Myers-Briggs personality type prediction with text classifier

Ali Etminan - LiuID:aliet176

732A92

Abstract

Myers Briggs Type Indicator (MBTI) is an assessment that divides personality types over four major binary axis defined as (Introversion, Extroversion), (Sensing, Intuition), (Feeling, Thinking) and (Perceiving, Judging). The combination of these indicators make up 16 unique types that are widely used to sort people with respect to their dominant psychological functions. The types have proven to be highly correlated with how one makes decisions or takes action in personal and professional contexts. This report aims to explore the underlying patterns and sentiment within a collection of user posts, labeled with their respective personality type, and build a classifier that can be applied to determine personality type of a person based on text he/she has produced. This report attempts to study a number of re-sampling techniques as a pre-step, evaluate classifiers with different parametrizations and implement an LSTM model to build a classifier with high-accuracy. Final model is compared to results of a similar paper by Stanford University and tested on Donald Trump's twitter data.

Introduction

Extensive applications of personality assessment tools have given rise to the popularity of psychological frameworks, namely Myers-Briggs Type Indicator (MBTI) and Big Five traits (Openness, Conscientiousness, Extroversion, Agreeableness, Neuroticism) also referred to as OCEAN or CANOE ("Big Five Personality Traits" 2021). MBTI is based on Carl Jung's Typology that divides the human cognition into 8 distinct functions ("(MBTI) Myers-Briggs Personality Type Dataset" n.d.) which was later extended by Katharine Cook Briggs and Isabel Briggs Myers to include 2 new functions. The main functions can be briefly described as follows.

- Introversion, Extroversion
- Sensing, Intuition
- Feeling, Thinking
- Perceiving, Judging

Every individual is believed to prefer one type over the other in each category, bringing a total of 16 unique personality types across functions that is represented by their abbreviations *(i.e. ESTP = Extroversion, Sensing, Thinking, Perceiving).*

While preference of Big Five by modern academics has made it the leading model of our era, MBTI has yet remained a desirable approach given its somewhat deterministic way to assign a personality type to an individual. As opposed to MBTI, Big Five indicates in percentage where an individual falls on each of its five categories (Drenth n.d.) and uses factor analysis to evaluate each trait. In this sense, MBTI has turned into a hot topic for text analysis and in particular, classification.

This report makes use of a dataset consisting of more than 8,600 unique records each containing the last 50 posts users wrote on PersonalityCafe Forum ("Personality Cafe" n.d.), labeled with their respective MBTI type. There has undoubtedly been numerous exploratory analysis and classification attempts on this data. However, the goal of this paper is to extend the common data pre-processing steps by implementation and comparison of different resampling techniques on a baseline *(Multinomial Naïve Bayes)* classifier and then use the best resampled dataset as a bag of words representation to perform model selection on common classifiers, namely Random Forests, Support Vector Classifier and Logistic Regression Classifier.

The research is further extended by facilitating word embedding representations in a RNN architecture consisting of an embedding, Long Short Term Memory (LSTM) and a dense layer to perform classification. The best model will be tested on unseen data from Donald Trump's tweets to predict his personality type.

Theory

Vectorization

Through a two phase transformation, the Term Frequency-Inverse Document Frequency (TF-IDF) converts a string into a list of tokens where each token represents a space-separated word in the string (tokenization). In the second step, the tf-idf representation of each token is calculated using this formula (vectorization):

$$tf - idf(t,d) = tf(t,d).(log\frac{1+N}{1+df(t)} + 1)$$

Term frequency indicates the number of times a term appears in a particular document whereas Document frequency says in how many documents did a particular term appear. Tf-idf attempts to balance the weight of terms with high frequency by multiplying it with the *inverse* of document frequency of those terms. This avoids assigning a high weight to generic words that appear in many documents and thus do not carry much information or point to a certain topic. The above formula is the implementation of tf-idf used by Scikit learn. The letter **N** is the number of documents in the data. The effect of adding "1" to the idf in the equation is that terms that occur in all documents in a training set, will not be entirely ignored. Scikit-learn does

smoothing by adding a 1 to the numerator and denominator in the formula to avoid division by zero. This is equivalent to saying every term occurs in 1 additional document ("TfIdf Transformer," n.d.).

Vectorizers come in different flavors. A classical alternative to td-idf vectorizer is **Count Vectorizer**, where terms are simply represented by the number of times they appear in a document. This method is also referred to as *Term Frequency*.

Resampling

A common challenge in binary or multinomial classification tasks is to deal with imbalanced classes in the dataset. Overseeing this property can yield misleading results even if a sophisticated classifier is at play. It is therefore a necessity to experiment a number of resampling techniques with the aim of achieving a less biased results towards the majority class(es). Regardless of the method used, the resampling strategy can either be defined as a simple strategy (i.e. resample only the majority) or customized by setting desired portions for each class label.

Random Under Sampling

In random under sampling, samples are randomly removed from the majority class to balance the amount of available samples with respect to the minority class. This can be done iteratively until all classes have the same number of samples as the minority class.

A limitation of this technique is that a good (informative) sample is not distinguished from a non-informative one and thus there is a risk of removing critical information from the data (Brownlee 2020a). Furthermore, in situations where the minority class is significantly smaller compared to the other class(es), this method basically removes a large portion of the samples which makes it an impractical approach.

Random Over Sampling

In contrast to under sampling, random over sampling duplicates samples from the minority class at random to contain as many samples as in the majority class. Also similarly, the oversampling can be iterated to balance all classes with respect to the majority class. The sample generation is carried out by sampling with replacement.

In cases where class distribution is highly skewed, oversampling can lead to overfitting the minority class since a single sample repeatedly appears in the training set (Brownlee 2020a).

SMOTE

SMOTE is an abbreviation for Synthetic Minority Oversampling TEchnique. The main difference between random oversampling and SMOTE is that the minority classes are oversampled by repetition of the original sample in the former, while new samples are generated using the distribution of the original samples in the latter. There are different implementations of SMOTE (*SVM SMOTE* and *KMeans SMOTE*) which define the criteria that the new samples should be generated from. The baseline SMOTE uses K-nearest neighbours to generate synthetic samples.

$$x_{new} = x_i + \lambda(x_{zi} - x_i)$$

x_{zi} is one of the nearest neighbors of the sample and λ is a regularizer between 0 and 1. This model picks a sample at random to generate a synthetic sample (Chawla et al 2002).

In *KMeans SMOTE* samples are grouped together using the KMeans algorithm prior to sample generation. The *SVM SMOTE* variant uses samples as support vectors and the penalty factor **C** to control the number of support vectors (Chawla et al 2002).

