Myers Briggs Personality Type Text Classification

Brandon Mabey V00795378
Brandon Vickery V00832266
Jake Runzer V00797175
Tania Akter V00810640

***Abstract*— The Myers Briggs type indicator personality system is a popular way of classifying personalities into 16 unique classes along 4 axes. Using a publicly available dataset on Kaggle, consisting of 433750 text posts from 8675 users on the Personality Cafe Forum, we have trained 4 machine learning models to classify text data. An Elman network, SVM, and various forms of Naive Bayes perform 16 class classification on the data. The Elman network achieves an accuracy of 22% which is equivalent to always predicting the most probable class. The average accuracy for the Naive Bayes models is slightly higher at 26%. The SVM achieves an accuracy of 18%. We also trained an LSTM to perform binary classification on each of the four character axes using balance data. The four models all achieved an accuracy of 83-84%.**

I. INTRODUCTION

The first version of the *Briggs Myers Type Indicator Handbook* was published in 1944 [1]. Katharine Cook Briggs and her daughter Isabel Briggs Myers thought a formal understanding of personality types would help women who were entering the workforce for the first time find a wartime job were they would be the "most comfortable and effective". Fast forward to 2018 and the Myers Briggs Type Indicator (MBTI) is still used as a way to of indicating different psychological preferences in how people perceive the world around them and make decisions [1].

The MBTI consists of a questionnaire which yields a four character personality code. In total there are 16 distinct personality types across 4 axes. In the rest of this paper, we call each of these axes a character set. The character sets are

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

Figure 1 shows a description of the 16 classes.

Our group proposed a project of classifying text into into the 16 possible MBTI personality types. We trained several different models on the last 50 online text posts of 8675 individuals to see which gave the best results, including three variants of naive Bayes, an SVM, and two different neural nets. This report discusses various aspects of our project, including related work, our data source, proposed algorithm, and distribution of tasks among team members.

II. DATASET

The dataset we used was completely provided by Kaggle [3]. Kaggle is an online platform for predictive modelling and analytics competitions. They provide thousands of publicly available datasets in many different categories. They encourage use of this data to build and test machine learning models. One of these is the *(MBTI) Myers-Briggs Personality Type Dataset* [4]. The dataset consisted of 8675 rows of an individual's Myers-Briggs personality type code in one column, paired with the last 50 things they have posted online in another column. In total there were 433,750 online posts with a labelled personality type. The post length differs from a few words to a short paragraph. The data was collected by the Personality Cafe Forum [5].

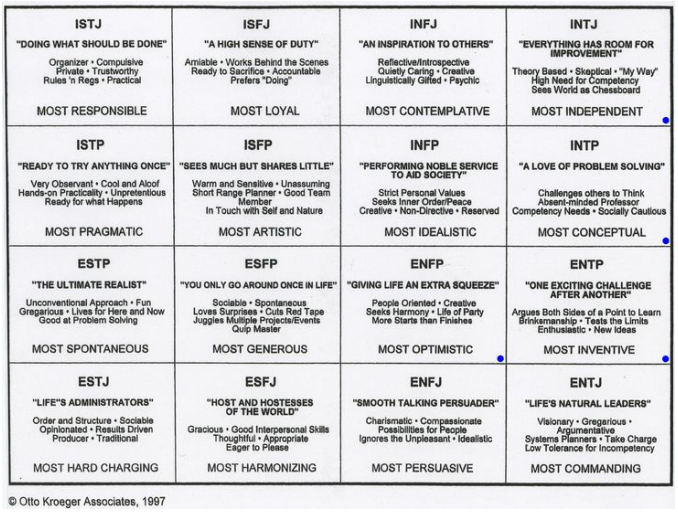

Fig. 1. MBTI Personality Type Descriptions [2]

The number of posts for each personality type was not evenly distributed per personality type as the general population was unevenly distributed per type. Table I shows how the population is distributed across all 16 personality types [6]. We expected the data to be distributed in a similar way.

We pre-processed the data to avoid overfitting the most common personality types because of the unbalanced dataset, so that the algorithms under consideration perform optimally. Different techniques of balancing, include but not limited to, changing the performance metric, generating synthetic samples, and under sampling the majority classes were considered.

