was repeated for the other two types of word embedding models used in this research, i.e. GloVe and FastText.

For is_T – Balanced Data

All the other parameters were same as before only the scoring parameter is changed to accuracy and roc_auc because we are dealing with a balanced dataset.

```
# is_T logistic regression
col = ['is_T']
preds = np.zeros((X_validate_FT_clean.shape[0], len(col))).astype(object)

for i, class_name in enumerate(col):
    print('fit '+ class_name)
    classifier = LogisticRegression(C=0.1, solver='sag', class_weight='balanced', max_iter=1000)

    cv_score = np.mean(cross_val_score(classifier, xtrain_FT_fasttext, Y_train_FT[class_name], cv=5, scoring='roc_auc'))
    print('CV score for class {} is {}'.format(class_name, cv_score))

    classifier.fit(xtrain_FT_fasttext, Y_train_FT[class_name])

    val_score = classifier.score(xvalid_FT_fasttext, Y_validate_FT[class_name])
    print('Validation score for class {} is {}'.format(class_name, val_score))
```

Fig 13: Code for Logistic Regression for balanced data with FastText embedding

3. XGBoost: XGBoost (XGB) stands for eXtreme Gradient Boosting and is an implementation of gradient boosting machines that pushes the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed. It is one of the most powerful available classifiers and has proved to be highly efficient in document classification over the literature. XGBoost offers several advanced features for model tuning, computing environments and algorithm enhancement. It can perform the three main forms of gradient boosting (Gradient Boosting (GB), Stochastic GB and Regularized GB) and it is robust enough to support fine tuning and addition of regularization parameters.

We have decided to use XGBoost for this project and evaluate its performance with respect to our baseline models SVM and LR. The steps taken in the implementation of XGB are as follows:

For is_E – Imbalanced Data

Before running XGBoost, we must set three types of parameters: general parameters, booster parameters and task parameters. We created a python function for XGBoost training in which, we set the important parameters as below:

- 'booster' is set to gbtree in order to use treebased models for gradient boosting.
- 'eta' or the learning rate is set to 0.1.
- 'max depth' is set to 6. Increasing this value will make the model more complex and more likely to overfit.
- 'min child weight' is set to 1. This is the minimum sum of instance weight(hessian) needed in a child node to continue its partitioning.
- 'subsample' (subsample ratio of the training instances) is set to 0.7.
- 'colsample bytree' (subsample ratio of columns when constructing each tree) is set to 0.7.
- 'objective' (learning objective) is set to binary: logistic, which is logistic regression for binary classification with output probability
- 'scale_pos_weight' – 0.3 this says the distribution of the classes in the dataset if it is highly imbalanced (total -ve classes/ total +ve classes)
- The normalized word embeddings for each sentence in the vectorized training dataset (for model training) and validation dataset (for model testing) was fed as input into the XGBoost classification model with number of rounds as 500 and early stopping rounds as 20. When the validation accuracy does not change considerably for 20 rounds, the XGB model stops learning.

```
# is_E XGBoost Imbalanced data

def runXGB_U(train_X, train_y, test_X, test_y=None, feature_names=None, seed_val=2017, num_rounds=500):
    param = {}
    param['objective'] = 'binary:logistic'
    param['eta'] = 0.1
    param['max_depth'] = 6
    param['verbosity'] = 0  # look into it
    param['eval_metric'] = 'auc'
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    param['scale_pos_weight'] = 0.3 # this says the distribution of the classes (total -ve classes/ total +ve classes)
    num_rounds = num_rounds

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)

    if test_y is not None:
        xgtest = xgb.DMatrix(test_X, label=test_y)
        watchlist = [ (xgtrain,'train'), (xgtest, 'valid') ]
        model = xgb.train(plst, xgtrain, num_rounds, watchlist, early_stopping_rounds=20)
    else:
        xgtest = xgb.DMatrix(test_X)
        model = xgb.train(plst, xgtrain, num_rounds)

    return model
```

```
col = ['is_E']
preds = np.zeros((X_test_IE.shape[0], len(col)))

for i, j in enumerate(col):
    print('fit '+j)
    model = runXGB_U(xtrain_IE_glove, Y_train_IE[j], xvalid_IE_glove, Y_validate_IE[j])
    preds[:,i] = model.predict(xgb.DMatrix(xtest_IE_glove), ntree_limit = model.best_ntree_limit)
    gc.collect()
```