Study shows that synthetic samples generated by SMOTE result in the same expected value in the minority class while reducing its variance (Blagus and Lusa 2013).

Exploratory Analysis

Sentiment Exploration

Sentiment analysis can be carried out as either an exploratory or predictive task. Predicting the sentiment that is specifically tailored to a certain text requires learning the sentiment through the means of a deep neural network, which could in turn be used to predict sentiment on a similar dataset.

For the purpose of exploration, there are numerous pre-built sentiment analysis libraries that can be applied to get an idea of the sentiment within the text. **TextBlob** library has two implementations of sentiment analyzer. One implementation takes advantage of python's pattern library to find the sentiment and outputs the results as two arguments:

Polarity: A score between 1 and -1 that can be roughly mapped to (excellent, best, good, bad, awful, etc).

Subjectivity: A value between 0 and 1 that quantifies the amount of "meaning" information (opinion) in the text(Malik n.d.).

Extracting the sentiment by personality type can help perform sanity check on the final results.

Topic Modelling

Topic models use statistical modelling to find the topics in a given document. The model is generative, meaning that documents are assigned topics with certain probabilities. *Gensim*'s **Latent Dirichlet Allocation (LDA)** library is widely used for topic modelling.

LDA builds a unigram model where every unigram (token) in document is assigned a probability for class k. Therefore a class (topic) for a given document is the topic that maximizes the product of unigram probabilities:

$$\hat{c} = argmax_{c \in C} P(c) \prod_{w \in V} P(w|c)^{\#(w)}$$

The learning can be improved by increasing the number of passes through the document. Each pass returns a marginal likelihood $P(w|z)$ which can be plotted to find out when and if the model has converged.

The model outputs n desired topics each containing the unigrams with the highest probability for that topic. In cases where a gold standard does not exist, a qualitative assessment reveals if the generated topics are informative or junk.

Classification

Multinomial Naive Bayes Classifier

Naive Bayes classifier is a baseline, yet an effective method to perform classification tasks. It naively assumes independence in the probability of terms appearing in a document and takes advantage of the Bayes rule to determine the class of a given document.

$$P(Class|Document) \propto P(Document|Class)P(Class)$$

In words, the probability of a document belonging to a class is proportional to the likelihood of observing words in a document given its class multiplied by the prior probability of observing the class. The prior class probabilities are determined by the distribution of the class labels (class frequencies) in the dataset. The predicted class is defined using the following formula.

$$\hat{c} = argmax P(C). \prod P(w|C)^{\#(w)}$$

The class label that maximizes the posterior probability calculated on the right hand side of the formula will be set as the predicted class label.

Random Forest Classifier

Random Forest is an ensemble method that was introduced to address the overfitting problem of Decision Trees. Steps performed by the Random Forest algorithm can be described as below.

- **Step1:** Bootstrap random samples with replacement from the data.
- **Step2:** Randomly select a subset of features (words) to fit a decision tree on the bootstrapped samples to make a prediction.
- **Step3:** Repeat step 2 a desired number of times and take the majority vote (label) as the final prediction.

Random Forest Classifiers can be tweaked to meet the specifications of the problem. Class weight balancing and tree pruning can control the model complexity and flexibility.

Support Vector Classifier

Support Vector Machines handle the task of classification by drawing decision boundaries in the d dimensional feature space. Optimally, the aim is to draw a decision boundary that has the maximum distance with the closest data points. This distance on either side of the decision boundary is called **margin** and the data points on the margins are referred to as **support vectors**.

SVMs use kernels to map data into d-dimensional feature space. The two frequently used kernels by SVMs are the **Linear** and **Radial Basis Function (RBF)** kernels. RBF is also known as the **Gaussian Kernel.**

Linear:

$$K(X, X') = XX'$$

The kernel only calculates the dot product of the two samples x and x'. Linear kernels are preferred only when the samples are linearly separable.

RBF:

$$K(X, X') = exp(-\frac{||X - X'||^2}{2\sigma^2})$$

or alternatively

$$K(X, X') = exp(-\gamma||X - X'||^2)$$

where $\gamma = \frac{1}{2\sigma^2}$

The numerator in the former definition is the euclidean distance between sample x and x' and σ is a smoothing coefficient to control the effect of the distance between the two samples. RBF is useful when samples are not linearly separable and a flexible decision boundary is required.

Algorithms like *Logistic Regression* and *Support Vector Machines* were primarily designed to solve a binary problem. There are two extensions to this methods that enables them to be applied to multiclass problems.

One-vs-One(ovo): The multiclass problem is turned into a binary one by comparing each class against every other class. This requires splitting the data into subsets that include all possible combinations of class pairs and create a model for each. A downside of this approach is the creation of a large number of datasets where there exists several classes (Brownlee 2020b).

Example: A dataset with labels A,B,C need a model to classify A vs B, A vs C and B vs C

One-vs-Rest(ovr): In contrast to *ovo*, *ovr* splits the data into two parts where one contains samples from only one class and the other contains samples from all the rest of the classes. A drawback of this technique is that the data is regenerated for every subset which can be computationally inefficient with large datasets (Brownlee 2020b).

Example: A dataset with labels A,B,C need a model to classify A vs [B,C], B vs [A,C] and C vs [A,B]

Logistic Regression Classifier

Logistic Regression Classifier uses the same principles as the Naive Bayes Classifier, in a sense that it calculates a vector of conditional probabilities for each class using a combination of a simple linear model and the sigmoid function.

$$z = xW + b\hat{y} = \frac{1}{1 + exp(-z)}$$

W is a $F \times K$ matrix containing class weights for each input vector. The resulting $\hat{y}$ is a vector of conditional class probabilties. However, logistic classifier uses *gradient descent* to directly optimize the conditional probability rather than optimizing the joint likelihood in naive bayes.

The model parameter θ, which is the W matrix, is initialized arbitrarily. The model then attempts to optimize θ by minizing the loss function (cross entropy):

$$L(\theta) = -(ylog\hat{y} + (1 - y)log(1 - \hat{y}))$$

using *gradient descent*:

$$\theta := \theta - \eta \triangledown L(\theta)$$

Softmax is another generalizations of the logistic classifier that is regularly used for classification tasks using neural networks.

Deep Learning

Word Embeddings

A more practical alternative representation to *bag of words (BOW)* is *word embeddings*. Bag of words, like TF or TF-IDF, can grow exponentially large for large texts and can therefore be computationally slow. Moreover, BOW representations are not context-sensitive which implies that words appear out of their context.