Additionally, pre-processing techniques not focused on the balance of the dataset was also done in order to improve the accuracy of the final model. One example is restricting the

Personality Type	Percent of Population
ISFJ	13.8
ESFJ	12.3
ISTJ	11.6
ISFP	8.8
ESTJ	8.7
ESFP	8.5
ENFP	8.1
ISTP	5.4
INFP	4.4
ESTP	4.3
INTP	3.3
ENTP	3.2
ENFJ	2.5
INTJ	2.1
ENTJ	1.8
INFJ	1.2

TABLE I
MYERS-BRIGGS PERSONALITY TYPE STATISTICS

feature size such that the model was forced to learn on fewer dimensions.

III. RELATED WORK

A. Neural Networks in Predicting Myers Brigg Personality Type From Writing Style

Students Anthony Ma and Gus Liu at Stanford University [7] used neural networks for a similar problem to the one we did. They proposed that an individuals writing style is tightly coupled to their MBTI personality type. However, instead of using online text posts to train their model, they used sentences obtained from novels written by authors with known MBTI labels. 10 full books were used for each of the 16 classes. In total they used 750,000 labelled sentences.

The main approach used in the paper was a recurrent neural network (RNN) with long short term memory (LSTM). They trained their model to classify the text into 16 unique classes. Their best model obtained an accuracy up to 37%. Unfortunately they did not attempt to perform binary classification with the text on each of four character sets.

A major difference between the dataset used by Ma and Liu and ours is that their data came from only ten individuals per class, but they used full books written by each individual. The dataset we have obtained had hundreds of samples per class, but the text length will be much shorter.

B. Detection of Myers- Briggs Type Indicator via Text Based Computer-Mediated Communication

Dan Brinks and Hugh White from Stanford University had done a similar project for Myers-Briggs Type indicators. They used almost 700 tweets from 4600 unique users[8]. Although the MBTI results were confidential, Brinks and White took leverage of the fact that the twitter users shared their results in variety of ways, one of them being "#INFP" in their tweets. Brinks and White used Twitter API to get tweets containing Myers-Briggs abbreviations [8].

They processed their normalised dataset by a few optional parameters such as porter stemming, substitution of emoticons into works, minimum token frequency, minimum user frequency, term frequency transform, and inverse document frequency transform [8]. After the processing, the data was fed into different model using 70/30 hold-out cross-validation.

Brinks and White initially used a Naive Bayes Classifier (multi-variate Bernoulli event model) and a custom built multinomial event model classifier, and later used a Support Vector Machine and logistic regression classifiers to verify the results. The Multinomial Event Model built by Brinks and White performed well on the training data but did not perform as well on the test data. The best model performed with a 74.6% accuracy on the N vs S character set [8].

It can be noted from their research that Twitter has a limit of 140 characters for all their posts. Brinks and White mentioned that the word limitations and the use of emoticons limits the number of indicator words in a post. Moreover, with sharing of URLs has caused many duplication of words across users and not many words were unique to a specific user [8]. However, the dataset we obtained from Kaggle showed a lot of variety in length and content for each users.

C. Predicting student personality based on a data-driven model from student behaviour on LMS and social networks

Halawa et. al published a paper in 2015 on creating a data model to identify individual and their personality type using data from the Moodle learning management system and the Facebook social network[9].

The trained model was applied on data collected from 240 students for testing. The test was done performed with 10 different classification algorithms which are: NaiveBayes, BayesNet, Kstar, Random forest, J48, OneR, JRIP, KNN /IBK, RandomTree, and Decision Table. It was found that OneR had the best accuracy of 97.40% followed by Random Forest and J48 with 93.23% and 92.19% respectively [9].

D. Predicting Personality with Social Media

Jennifer Golbeck, Cristina Robles, and Karen Turner from the university of Maryland attempted to predict the personality of people by using publicly available information on their facebook profile. Golbeck, Robles and Turner created and distributed a Facebook application to 279 subjects that administered a 45 question big five personality test and collected all the of the publicly available information on their profile. From the data collected from the subjects profile the main sources of data used were language features such as statuses, profile descriptions, etc. , personal information and activities and preferences. From the original 279 many had fewer than 10 words which is not enough for a text analysis and were eliminated bringing the the number of subjects down to 167[10].

They used a variety of regression algorithms with 10-fold cross-validation on 74 features per user. The algorithms that performed the best were M5Rules and gaussian processes with which they were able to predict each of the traits to within 11% of their actual value[10].

E. Using syntactic features to predict author personality from text

Kim Luyckx and Walter Daelemans used Memory-Based Shallow Parsing and TiMBL which is a memory based learning algorithm[11] to predict an author's Myers Briggs personality result. Their data consisted of 145 essays of about 1400 words by BA students, where all the essays are about a documentary on Artificial Life[11].