Fig 14: Code for XGBoost with imbalanced data and GloVe embedding

Figure 14 shows the code snippet for our XGBoost model with GloVe embeddings. This same procedure was repeated for the other two types of word embedding models used in this research, i.e. Word2Vec and FastText.

For is_T – Balanced Data

Similarly, we run the XGBoost model on the Balanced data but just remove the scaled_pos_weight parameter as it won't be necessary for a balanced data.

```
# is_T XGBoost Balanced data

def runXGB_B(train_X, train_y, test_X, test_y=None, feature_names=None, seed_val=2017, num_rounds=500):
    param = {}
    param['objective'] = 'binary:logistic'
    param['eta'] = 0.1
    param['max_depth'] = 6
    param['verbosity'] = 0  # look into it
    param['eval_metric'] = 'auc'
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    num_rounds = num_rounds

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)

    if test_y is not None:
        xgtest = xgb.DMatrix(test_X, label=test_y)
        watchlist = [ (xgtrain,'train'), (xgtest, 'valid') ]
        model = xgb.train(plst, xgtrain, num_rounds, watchlist, early_stopping_rounds=20)
    else:
        xgtest = xgb.DMatrix(test_X)
        model = xgb.train(plst, xgtrain, num_rounds)

    return model
```

```
col = ['is_T']
preds = np.zeros((X_test_FT.shape[0], len(col)))

for i, j in enumerate(col):
    print('fit '+j)
    model = runXGB_B(xtrain_FT_glove, Y_train_FT[j], xvalid_FT_glove, Y_validate_FT[j])
    preds[:,i] = model.predict(xgb.DMatrix(xtest_FT_glove), ntree_limit = model.best_ntree_limit)
    gc.collect()
```

Fig 15: Code for XGBoost for balanced data with GloVe embedding

C.5.2 Deep Neural Network Models

We have used three deep neural network models for our dataset after the preprocessing it. The models are:

1. Long Short-Term Memory (LSTM)
2. Pooled Gated Recurrent Unit (GRU)
3. Bidirectional Gated Recurrent Unit (GRU)

Deep Learning

Deep Learning is a subset of Machine Learning. It is a type of Machine Learning technique motivated by our Human Brain. Deep learning is just a term which describes certain types of Neural Networks (NN). It tries to work according to how a human brain can work. Both can learn and become expert in the area. Just like on our lifetime we see different things and learn it; the same way Neural Network learns when feed data into it.

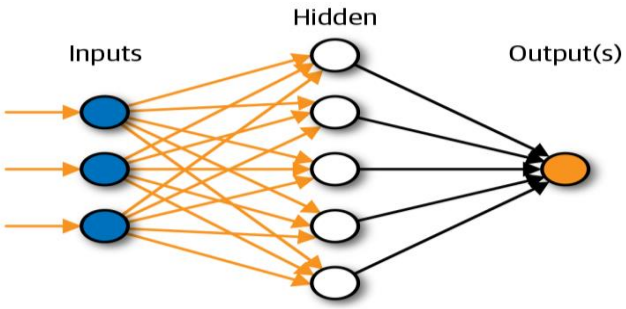

Fig 16: Artificial Neural Network

The NN process the data through layers of non-linear transformation of an Input data in order to calculate the output. A NN can consist of as many Inputs with as many numbers of Hidden Layers and then finally after processing we get the out depending upon the type of input.

It is used in Natural Language Processing, Visual Recognition, Fraud Detection, etc.

The problem with NN or even convolutional neural network (type of NN) it works for fixed grids or dimensions, so the input and output is fixed. Images of varying dimensions cannot be fed to CNN or NN and also there is no long-term dependency among the data.