In word embeddings, words in a document are mapped to d-dimensional vectors for a fixed d value that is significantly smaller than the length of the document. The idea is that words that frequently appear in a similar context, tend to have similar meanings and distribution and thus should be grouped together in a word embedding vector. In this regard, word embeddings are dense vectors that contain words in a particular context or in other words are context-sensitive.

One can use *Singular Value Decomposition (SVD)* or *Positive Pointwise Mutual Information (PPMI)* to obtain word embeddings. However, neural networks have proven to be the most practical approach when it comes to obtaining this representation.

Initially, a document, similar to the pre-processing steps discussed so far, is tokenized and converted to a sparse matrix of sequences. For the sake of a NN, all sequences need to be padded or simply aligned to have the same length as the longest sequence. The matrix is passed to the front layer of the neural network that is the embedding layer.

The embedding layer is initialized at random and update its values from the data using backpropagation. The output dimension of the layer is commonly chosen to be in a range from 50 to 300. In line with the deep architecture used by this paper, the output of the embedding layers are forwarded to an LSTM layer for further processing.

LSTM

Model of choice when it comes to text data are the Recurrent Neural Networks (RNN). This is due to two main reasons. One, RNNs can process long sequences of data without increasing model size and two, they can use historical information within the data through *hidden states*, which makes them perfect for NLP tasks (Amidi, n.d.). A special implementation of RNN is the *Long Short Term Memory (LSTM)* networks.

The idea behind LSTM networks is to control which parts of the sequence to memorize and which parts to forget. Therefore, an LSTM cell is able to *forget* the irrelevant and *remember* the relevent information regardless of how distant the memory is. LSTM cells are sequentially connected to each other where they take a sequence (word embedding sequences) and the hidden state from the previous cell as input, perform multiple transformations on the input, update the hidden state and pass it to the next cell (Olah 2015).

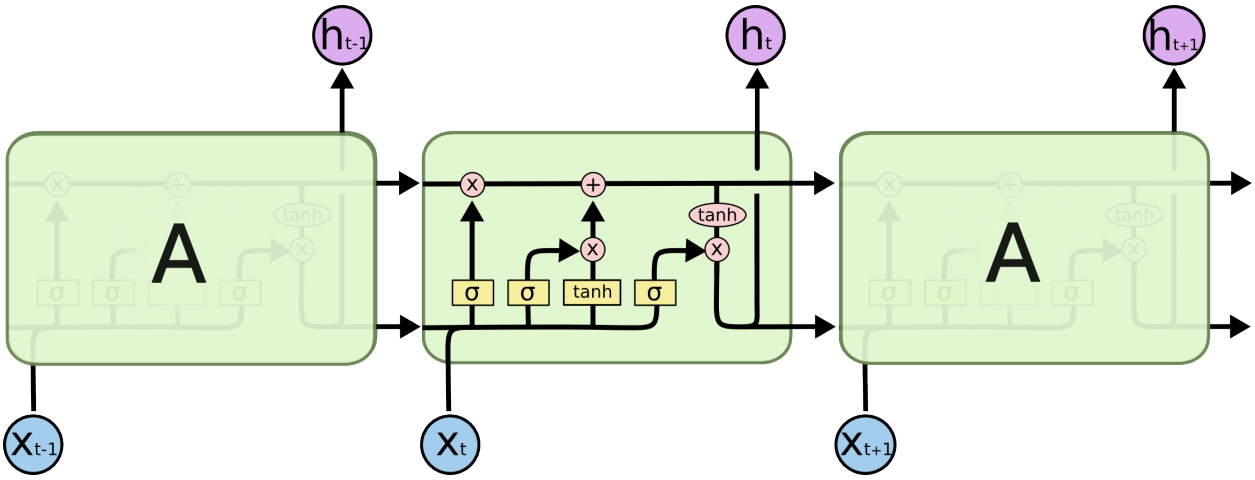


Figure 1: LSTM Cell (*Credit: Christopher Olah*)

From the left side, functionality of an LSTM cell can be summarized as:

- **Forget Gate:** Uses a sigmoid function to output a value between 0 and 1, 0 meaning completely forget and 1 meaning compteley pass the value through.
- **Input gate + tanh:** A second sigmoid layer (input gate) decides what values to update and the tanh layer creates a vector of candidates to add to the cell state.

- **Update state:** The result of the previous steps are combined to update the cell state.
- **Output:** The result of step 3 goes through a third sigmoid layer and consecutively another tanh layer to output only the desired values (Olah 2015).

Bidirectional LSTMs

In bidirectional LSTM, there are two parallel sequence of LSTM cells, where one reads the input as it appears and the other takes the same input but in a reversed order.

The idea was initialized in the domain of speech recognition since there is evidence that the whole context around a term is used to interpret a speech rather than a linear interpretation (Brownlee 2017a).

Dense Layer

The output of the LSTM networks is finally passed to a dense layer with a *Softmax* activation function that is a generalization of the *sigmoid* function for multiclass classification.

$$softmax(z)[i] = \frac{exp(z[i])}{\sum_k exp(z[k])}$$

Softmax function provides the final output as one of the class labels used in the training data.

Loss Function and Optimizer

A convenient loss function to optimize the model is *Cross Entropy (see Logistic Regression classifier)* that can be applied to either binary or multinomial problems.

As opposed to *Stochastic Grdient Descent* where only a single learning rate is used for all weights (parameters) in the network, *Adam optimizer* maintains a specific learning rate for every network parameters during the training. *Adam* has proven to be fast and accurate for deep learning and in particular for NLP domain (Brownlee 2017b).

Regularizers

Neural Networks are prone to overfitting, especially when the training data is not sufficiently large. One way to overcome this issue is taking advantage of regularizers. In particular, two commonly used techniques will be discussed.

Dropout: A dropout layer in a NN takes a probability p as its argument to decide what portion of the nodes in the network to ignore. If, for instance, a value of $p = 0.25$ is passed, it means that the layer will pass the incoming nodes with a probability of 25% and drops them with the probability of 75% percent. This method is particularly useful in deep architectures where numerous nodes are involved where they start to build co-linearity, resulting in an overfit model.

Max Pooling: Also helps prevent overfitting by outputing an abstract representation of its input through a *max* filter. In other words, the max pooling layer outputs the maximum value of its input sequence which is invariant to the internal representation and costs less computationally ("Max-Pooling," n.d.).