Luyckx and Daelemans were able to predict Feeling and Judging rather well with F-scores of 85% and 90% respectively, for Introverted, Extroverted, intuitive, and Sensing they were able to achieve moderate results with F-scores between 64% and 73%. However with Thinking and Perceiving their results were poor with F-scores of 44% and 38% respectively[11].

When comparing this paper to the one by Anthony Ma and Gus Liu [7] it should be observed that their classification approaches were very different. Though they both built models to predict MBTIs, Luyckx and Daelemans performed binary classification on each of the character sets, while Ma and Gus performed multiclass classification on the entire 16 possibly personality classes. Binary classification for each of the character sets performed significantly better for the first two sets, and equally on the last two.

IV. Preprocessing

When initially inspecting the data, we found it to be very inconsistent. Since the text was taken from an online forum, the sentence length and content were very inconsistent. Since each row in the original dataset contained text for 50 posts by an individual user, we split the text so that each row was an individual users post. This left us with $433,750$ rows to be sent through preprocessing.

In the initial preprocessing of the data we removed all the URLs, punctuation, non-alphanumeric characters, and the stopwords (e.g. it, a, the, etc...) from the posts. The stopwords we used were from the Python NLTK libraries English corpus [12]. Once these had been removed we further removed any posts that were less than 7 words long. We found that this helped remove posts that provided no value to our models. We were left with 329136 posts ot train our models. Once the data had been preprocessed it was clear that the number of posts per personality type did not at all line up with the distribution of personality types in the general population. In our dataset the introverted types were much more well represented, while in the general population the extroverted personality type is much more common. We expect this is a result of the text posts being taken from an online Internet forum. Figure 2 shows how our data is distributed across the 16 classes in comparison to how the population is distributed.

Since we wanted to also perform binary classification for each personality type code (first, second, third, or fourth letter in the MBTI code), we wanted to see how our data was distributed across each binary class. Table II shows how our data is distributed across the four binary classes.

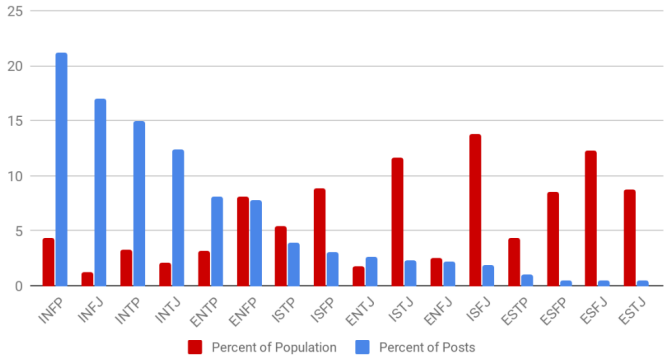

Fig. 2. Distribution of Classes in Our Dataset Compared to the General Population

Personality Code	Percent of Posts
I	76.75
E	23.25
N	86.31
S	13.69
F	54.19
T	45.81
P	60.42
J	39.58

TABLE II
DISTRIBUTION OF POSTS FROM OUR DATASET FOR MBTI CODES

When training our models for binary classification, we balance the data across both classes as we have enough posts to do this. After being balanced the minimum amount of posts for a character code (the second axis), will be 45058. This leaves 22529 posts per class.

V. Word2Vec and Embeddings

Before we trained the LSTM we performed pre-training on the data using a continuous bag of words (CBOW) Word2Vec [13] to create word embeddings. Word2Vec is a simple neural network that takes in the preprocessed data. At first, it creates a vocabulary of all the words that appear at least 10 times in the data set. Then it goes through the posts one at a time looking at each word in the post and the other words that are within 10 words of the current word to create a 300 dimensional vector for each word in the vocabulary of the other words that are similar. The output of Word2Vec ended up being a $V \times 300$ size array where V is the size of the vocabulary. The vocabulary for our data has $13,983$ words using the current word2vec configuration. Once we obtained the embeddings for each words that appeared at least 10 times, we reconstructed all the posts using the embeddings in place of the words. Hence a post was represented as a series of 300 dimensional vector instead of a series of words.

Since we used the word vectors for the future classification, it was important verify word2vec worked correctly. To visualise our vocabulary, we performed t-distributed stochastic neighbour embedding (t-SNE) on our generated vocabulary to reduce our 300 dimensional word vectors to 2 and 3 dimensions. A large reduction in dimension was due

to visualize data in 2D space to anticipate any connection between the words which where close in proximity. Figure 3 shows a 2D representation of 100 random words in our vocabulary.