Recurrent Neural Network (RNN)

- To overcome this limitation Recurrent neural networks (RNN) were based on David Rumelhart's work in 1986 as developed.
- RNN captures the sequential information present in the Input data i.e., dependency among the words in the text while making assumptions.

- But the drawback of RNN was it has vanishing gradient problem. We cannot train the network properly and this causes the RNN sequences not to retain in long term memory.
- To overcome the problems of RNN, Long Short Term Memory (LSTM) was proposed by Sepp Hochreiter and Jurgen Schmidhuber [5].
- For example: The sky is _____ (answer is blue)

  In the above sentence RNN model can easily predict that the answer can be "blue". This is because the sequence of the sentence is not too long.

- But the in cases where the sentence sequence is too long like for eg:
  Fuji lived in Japan for 15 years. He loved the Japanese culture, and he is very keen of the people there. He is fluent in ____ (answer should be Japanese)

  This above sentence cannot be predicted by RNN as it's a long sentence as it cannot remember relation between ling term sequences. In order to solve these kind of problems LSTM was introduced.

1. Long Short-Term Memory (LSTM)

LSTM consist of Input Gate, Forget Gate and Output Gate.

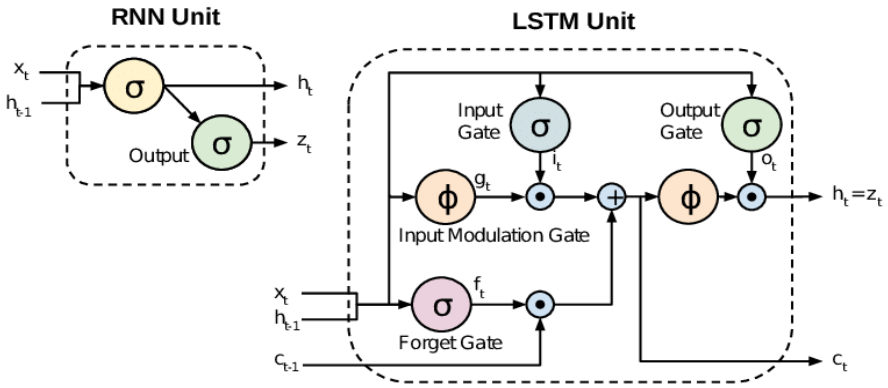

Fig 17: RNN and LSTM

In the above figure, we can see the comparison of LSTM with RNN. LSTM outdoes RNN in long sequence of text. As the MBTI dataset has posts and we wanted to do long sequence test classification, LSTM was the better choice than RNN. We have classified the post into 4 different personality types as discussed in the Data preprocessing section. (i.e., "is_E", "is_T", "is_S","is_J") where LSTM can detect long sequences for each category of personality.

1) **glove.840B.300d** for word embedding was used
2) The parameters are fitted as:

-embed size = 50 (how big is each word vector)
-max features = 20000 (how many unique words to use)
-maxlen = 100
-loss='binary crossentropy'
-optimizer='adam'
-metrics=['accuracy'].

binary crossentropy' is used as a loss function as we have classified it into0,1 for each personality type,'binary crossentropy' performs well which this type of classification.

After obtaining the cleaned data from the data processing done above up too to step where the “posts” are cleaned and the personality types are divided into 4 types, we have prepossessed the data with padding and tokenization using Keras in Tensorflow.