Model Evaluation

Precision vs Recall

A classical and yet effective tool to measure and evaluate the quality of a classifier is through the means of a confusion matrix. It is a tabular representation of the actual versus the predicted classes that allows two scores be calculated from its values.

Precision is the ratio of correctly predicted classes versus all predicted classes.

$$Precision = \frac{tp}{tp + fp}$$

Recall is the ratio of correctly predicted classes versus all correct classes (that were not correctly predicted).

$$Recall = \frac{tp}{tp + fn}$$

The two scores stand at a trade-off where if you increase precision, you lose some recall and vice verse. The preference of either scores depends on how the results are supposed to be interpreted. An alternative solution is to use the F1 score.

F1 score

F1 aims to find the right balance between *Precision* and *Recall* by combining the two as follows.

$$F1 = \frac{2.precision.recall}{precision + recall}$$

F1 is widely used as an overall as well as per class score to evaluate a classification model.

Data

The dataset used to perform personality classification consists of more than 8,600 unique records each containing the last 50 posts users wrote on PersonalityCafe Forum ("Personality Cafe" n.d.), labeled with their respective MBTI type. Here is a distribution of the personality types in the data.

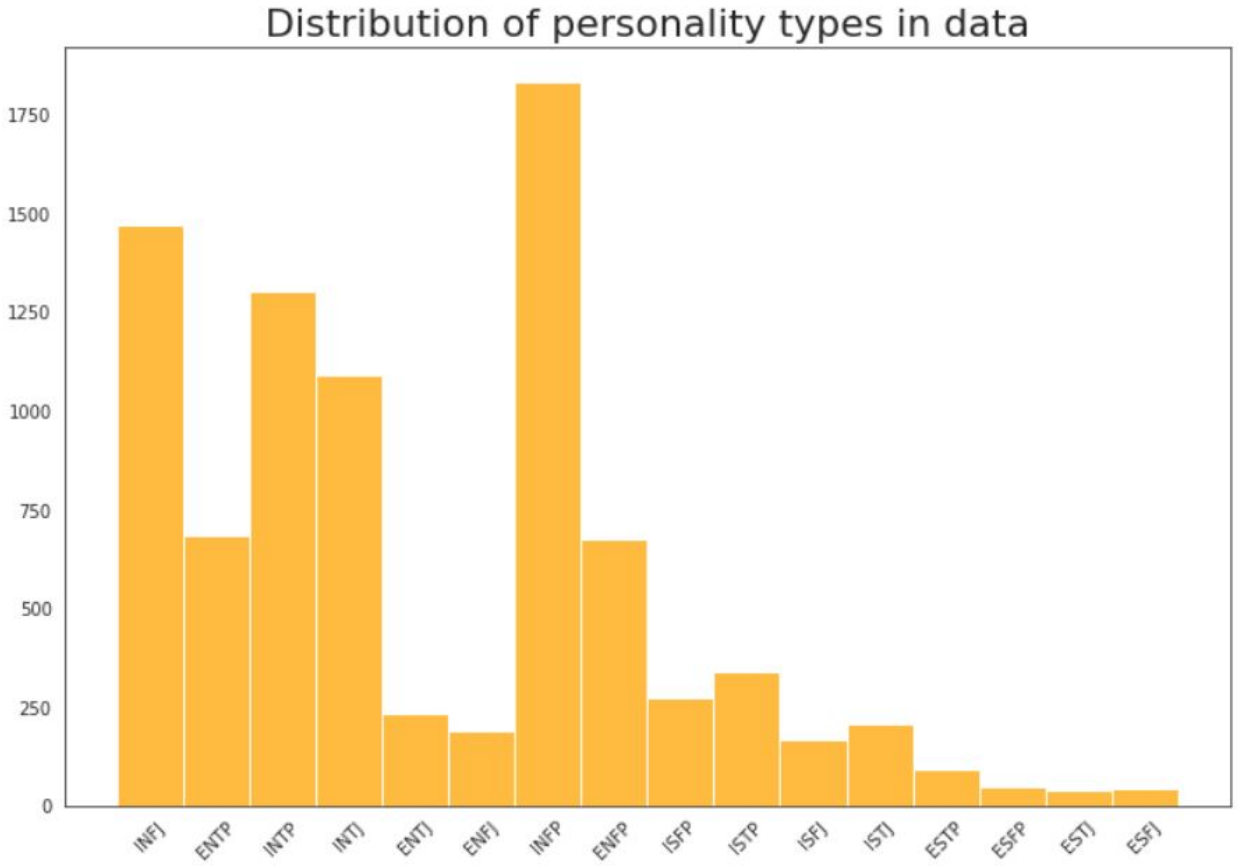
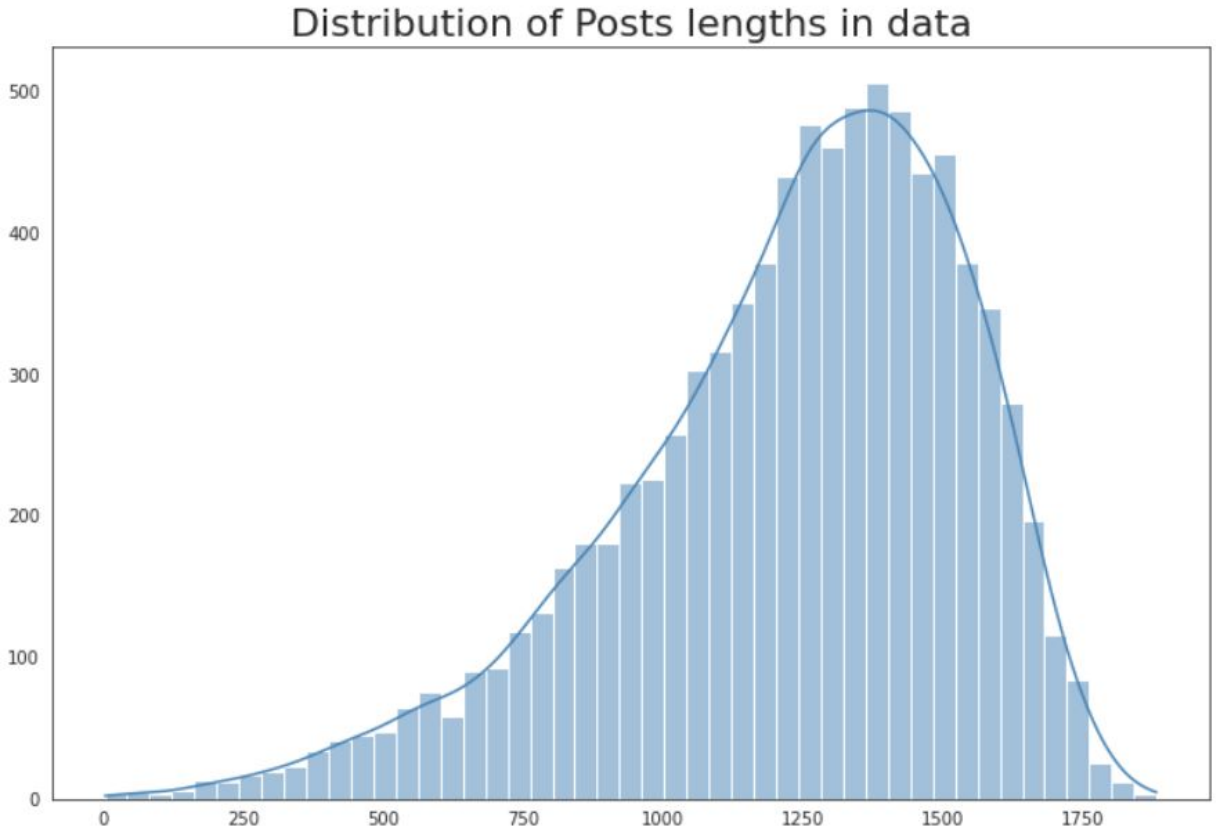
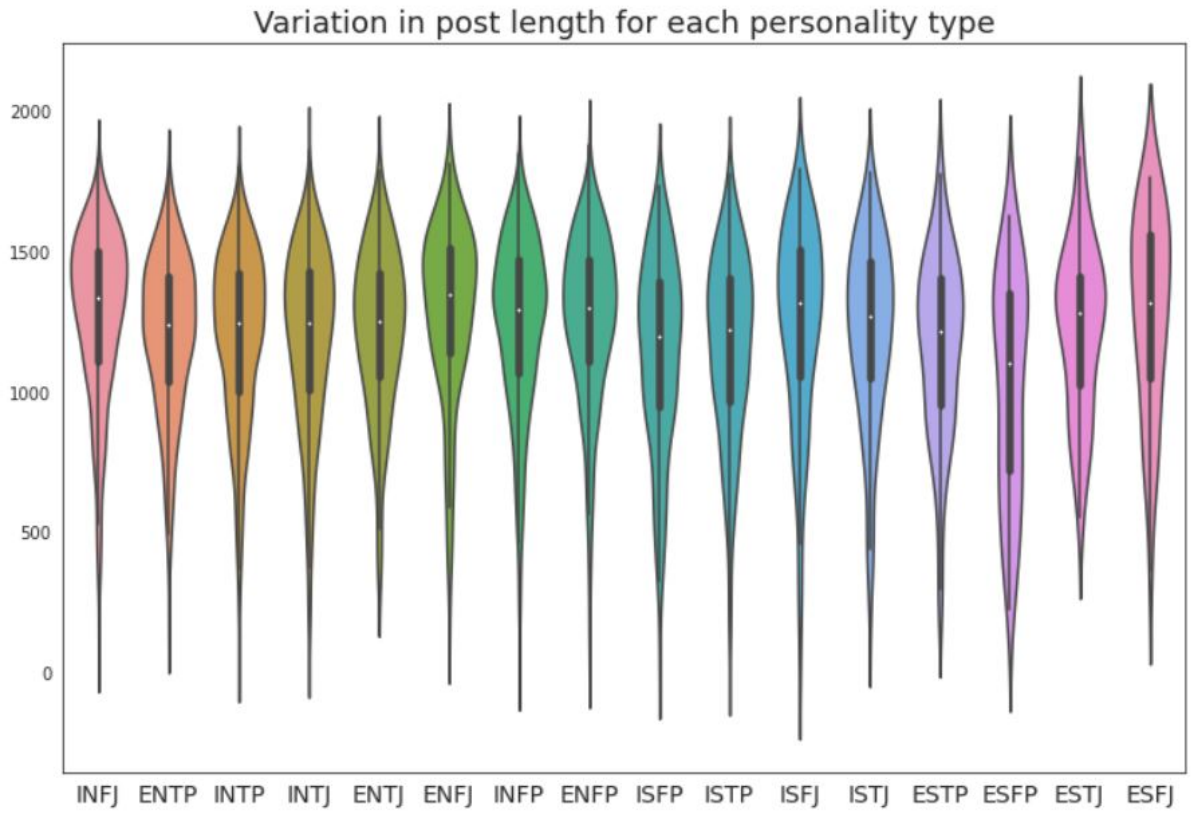