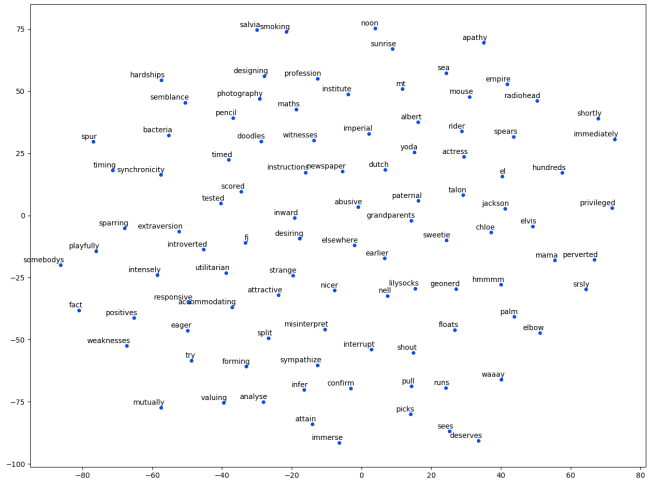

Fig. 3. 2D Representation of 100 random words

Words such as ”noon” and sunrise”, ”photography” and ”designing”, and ”extraversion” and ”introverted” are close together. This confirmed that the word2vec process worked correctly. We have also created a script using existing *Gensim* library which showed us the most similar words to another. The following listing are some examples that this script generated.

```
Words similar to hello
          hi 86.30%
         hey 69.31%
        here 50.13%
   greetings 48.36%
      fellow 45.76%
       hullo 43.82%
        perc 43.45%
        hiya 43.20%
     welcome 42.40%
       forum 39.63%

Words similar to internet
      online 44.57%
        perc 43.32%
        site 42.86%
    facebook 42.38%
         net 41.37%
   interwebs 40.41%
       forum 40.01%
         web 39.85%
      forums 38.06%
    computer 37.50%

Words similar to food
      snacks 44.08%
      eating 43.33%
       pizza 42.67%
       foods 41.59%
         eat 41.32%
     cooking 41.15%
       ramen 41.12%
     veggies 40.98%
        meat 40.60%
 restaurants 39.96%
```

Again, this confirms our word2vec worked as expected.

VI. RESULTS

A. Naive Bayes

There were three Naive Bayes that have been trained and tested: Multinomial Naive Bayes, Bernoulli Naive Bayes and Gaussian Naive Bayes. The data collected was reprocessed to normalise the data, and split into train and test data with 80:20 ratio.

The models were built for classification to predict one of the 16 personality classes i.e. INTJ, ESFP, etc. , as shown in table 2.

Initially, we used word embeddings we learned from the Word2Vec encoder to train the Naive Bayes models. After splitting the data into training and test sets, the personality type and its associated word embeddings were separated out to have separate label and feature representations. For each label in the data, the corresponding encoding of the sentences were taken and for each sentence, mean value of the encoding were calculated and put into a list. The model was trained with Naive Bayes libraries provided by Scikit-learn. The trained model was later tested on the test dataset and a classification report was generated.

The following report shows the average of the accuracy report provided by the classifiers:

Precision	Recall	f1-score	accuracy	Support
0.25	0.31	0.16	0.26	401

TABLE III
AVERAGE CLASSIFICATION PERFORMANCE REPORT FOR THE NAIVE BAYES MODELS

While training the data, it was found that each post did not have the same sentence length. When we were trying to get the mean for each sentence in the post, it was giving an inconsistent number of elements associated with each label. We have decided to overcome this problem by taking the mean of each sentences’ encoding first, and then taking the mean obtained from the sentences to obtain single mean value of encoding to represent the label.

Here, it should be noted that a single mean value was formulated for each personality type such that all the feature have the same length. This might have been a contributing factor for the poor performance of the model, since a single float value representing a class is not a good representation, due to overgeneralising of the features.

Due to the poor performance, we have decided to use different a method to extract our feature data to train and test the Naive Bayes models.

Gaussian Naive Bayes: We have used CountVectorizer as for feature extraction to train and classify with Guassian Naive Bayes, both taken from the Sklearn library.