The code snippet is attached in Figure below:

```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(list_sentences_train))
list_tokenized_train = tokenizer.texts_to_sequences(list_sentences_train)
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

Fig 18: Tokenizer code for deep learning

LSTM Preprocessing

We have also added a dropout Layer in out model. Dropout layer is used to reduce the problem of overfitting. In this model, we set one embedding layer, then bidirectional LSTM, two pooling layers before dropout layer.

2. Bidirectional Gated Recurrent Unit (GRU)

Gated Recurrent Unit is the modification of Recurrent network and the hidden Layer.It is similar to LSTM and it is much better at capturing long range connection and helps a lot with vanishing gradient problems.

It was invented in 2014 by Kyunghyun Cho et al. It is closely related to LTSM with lower parameter and easy to compute.

It is utilizing the gating mechanism as same as LSTM to manage and control the flow of information between the cells in the NN.

GRU has two gates: 1) Update Gate 2) Reset Gate

Gates helps in determining what has to be retained/passed or dropped.

Update Gate: This gate helps in processing how much of the information needs to be maintained i.e., to be passed along to the future.

Reset Gate: This gate is used from the model o decide how much of the past information to forget.

A unidirectional GRU characteristically reads the input sequence from one direction. A bidirectional GRU consists of 2 vanilla unidirectional GRUs stacked side by side, but the second GRU reads the input sequence from the opposite direction

In this model, we have used glove.840B.300d for word embedding. Then we put max features=100000 (maximumfeatures) maxlen=150 (maximum length)
embed size=300 (size of embedding)

Preprocessing as followings:
Did tokenization and padding of each post and then convert it vector by using word embeddings. Bi-Directional GRU architecture is depicted in figure below. In this Bidirectional GRU model, we have set two pooling layers (average pooling and max pooling) layer after bidirectional GRU layer. Then we set dense layer. The batch size and number of epochs are 128 and 2 respectively. We have set binary cross entropy as loss function. We have done our experiment with 90% labeled data as training and 10% data as testing.

```
batch_size = 128
epochs = 2
X_tra, X_val, y_tra, y_val = train_test_split(x_train, y_train, train_size=0.9, random_state=233)
```

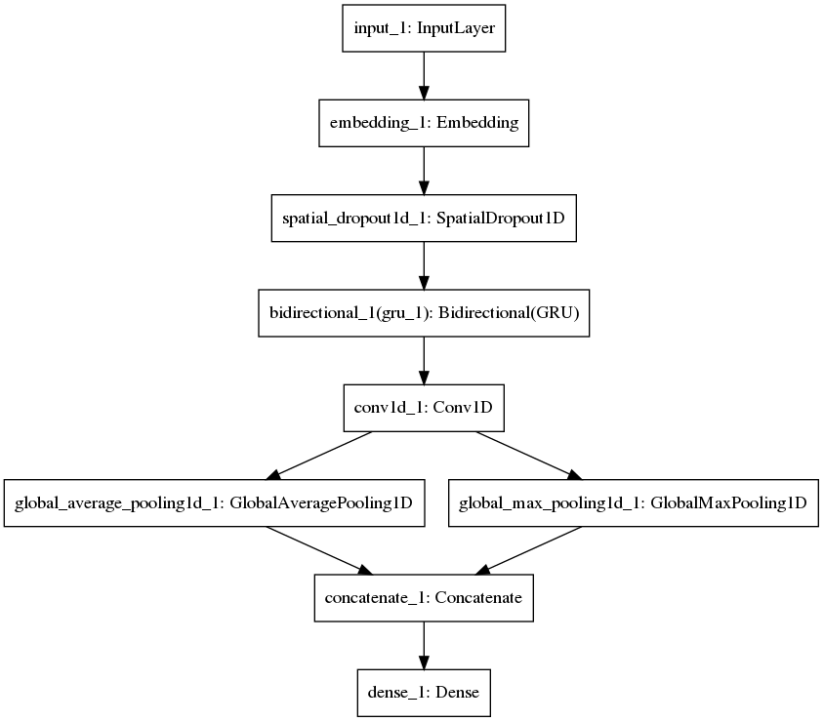

Fig 19: Bidirectional GRU model

3. Pooled Gated Recurrent Unit (GRU)

Pooled GRU is a simple GRU model with pooling layers.

It is used to reduce the number of dimensions of the feature matrix. Thus it reduces the number of parameters to learn and the amount of computation performed in the network.

We also add dropout layer for reducing the overfitting problem.

For Preprocessing of Data

Did tokenization and padding of each post and then convert it vector by using word embeddings. Bi-Directional GRU architecture is depicted in figure below. In this Bidirectional GRU model, we have set two pooling layers (average pooling and max pooling) layer after bidirectional GRU layer. Then we set dense layer. The batch size and number of epochs are 32 and 2 respectively. We have set binary cross entropy as loss function. We have done our experiment with 90% labeled data as training and 10% data as testing.