Figure 2: Distribution of personality types

It can be seen that the number of samples are not balanced for every personality type. *INFP* and *INFJ* have the largest samples followed by *INTP* and *INTJ* types. The remaining 12 types have significantly less samples. Therefore it is essential to build a more balanced resampled dataset prior to building a classifier. The following two figures visualize the overall and per-class distribution of post lengths. The average length of a post is approximately 1226 words.


Distribution of Posts lengths in data
500
400
300
200
100
0
0
250
500
750
1000
1250
1500
1750


Variation in post length for each personality type
2000
1500
1000
500
0
INFJ
ENTP
INTP
INTJ
ENTJ
ENFJ
INFP
ENFP
ISFP
ISTP
ISFJ
ISTJ
ESTP
ESFP
ESTJ
ESFJ

Method

Exploration

As a primary step, the average sentiment for each of the 16 personality types is compared using textblob's sentiment analyzer with the *pattern analyzer* method. The analyzer outputs a *polarity* and *subjectivity* score per document which are averaged with respect to each personality type.

A common approach by many literatures is regrouping the data over the four major axis (i.e. Introvert vs Extrovert, Sensing vs Intuition etc). This is beneficial to overcome the imbalanced distribution of classes as well as to be able to compare how each major function differs from its opposite. This paper uses this representations to perform topic modelling and create word clouds.

Using Gensim's *Latent Dirichlet Allocation (LDA)* model library, five topics were generated for each segment of the data using a tokenized representation as input. In order to maximize the likelihood of obtaining informative topics, 50 iterations were used for each segment. Consecutively, a word cloud was generated for every segment through the *Wordcloud* library. Wordcloud uses the same tokens as of the LDA to generate a word cloud. The topic-word cloud pairs highlight key features of each segment and visualize the overall differences of different groups.

Vectorization

Three vectorization methods were implemented and evaluated on a *Multinomial Naive Bayes* classifier to see which performs best.