The prepossessed file from the raw data was taken as input to form the class and the feature vectors. Initially, due to memory issues, we have decided to train on the first 20,000 instance of the data. The label and the feature set was split into test and train sets with 80:20 ratio. The data obtained was run through CountVectorizer with a regex tokenizer that splits on empty spaces in order to convert the collection of sentences as feature into a matrix of token counts. The Gaussian Naive Bayes was trained with the the extracted training feature matrix and the class vector. A final accuracy result was obtained using the test data; the following results were obtained:

Precision	Recall	f1-score	accuracy	Support
0.19	0.18	0.18	0.17.7	4000

TABLE IV
CLASSIFICATION PERFORMANCE REPORT FOR THE GAUSSIAN NAIVE BAYES MODEL -1

Here, the accuracy obtained was fairly low. When we looked further into the data, it was found that the distribution of data between the classes had a high disparity. The most occurring class had about $86,000$ instances of the data whereas the least occurring class had about $1,870$ instances of data. Moreover, it was discovered that a lot of the sentences had length of 7 or less, which does not provide much context for deciding one's personality. When we took the distribution according to the sentence length, it was found that the most optimal length was to consider sentences of length greater than 7 and less than 22. The data was pre-processed to remove sentences having length less than 8 or greater than 22.

The model was updated with these factors considered. To get even distribution of data, $1,850$ instances from each classes was taken from the pre-processed data for training and testing the model. The following classification performance report was obtained

Precision	Recall	f1-score	accuracy	Support
0.13	0.11	0.11	0.11	5741

TABLE V
CLASSIFICATION PERFORMANCE REPORT FOR THE GAUSSIAN NAIVE BAYES MODEL-2

It can be noticed that the overall performance decreased significantly. This was partly because using CountVectorizer to extract feature gave us a non-continuous representation of the data. However, Gaussian Naive Bayes requires a continuous data input to perform well, for which it did not work well.

Multinomial Naive Bayes: Similar data extraction process as mentioned was carried out for training and testing the model with Multinomial Naive Bayes classifier. Initially $20,000$ rows were collected due to memory limit. The following results were obtained:

Precision	Recall	f1-score	accuracy	Support
0.27	0.27	0.21	0.27	4000

TABLE VI
CLASSIFICATION PERFORMANCE REPORT FOR MULTINOMIAL NAIVE BAYES -1

A second attempt to pre-prune the data more to get an even distribution of input for all the classes was taken, as described above. Each classes had $1,850$ instances, and a range of sentence length was pre-pruned. The following performance report was obtained:

Precision	Recall	f1-score	accuracy	Support
0.20	0.19	0.19	0.187	5741

TABLE VII
CLASSIFICATION PERFORMANCE REPORT FOR MULTINOMIAL NAIVE BAYES-2

It can be noticed that the overall performance had dropped. We believe this might be due to the restriction of the data size which led to too few training samples, for which the model could not learn enough to have a better prediction.

Bernoulli Naive Bayes: Similar data extraction process was carried out for training and testing the model with Bernoulli Naive Bayes classifier. $20,000$ rows were collected due to memory limit. The following results were obtained:

Precision	Recall	f1-score	accuracy	Support
0.35	0.25	0.16	0.25	4000

TABLE VIII
CLASSIFICATION PERFORMANCE REPORT FOR BERNOULLI NAIVE BAYES-1

After further processing and the data with equal distribution of the classes and limited sentence length, the following performance report was obtained:

Precision	Recall	f1-score	accuracy	Support
0.26	0.18	0.17	0.178	4000

TABLE IX
CLASSIFICATION PERFORMANCE REPORT FOR BERNOULLI NAIVE BAYES-2

Similar to Multinomial Naive Bayes, we believe that too few data for training resulted in poor prediction, i.e performance.

B. SVM

The models were built for binary classification to predict one of the 16 personality classes i.e. INTJ, ESFP, etc. , as shown in table 2.

Initially, first $20,000$ instances of the input sentences were taken as an input. The data collected was reprocessed to normalise the data, and split into train and test data with 80:20 ratio. The following performance report was obtained:

Precision	Recall	f1-score	accuracy	Support
0.05	0.23	0.09	0.233	4000

TABLE X
CLASSIFICATION PERFORMANCE REPORT FOR SVM-1

Further pre-processing was done to limit the data size to be $1,850$ for each classes and a limited range of sentence length. The data obtained was split into train and test data with a ratio of 80:20.

Precision	Recall	f1-score	accuracy	Support
0.00	0.06	0.01	0.0611	5741

TABLE XI
CLASSIFICATION PERFORMANCE REPORT FOR SVM-2

Further preprocessing gave worse performance. We believe that choosing between four different models do not follow a linear behaviour, combined with a small amount of data, produced a lower overall performance. When looking at the classification performance report, it was seen that the further pre-processing lead to a 0% precision, which indicates that the data for input was not taken correctly. When the data was split for train and test data, the training feature for the testing classes did not get put into the training data at all, for which it could not do true positive predictions properly.