```
X_tra, X_val, y_tra, y_val = train_test_split(x_train, y_train, train_size=0.95, random_state=233)
RocAuc = RocAucEvaluation(validation_data=(X_val, y_val), interval=1)

hist = model.fit(X_tra, y_tra, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val),
                 callbacks=[RocAuc], verbose=2)
```

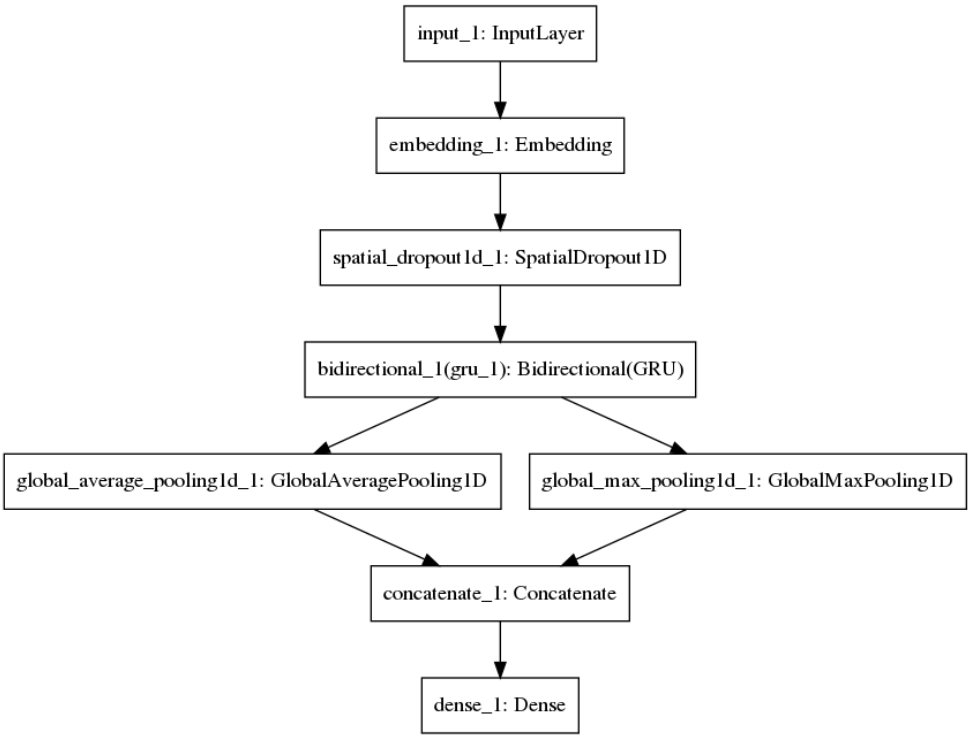

Fig 20: Pooled GRU model

D. Analysis Results and Comparison

We have done validation on training dataset. The training dataset is labeled. For the imbalanced data we used F1 score, Recall and Balanced_accuracy as our measurements and for the balanced data we measure the results in Accuracy and Receiver Operating Characteristic (ROC) curve. The Area of Curve (AUC) is an effective measurement for binary classification. It plots the true positive rate against the false positive rate in various thresholds. So, we compare our models using the AUC values from each model output.

Environment Setup: We have done our experiments using following environments:
pyhton=3.7
tensorflow 2.1
Jupyter Notebook

D.1 Machine Learning Models

The performance of three machine learning models with three different word embeddings are shown in Figure16. We have done total 9 experiments for three machine learning model with three different embeddings for balanced and 9 more with same combination for imbalanced data.

1. is_E – Imbalanced dataset:

	SVM	Logistic Reg	XGBoost
Word2vec	bal acc:0.783 F1: 0.7596 Recall: 0.7579	bal acc: 0.619 F1: 0.6195 Recall: 0.622	Acc: 0.768 AUC: 0.682
GloVE	bal acc: 0.752 F1: 0.696 Recall: 0.757	bal acc: 0.612 F1: 0.611 Recall: 0.613	Acc: 0.763 AUC: 0.671
FastText	bal acc: 0.746 F1: 0.748 Recall: 0.746	bal acc: 0.628 F1: 0.628 Recall: 0.627	Acc: 0.773 AUC: 0.691

Fig 21: Results for imbalanced data for Machine Learning

Above are the results – Balanced_accuracy, F1 score and Recall for SVM and Logistic Regression. For XGBoost we calculated the Accuracy and the AUC score. For the imbalanced data, between SVM and Logistic Regression – SVM performed the best. As we calculated accuracy for all the models XGBoost withFastText word embedding gave the best result of 77.3%.

2. is_T – Balanced dataset:

	SVM	Logistic Reg	XGBoost
Word2vec	Acc: 0.778 AUC: 0.777	Acc: 0.71 AUC: 0.709	Acc: 0.76 AUC: 0.85
GloVE	Acc: 0.771 AUC: 0.767	Acc: 0.693 AUC: 0.693	Acc: 0.745 AUC: 0.83
FastText	Acc: 0.776 AUC: 0.783	Acc: 0.701 AUC: 0.717	Acc: 0.783 AUC: 0.853

Fig 22: Results for Balanced data for Machine Learning

Above is the Accuracy and AUC results for the SVM, Logistic regression and XGBoost. The highest accuracy we got is 78.3% from the XGBoost Model with FastText as its word embedding. This same model also has the highest AUC(Area Under the Curve) 85.3%. So, we can conclude among the 9 models **XGBoost performed the better overall and with FastText gave the best result**.

D.2 Deep Learning Models

We have applied deep learning models and compared the results among them. In Table below, we can see the performance of our models where T is the Balanced Data and S is the Imbalanced one.

	LSTM + GloVe(T)	LSTM + GloVe(S)	Bidirectional GRU for T	Bidirectional GRU for S	Pooled GRU + FastText for T	Pooled GRU + FastText for S
Accuracy	0.8694	0.6287	0.8645	0.6752	0.8593	0.6982
Precision	0.744	0.663	0.7083	0.5869	0.6714	0.6259