- Custom tf-idf vectorizer: Lemmatized and tokenized documents using SpaCy's tokenizer. Personality type labels were passed as *stop words* to avoid their effect in class predictions. The tokenizer was passed to scikit learn's tf-idf vectorizer.
- Regular tf-idf vectorizer: Tokenized documents using the built-in *english* stop word removing method in scikit learn's tf-idf vectorizer.
- Count vectorizer: Tokenized documents using the built-in *english* stop word removing method in scikit learn's count vectorizer.

Resampling

Using the best performing document representation from the previous step, a number of resampling techniques is applied to the data to overcome the the lack of samples for certain types. For this purpose, the following resampling functions from *imblearn* was applied:

- Random undersampling
- Random oversampling
- SMOTE *(KMeans SMOTE)*

Building a classifier

The resampled dataset that yielded the best accuracy on naive bayes classifier is split into train, validation and test sets (60/20/20) for following steps. The report proceeds to train the following three classification algorithms on their default parameter settings on the train set and evaluate them on the validation set.

- Random Forest Classifier
- Support Vector Classifier
- Logistic Regression Classifier

After observing the overall and per class *recall*, *precision* and *f1-score*, we attempt to perform grid search on each of the classification algorithms to find the best performing setting. The search seeks to find the best combination of the following parameters with 5-fold cross validation.

- Random Forest

  - **n_estimators:** number of decision trees
  - **criterion:** function to evaluate leaf purity
  - **min_samples_split:** minimum number of samples required to split an internal node
  - **min_samples_leaf:** minimum number of samples to split a leaf node
  - **ccp_alpha:** Cost-complexity parameter to peform pruning
- Support Vector Classifier
  - **C:** Penalty factor
  - **Kernel:** Kernel function
- Logistic Regression Classifier
  - **Penalty:** Penalty method (e.g. l1, l2 etc)
  - **C:** Penalty factor
  - **Solver:** Optimization function

The grid search uses a pipeline to perform the search on the train set. The best setting is finally evaluated on the test set to choose the final model.

Word Embeddings and LSTM

In the next phase, we turn to a sequential RNN model with an LSTM network using Tensorflow and Keras libraries. The below steps are performed to pre-process the data for the RNN model.

- **Tokenization:** Tokenize the documents using Keras' built-in tokenizer
- **Turning to sequence:** Tokenized vectors are converted to sequence format compatible with NN
- **Padding:** Sequences are all padded to have the same length as the longest sequence.
- **Label conversion:** Personality types (labels) are converted using *One Hot Encoding (OHE)*

The padded sequences are passed to the front embedding layer in the RNN with the following three configurations.

- **Configuration 1**
  - *Over sampled training data*
  - Embedding layer (output dimension = 64)
  - Bidirectional LSTM with return_sequence=True
  - MaxPooling1D layer
  - Dropout layer (0.25)
  - Dense layer (16 units, *softmax activation*)
  - Optimizer: *Adam*
  - Loss: *Categorical Cross Entropy*
  - Batch size: 100
  - Epochs: 10
- **Configuration 2**
  - *Original training data*
  - Similar to configuration 1 with original samples
- **Configuration 3**
  - *Original training data*
  - Embedding layer (output dimension = 100)
  - Dropout layer (0.50)
  - Bidirectional LSTM with return_sequence=True
  - MaxPooling1D layer
  - Dropout layer (0.50)
  - Dense layer (16 units, *softmax activation*)
  - Optimizer: *Adam*
  - Loss: *Categorical Cross Entropy*
  - Batch size: 100
  - Epochs: 15

All configurations use the train and validations sets (60/20) during training and tested on the remaining 20% test set. Loss functions and train/validation accuracy plots are used to monitor model performance. For the final test, the test accuracy is observed.

Performance on external data

As an external validation effort for the best performing classifier, Donald Trump's twitter data with more than 43,000 tweets ("Trump Tweets" n.d.) is used to predict his personality type.

Results

Topic Modelling

In this section, the data is regrouped according to 8 main personality functions: *Introverts*, *Exroverts*, *Intuition*, *Sensing*, *Feeling*, *Thinking*, *Judging* and *Perceiving*. The groupings are used to perform topic modeling and create a word cloud. The results of opposing functions are compared.

- *Introverts* vs *Exroverts*
- *Intuition* vs *Sensing*
- *Feeling* vs *Thinking*
- *Judging* vs *Perceiving*

Introvert

```
[(0,
  '0.021*"like" + 0.016*"think" + 0.012*"feel" + 0.012*"know" + 0.012*"people" + 0.010*"time" + 0.009*"thing" + 0.008*"want" + 0.008*"love" + 0.007*"friend"'),
 (1,
  '0.015*"think" + 0.015*"like" + 0.011*"people" + 0.009*"know" + 0.008*"thing" + 0.007*"time" + 0.006*"want" + 0.006*"feel" + 0.005*"find" + 0.005*"say"'),
 (2,
  '0.016*"like" + 0.015*"think" + 0.010*"people" + 0.009*"know" + 0.008*"time" + 0.007*"thing" + 0.006*"feel" + 0.006*"type" + 0.005*"want" + 0.005*"find"'),
 (3,
  '0.017*"like" + 0.015*"think" + 0.011*"people" + 0.010*"know" + 0.009*"time" + 0.008*"thing" + 0.007*"want" + 0.007*"feel" + 0.005*"find" + 0.005*"work"'),
 (4,
  '0.022*"think" + 0.019*"like" + 0.012*"people" + 0.012*"know" + 0.011*"type" + 0.010*"feel" + 0.009*"thing" + 0.008*"time" + 0.007*"want" + 0.006*"find"')]
```

Extrovert

```
[(0,
  '0.019*"think" + 0.016*"like" + 0.012*"people" + 0.011*"know" + 0.009*"thing" + 0.008*"type" + 0.008*"time" + 0.007*"feel" + 0.006*"want" + 0.006*"find"'),
 (1,
  '0.017*"think" + 0.017*"like" + 0.011*"know" + 0.011*"people" + 0.008*"thing" + 0.008*"time" + 0.007*"type" + 0.007*"feel" + 0.006*"want" + 0.006*"friend"'),
 (2,
  '0.021*"like" + 0.018*"think" + 0.013*"know" + 0.012*"people" + 0.010*"feel" + 0.010*"thing" + 0.009*"time" + 0.008*"love" + 0.007*"friend" + 0.007*"want"'),
 (3,
  '0.019*"like" + 0.017*"think" + 0.012*"know" + 0.011*"people" + 0.008*"thing" + 0.008*"time" + 0.007*"type" + 0.007*"feel" + 0.007*"want" + 0.005*"friend"'),
 (4,
  '0.017*"like" + 0.017*"think" + 0.011*"people" + 0.011*"know" + 0.008*"thing" + 0.008*"feel" + 0.008*"time" + 0.007*"type" + 0.006*"want" + 0.005*"love"')]
```