C. Long short-term memory

Along with the other methods described above, we implemented a recurrent neural network (RNN) with long short-term memory (LSTM) cells. We decided to perform binary classification on the data on each of the four character axes.

LSTM Network: An LSTM network is simply an RNN that uses LSTM cells, which enable the network to learn long term dependencies better than a standard RNN. An LSTM cell consists of gates, which are responsible for determining how input data flows through network and what data is remembered. These gates are the input gate, output gate, and forget gate. Figure 4 shows a simple LSTM chain and the contents of a single cell.

The forget gate is a sigmoid layer that controls the extent to which a value remains in the cell. It answer the question, "should I remember this value". It is represented by the equation

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The input gate is another sigmoid layer that controls what values in the cell will be updated. This is combined with a

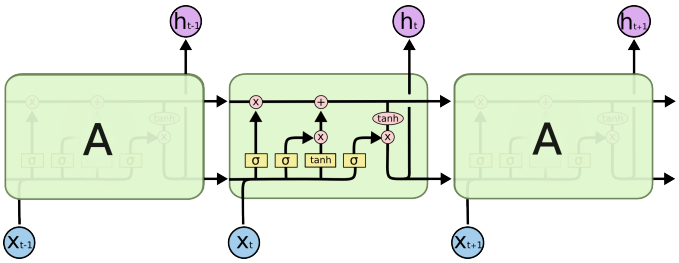

Fig. 4. LSTM Chain [14]

tanh layer which creates a vector of new candidate values, $\tilde{C}_t$.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Finally, the new cell state is calculated using the previous cell state and the values calculated above.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Balanced Data: Since the LSTM was only performing binary classification, we were able to balance the data across the classes. We ensured that there was an equal amount of data trained for each personality type code on each of the character axes. In addition to preventing the network from simply learning the most probable type code, it also allowed us to more easily compare the resulting accuracy. Random guessing would simply be 50%, as apposed to the different percentages shown in Table II.

Model: Our LSTM network accepts batches of preprocessed posts. Each post is a 300 dimensional word embedding specifically trained on our data. The output of the network is simply the probability of the post being in on of the two possible classes. Our model consists of

- 128 hidden layer cells
- 20% dropout
- Cross entropy loss
- Adap optimizer with 1e-3 learning rate

The dropout played an essential role in preventing the network from overfitting to the training data. The network was implemented using the PyTorch `nn.LSTM` module. After training for 100 epochs with a batch size of 64, the following accuracies on each of the character axes were achieved.

Personality Codes	Accuracy
I vs E	83.46%
S vs N	84.06%
T vs F	83.21%
P vs J	84.19%

TABLE XII
LSTM ACCURACY ON EACH CHARACTER AXES

As seen in Table XII, the accuracy for each model is in the $83 - 84\%$ range, with the best accuracy being for the

second character code. Figure 5 shows the loss and accuracy curves.

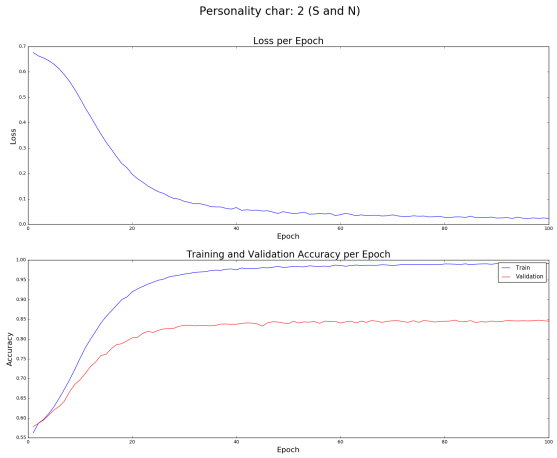

Fig. 5. Loss and Accuracy per Epoch using LSTM to classify second MBTI character code

D. Elman Network

The other recurrent neural network that we have attempted to use was the Elman Network which was made using a much simpler structure than an LSTM network. This network instead used a 1-hot input vector of size 44 that is a single character per time slice. The output was a size 16 vector containing probabilities of the likelihood of each of the personality classes. Internally, the network used the PyTorch `nn.rnn` module which implemented an Elman network, using a hidden layer size of 128. An additional linear layer just before the output softmax layer was inserted to allow the network to internally learn how to convert from a layer of size 128 to a layer with size 16.