ROC Curve	0.8517	0.7187	0.7815	0.7724	0.7714	0.7622

We have split the train dataset into 90% as training size and 10% as validation size.

Overall, LSTM gives the best performance with AUC score 0.8694% which is more than 85% accurate for the balanced class “is_T”.

LSTM also gives the best result for loss. It minimizes the loss to 0.5451 and Pooled GRU gives maximum loss 0.573 for balanced class “is_T. Bidirectional GRU gives maximum loss of 0.4399.

D.3 Result Analysis

In this work, we compare three machine learning models and three deep learning models. Among all the models, XGBoost gives the best performance for this specific dataset. For that reason, we are going to propose to use XGBoost model to do the classification of personalities. We also analyzed our results. We have found that in this case traditional machine learning gave good performance in text classification where the deep learning models fell behind. Moreover, we use different word embedding techniques. Those techniques help us to improve the accuracy of our models. FastText embeddings provides **well dictionary** for this personality prediction.

We compared our outcome with the results with that in the paper - “**Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator**”[1]. They used the same dataset and XGBoost as their model. Below is the screenshot of their results:

Binary Class	MBTI Personality Type	Accuracy of Extreme Gradient Boosting
IE 10.75%	Introversion (I)–Extroversion (E)	78.17%
NS 24.06%	Intuition (I)–Sensing (S)	86.06%
FT 6.02%	Feeling (F)–Thinking (T)	71.78%
JP 2.0%	Judging (J)–Perceiving (P)	65.70%

Fig 29: Result from Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator

For our XGBoost Model with FastText for is_E is 77.1% which is quite close to their result. Now for the balanced data is_T our outcome is 78.3% which is outperforms their result.

Another paper which used a different data - Twitter posts and applied Logistic Regression model for prediction of MBTI personality – **"Personality prediction of Twitter users with Logistic Regression Classifier learned using Stochastic Gradient Descent"**. Below are their results:

Number of features	Accuracy of Naïve Bayes	Accuracy of Proposed Model
100	68.91	70.25
500	71.48	72.24
1000	74.39	76.78
1500	79.21	81.21
10000	83.36	84.45
15000	85.19	87.84
20000	88.57	89.66

Fig 30: Results from above paper [2] for Logistic Regression

In the above table the column which is labeled as "Accuracy of Pproposed Model" is the results of the Logistic Regression Model. With our every word embedding being of 300 Dimension the closest result we can compare is with their 500 features. For 500 features their Accuracy is 72.24% whereas our accuracy with 300 features is 71.7% which is very close to them with fewer features. If we try our model with higher dimension word embedding, we are hopeful that it will increase our accuracy and might exceed theirs.

We couldn't find any papers for comparing the deep learning models which used our datset and has done approached the data preprocessing as we did for this project. But overall if we were to use deep learning model, LSTM with Glove word embedding works the best. But overall, we got XGBoost with FastText embedding gives the best result among 6 models.

D.4 Limitations and Future Work