Intuition

```
[(0,
  '0.021*"like" + 0.020*"think" + 0.015*"feel" + 0.014*"people" + 0.014*"know" + 0.011*"thing" + 0.010*"time" + 0.009*"want" + 0.008*"love" + 0.008*"friend"'),
 (1,
  '0.016*"think" + 0.014*"like" + 0.011*"people" + 0.009*"know" + 0.008*"thing" + 0.007*"time" + 0.006*"type" + 0.005*"feel" + 0.005*"want" + 0.005*"find"'),
 (2,
  '0.022*"think" + 0.019*"like" + 0.014*"type" + 0.012*"know" + 0.011*"people" + 0.008*"thing" + 0.007*"feel" + 0.007*"time" + 0.006*"want" + 0.006*"INFJ"'),
 (3,
  '0.018*"like" + 0.014*"think" + 0.010*"know" + 0.010*"people" + 0.008*"time" + 0.007*"thing" + 0.007*"feel" + 0.007*"want" + 0.007*"love" + 0.005*"good"'),
 (4,
  '0.018*"like" + 0.016*"think" + 0.011*"people" + 0.010*"know" + 0.009*"time" + 0.008*"thing" + 0.008*"feel" + 0.006*"want" + 0.006*"go" + 0.005*"love"')]
```

Sensing

```
[(0,
  '0.017*"like" + 0.015*"think" + 0.010*"know" + 0.010*"people" + 0.009*"thing" + 0.009*"time" + 0.007*"type" + 0.007*"feel" + 0.007*"want" + 0.005*"good"'),
 (1,
  '0.019*"like" + 0.019*"think" + 0.011*"people" + 0.011*"know" + 0.009*"type" + 0.008*"thing" + 0.008*"time" + 0.008*"feel" + 0.006*"want" + 0.006*"love"'),
 (2,
  '0.021*"like" + 0.018*"think" + 0.012*"know" + 0.012*"people" + 0.010*"thing" + 0.009*"time" + 0.009*"feel" + 0.008*"want" + 0.007*"friend" + 0.007*"type"'),
 (3,
  '0.018*"like" + 0.015*"think" + 0.010*"know" + 0.010*"people" + 0.008*"time" + 0.008*"thing" + 0.007*"type" + 0.007*"feel" + 0.007*"want" + 0.005*"go"'),
 (4,
  '0.017*"like" + 0.014*"think" + 0.010*"people" + 0.010*"know" + 0.008*"time" + 0.008*"thing" + 0.008*"type" + 0.007*"feel" + 0.007*"want" + 0.005*"go"')]
```

Feeling

```
[(0,
  '0.023*"like" + 0.018*"think" + 0.014*"know" + 0.013*"feel" + 0.012*"people" + 0.010*"thing" + 0.010*"time" + 0.009*"love" + 0.008*"want" + 0.008*"friend"'),
 (1,
  '0.022*"think" + 0.018*"like" + 0.012*"people" + 0.012*"know" + 0.011*"type" + 0.010*"feel" + 0.009*"thing" + 0.008*"time" + 0.006*"want" + 0.006*"love"'),
 (2,
  '0.018*"like" + 0.018*"think" + 0.013*"people" + 0.013*"feel" + 0.011*"know" + 0.010*"thing" + 0.009*"time" + 0.008*"want" + 0.007*"love" + 0.006*"find"'),
 (3,
  '0.018*"like" + 0.016*"think" + 0.010*"know" + 0.010*"people" + 0.009*"feel" + 0.009*"time" + 0.008*"thing" + 0.007*"love" + 0.006*"want" + 0.006*"friend"'),
 (4,
  '0.017*"like" + 0.016*"think" + 0.011*"people" + 0.011*"know" + 0.009*"feel" + 0.008*"thing" + 0.008*"time" + 0.006*"want" + 0.006*"love" + 0.005*"find"')]
```

Thinking

```
[(0,
  '0.016*"like" + 0.016*"think" + 0.011*"people" + 0.010*"know" + 0.008*"time" + 0.008*"thing" + 0.006*"want" + 0.006*"feel" + 0.005*"find" + 0.005*"go"'),
 (1,
  '0.017*"like" + 0.015*"think" + 0.010*"people" + 0.010*"know" + 0.008*"time" + 0.008*"thing" + 0.006*"want" + 0.006*"feel" + 0.006*"type" + 0.005*"find"'),
 (2,
  '0.016*"like" + 0.015*"think" + 0.011*"people" + 0.010*"know" + 0.008*"time" + 0.008*"thing" + 0.007*"want" + 0.007*"feel" + 0.005*"find" + 0.005*"go"'),
 (3,
  '0.020*"like" + 0.019*"think" + 0.013*"people" + 0.012*"know" + 0.009*"thing" + 0.009*"time" + 0.008*"feel" + 0.007*"want" + 0.006*"friend" + 0.006*"find"'),
 (4,
  '0.018*"think" + 0.016*"like" + 0.011*"people" + 0.010*"know" + 0.010*"type" + 0.008*"thing" + 0.007*"time" + 0.006*"feel" + 0.006*"want" + 0.005*"find"')]
```

Perceiving

```
[(0,
  '0.017*"like" + 0.016*"think" + 0.011*"people" + 0.010*"know" + 0.008*"time" + 0.008*"thing" + 0.006*"feel" + 0.006*"want" + 0.005*"go" + 0.005*"good"'),
 (1,
  '0.020*"think" + 0.017*"like" + 0.012*"people" + 0.011*"know" + 0.009*"thing" + 0.009*"type" + 0.008*"feel" + 0.008*"time" + 0.006*"want" + 0.006*"find"'),
 (2,
  '0.019*"like" + 0.017*"think" + 0.012*"feel" + 0.012*"people" + 0.011*"know" + 0.010*"time" + 0.009*"thing" + 0.008*"want" + 0.008*"love" + 0.006*"friend"'),
 (3,
  '0.023*"like" + 0.019*"think" + 0.013*"know" + 0.012*"people" + 0.010*"feel" + 0.009*"thing" + 0.009*"time" + 0.009*"friend" + 0.008*"love" + 0.007*"want"'),
 (4,
  '0.018*"think" + 0.018*"like" + 0.011*"know" + 0.010*"people" + 0.008*"type" + 0.008*"feel" + 0.008*"thing" + 0.008*"time" + 0.006*"want" + 0.006*"love"')]
```