The choice of running the network on a character by character basis was chosen due to the possibility common typing errors such as mixing the order of 'i' and 'e' having the potential of providing additional insight into what personality the input was created from.

Training was completed using stochastic gradient descent instead of batch or mini-batch learning as the network was slow due to requiring each letter to have a pass-through step instead of each word as in the Word2Vec preprocessing case. Similarly, to limit training and test time to a reasonable duration, the entire dataset was not used to train the Elman network. In the final iteration of the network, approximately 80000 samples were used to train, and about 40000 samples to test to get the single network iteration time down to a few hours. The accuracy for predicting all 16 classes is 22%. This is equivalent to always predicting the most probable class, which is what the network is doing.

VII. FUTURE WORK

The team has identified a few areas in which future projects could improve on the results of the project described by this paper. They mainly consist of refining the reported models to try and achieve as much performance out of each models as possible. For Word2Vec there were a couple of different parameters that could have still been changed such as the dimension size of the embeddings. Using a Skip-Gram model to encode the words rather than CBOW, would also be a potential avenue of improvement, as Skip-Gram can perform better with less data and performs better on less frequent words. It would also be interesting to test the models using negative sampling in both Word2Vec models rather than hierarchical softmax. Additionally, we believe that future projects could make some improvements to the preprocessing phase, as we noticed during the project that some sentence have only a few words, and some words were only 1 or 2 characters long. We believe the performance of our models could be improved if pruning was done on the faulty data.

The team proposes that an interesting future addition to the project would be generate synthetic data for the classes that were lacking in data. One interesting future project could be to use generative adversarial networks in order to increase the quantity of the data which maintaining a reasonably high quality of the data.

Future work on the LSTM and Elman network includes trying out different layers, different number of layers, testing different activation functions, and trying to tune some of the hyper parameters. Another point to test on for the LSTM is to get it to predict one of the 16 personality classes (i.e. INTJ, ESFP, etc) rather than predicting each individual letter one at a time and then combining them to make a class prediction e.g. predict I, then predict S, then T, then P to get ISTP. Similar work could be completed for the Elman network to try and predict on each binary classification instead of on one of the 16 personality classes. Additionally the Elman network could be changed so that it works with the Word2Vec encodings rather than just one-hot encodings of the words, which may give it a further boost in its scores.

On the Elman network, the network execution time was vastly slower than the execution time of the LSTM network causing it to be unfeasible to run a single epoch on the data. Future projects could focus on optimizing the speed of this network to be able to run more efficiently such that more than a single epoch can be completed on the input. Additionally, the data could be pre-processed more to balance out the frequency of each of the input classes so that the Elman network does not always predict a INFP label for every input.

For the Naive Bayes models there were issues with fitting all the data into memory and training time which made it impossible to use all the data for training. When the data was pre-pruned and an equal distribution of all the classes were taken, there was not enough data to train the model on. Moreover, when the full classification performance has been analyzed, it showed that a lot of the personality types were not getting predicted in GaussianNB. This might have been because a single value was taken as a feature representation of a class and that is over generalizing the data. For Gaussian distribution, feature extraction could be improved to get and

a continuous distribution of data. The model could be also changed to classify one character of the personality at a time, which may give better results since the data is more evenly distributed for each letter individually. Feature collection using the TfidfVectorizer can be also used for collecting training and testing data for Naive Bayes and Support Vector Machine.

Moreover, all the Naive Bayes models as well as the Support Vector Machine could be re-trained and tested to perform binary classification for each personality type code (first, second, third, or fourth letter in the MBTI code).

VIII. ETHICS

The dataset was created from text that was gathered from an English website and from people who have English as a first language. The use of our models to classify text from people who have English as a second language or non-English text could give very inaccurate results. Also, using the models on more formal text, such as that in a resume or a formal document, could results in misclassifications.

The Myers Briggs personality test has a history in worker placement. It is possible that our models could be taken out of context and be used to determine if a person should get a job or not based soley on their writing in social media or their cover letter/resume. This could lead to a job being given specifically to people who are from the demographic of our dataset, and overlook individuals from different demographic, creating a bias. This could lead to a job that is historically dominated by people from our trained demographic being more likely to get the job.

IX. DISTRIBUTION AMONGST TEAM MEMBERS

In order to ensure that the project moved in a timely manner, the various tasks that were required to be completed were split across the four team-members. Each member worked on the tasks that are most in-line with their previous experience to ensure that the project was able to continue with maximum efficiency.