- The Natural language processing (NLP) classification to domain specific.
- The meaning of text may defer for different domain.
- Soo we are not sure if out model can perform well for different domain dataset.
- In Future we would like to evaluate our model on different dataset to see the accuracy and performance.
- We would also like to incorporate different other models to make it more reliable.
- Also use better data preprocessing techniques along with making the dataset balanced for all the classes to have much better accuracy.

E. Conclusion

We have implemented six different models to perform the classification of different personality types. We have found that XGBoost models outperform the deep neural network learning models. Moreover, XGBoost with FastText embedding gives the best result among 6 models. Before passing the comments to each, we work on different prepossessing to find a good result. We have compared the models by using Accuracy and AUC score for the balanced data whereas for the imbalanced one we used F1 score, Recall and balanced_accuracy as metrics.

When compared with 2 other papers **Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator** and **Personality prediction of Twitter users with Logistic Regression Classifier learned using Stochastic Gradient Descent** we performed better with XGBoost than the former mentioned research and for Logistic Regression – the later paper we performed almost similar.

We propose to use XGBoost model for Personality Prediction using MBTI dataset.

Repository

Link [6] to get our code and output.

Member Contribution

- Nalisha Rathod (GS9440) worked on literature review, Data preprocessing for Deep learning models and building deep neural network models.
- Sujata Gorai(hc6837) worked on Preprocessing the dataset and building conventional machine learning models.
- Both of us worked on analysis part and Literature Review.

References

[1]"**Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator**" - Mohammad Hossein Amirhosseini and Hassan Kazemian in March 2019

[2] **Personality prediction of Twitter users with Logistic Regression Classifier learned using Stochastic Gradient Descent -** Journals Iosr, Sharma Kanupriya, Kaur Amanpreet- Published 2015

[3] **Predicting Myers-Briggs Type Indicator with Text Classification (2017)** – Rayne Hernandez and Ian Scott

[4] **Zhanchen Reg , Qiang Shen , Xiaolei Diao ,Hao Xu in January 2021 "A sentiment-aware deep learning approach for personality detection from text**"

[5] Sepp Hochreiter and Ju¨rgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] https://github.com/nalisharathod01/Myers-Briggs-Personality-Prediction

[7] Anaıs Ollagnier and Hywel Williams. **Classification and event identification using word embedding. neural networks**, 6:7, 2019.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. **Efficient estimation of word representations in vector space.** arXiv preprint arXiv:1301.3781, 2013.

[9] Jeffrey Pennington, Richard Socher, and Christopher D Manning. **Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)**, pages 1532–1543, 2014.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. **Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics**, 5:135–146, 2017.10

[11] **(MBTI) Myer-Brigs Personality Type Dataset – Kaggle -** https://www.kaggle.com/datasnaek/mbti-type