A. Brandon Mabey

Mr. Mabey was responsible for creating and maintaining the source code repository. Mabey ensured that the team's progress was always in a state that was ready to test additional data to ensure that any member could quickly iterate and test new models without having to solve old issues.

Additionally, Mr. Mabey was responsible, together with Mr. Vickery, for researching and implementing the Elman RNN model.

Finally, he ensured that the code produced by the team will be properly GPU accelerated to ensure that training times remained reasonable for the team.

B. Brandon Vickery

Mr. Vickery was the resident math expert of the team; as such, Vickery ensured that the models under use were both valid and optimal for the classification of the 16 personality types.

He worked closely with Jake to ensure that cleaned data was in a format that is learn-able by each model, and ensured that the feature set of the data is properly constrained.

Part of Mr. Vickery's role was also to design and choose the layers under use when attempting to classify with neural networks. To this end, he worked with Mr. Mabey on the RNN model.

C. Jake Runzer

Jake was responsible for finding, formatting, cleaning, and splitting the data that will be used to train the neural network. Additionally, he was responsible for finding useful transformations on the data into a format that RNNs better train with.

Finally, together with Tania, he researched and created the LSTM model to perform binary classification on the data along the four character axes.

D. Tania Akter

The team coordinator was Tania. Responsible for team organisation and coordination, she ensured that the team worked together in a cohesive manner. Responsible for delegating tasks and jobs, she ensured that the team had good communication throughout the entire project. She implemented, trained and tested the data on Multinomial Naive Bayes, Gaussian Naive Bayes, Bernoulli Naive Bayes and Support Vector Machine as classifiers.

Additionally, she was responsible for helping implement the various models in the python programming language. Most notably, she worked with Jake to research, implement, and test the LSTM network model.

X. CONCLUSION

In the end the LSTM neural net showed the most promising results of any of the models. This is most likely to do with the fact that the LSTM was the only network that predicted on the four axes one at a time and the data could be more properly balanced along these four axes and still keep considerable amounts of data for each of the two classes on each axis. The main source of error for the other models stemmed from them predicting one of the 16 personality classes and the data for these 16 classes not being very well balanced at all. Any attempt to balance the data by under sampling some of the classes to make the distribution equal resulted in too little data in any of the classes to be able to properly fit a model. Other reasons why these models didn't perform as well as expected was due to not being able to fit all the data into memory. Also in the case of Gaussian Naive Bayes the use of CountVectorizor was not ideal as it does not give continuous data which is what Gaussian Naive Bayes is expecting.

REFERENCES

[1] "Myers-briggs type indicator." https://en.wikipedia.org/wiki/Myers_Briggs_Type_Indicator. (Accessed on 02/05/2018).

[2] "Myers briggs personality type chart." https://www.business2community.com/leadership/intro-myers-briggs-personality-types-01666155. (Accessed on 02/05/2018).

[3] "Kaggle." https://www.kaggle.com/. (Accessed on 02/05/2018).

[4] "Kaggle (mbti) myers-briggs personality type dataset." https://www.kaggle.com/datasnaek/mbti-type. (Accessed on 02/05/2018).

[5] "Personality cafe forum." personalitycafe.com/forum/. (Accessed on 02/05/2018).

[6] "Myers briggs statistics." https://www.statisticbrain.com/myers-briggs-statistics/. (Accessed on 02/05/2018).

[7] A. Ma and G. Liu, "Neural networks in predicting myers brigg personality type from writing style,"

[8] H. W. Dan Brinks, "Detection of myers- briggs type indicator via text based computer-mediated communication." http://cs229.stanford.edu/proj2012/BrinksWhite-DetectionOfMyersBriggsType.pdf. (Accessed on 02/05/2018).

[9] M. S. Halawa, M. E. Shehab, and E. M. R. Hamed, "Predicting student personality based on a data-driven model from student behavior on lms and social networks," in *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*, pp. 294–299, Oct 2015.

[10] J. Golbeck and C. R. andKaren Turner, "Predicting personality with social media." http://www.cs.umd.edu/hcil/trs/2010-30/2010-30.pdf. (Accessed on 02/05/2018).

[11] K. Luyckx and W. Daelemans, *Using Syntactic Features to Predict Author Personality from Text*. 01 2008.

[12] "Nltk english stopwords." https://gist.github.com/sebleier/554280, 04 2009. (Accessed on 03/09/2018).

[13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013.

[14] colah, "Understanding lstm networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed on 02/06/2018).