Judging

```
[(0,
  '0.018*"like" + 0.018*"think" + 0.013*"people" + 0.012*"know" + 0.011*"feel" + 0.010*"thing" + 0.009*"time" + 0.007*"want" + 0.007*"friend" + 0.006*"find"'),
 (1,
  '0.018*"like" + 0.016*"think" + 0.012*"people" + 0.011*"know" + 0.009*"time" + 0.009*"feel" + 0.009*"thing" + 0.007*"want" + 0.006*"love" + 0.006*"friend"'),
 (2,
  '0.019*"think" + 0.017*"like" + 0.011*"people" + 0.011*"know" + 0.010*"type" + 0.008*"thing" + 0.008*"feel" + 0.007*"time" + 0.006*"want" + 0.005*"find"'),
 (3,
  '0.019*"like" + 0.017*"think" + 0.012*"know" + 0.012*"people" + 0.009*"feel" + 0.008*"time" + 0.008*"thing" + 0.007*"want" + 0.006*"type" + 0.006*"find"'),
 (4,
  '0.017*"like" + 0.017*"think" + 0.011*"know" + 0.011*"people" + 0.008*"thing" + 0.008*"feel" + 0.008*"time" + 0.006*"want" + 0.006*"type" + 0.006*"love"')]
```


Intuition
mean
say
come
good
look
work
well
type
need
right
actually
talk
life
person
understand
INTJ
INFP
tell
sure
believe
interest


Sensing
mean
say
look
come
good
well
work
type
need
talk
life
person
INFP
actually
INFJ
tell
right
sure
interest

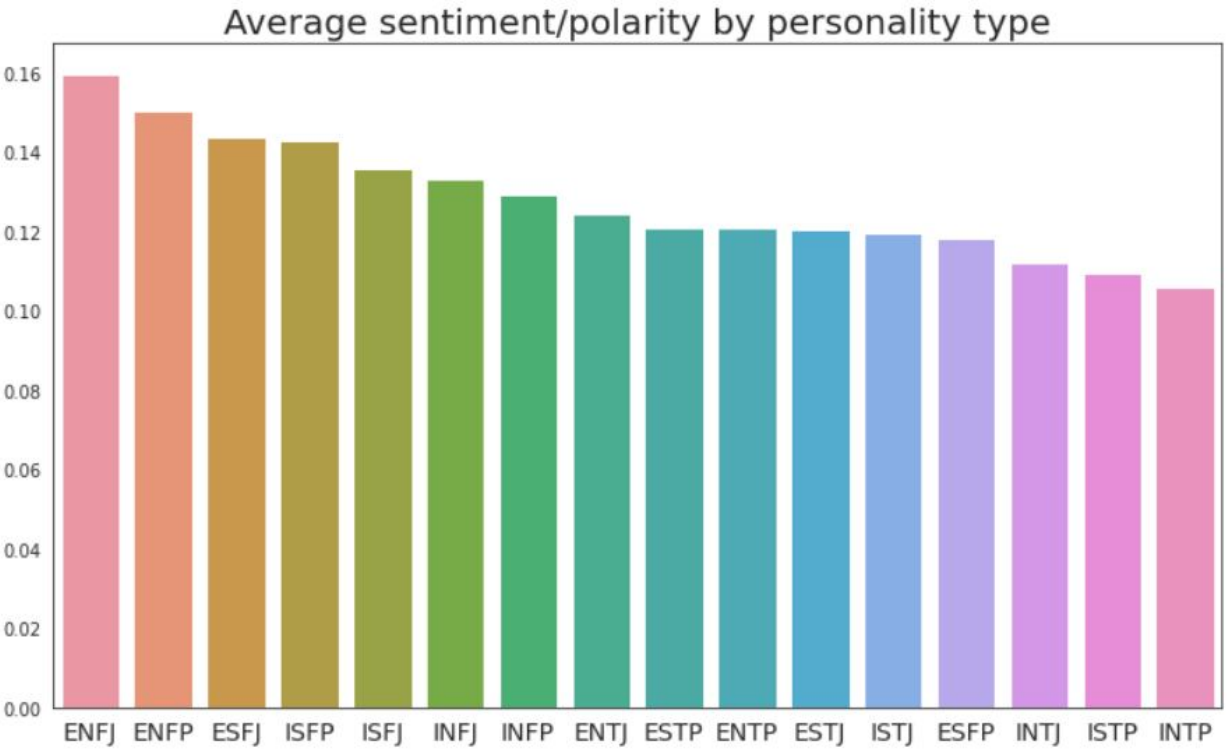

Thinking
say
type
INTJ
mean
work
good
look
need
come
well
actually
INTP
person
understand
ENTP
sure
interest
life
tell
right
probably


Sensing
mean
say
look
come
good
well
work
type
need
talk
life
person
INFP
actually
INFJ
tell
right
sure
interest

Topics do not provide valuable information on each type. A particular set of words are being repeated and modeled throughout all the 8 categories. The words clouds approve of this point by displaying almost similar terms for each category. However, it is observed that **Extroverts**, **Feeling**, **Perceiving** and **Judging** types have more mentions of their personality type compared to the other 4 categories.

Sentiments

In this step, the average sentiment is extracted for each unique personality type using *textblob*'s sentiment library.



The barplot provides a comparison of the overall sentiment (mood) for each unique personality type with **ENFJ** and **INTP** having the highest and lowest scores respectively.

Vectorization

Three vector representations are implemented and tested on *Mutinomial Naive Bayes* classifier.

- Count Vectorizer
- Tf-Idf Vectorizer
- Counter Vectorizer with custom tokenizer that turns words into lemmas, removes numbers and ignores class labels in the text to avoid possible undesired effect.

Count Vectorizer

	precision	recall	f1-score	support
ENFJ	0.00	0.00	0.00	40
ENFP	0.00	0.00	0.00	146
ENTJ	0.00	0.00	0.00	57
ENTP	0.29	0.02	0.03	128
ESFJ	0.00	0.00	0.00	8
ESFP	0.00	0.00	0.00	6
ESTJ	0.00	0.00	0.00	8
ESTP	0.00	0.00	0.00	14
INFJ	0.35	0.63	0.45	281
INFP	0.35	0.88	0.50	372
INTJ	0.74	0.10	0.17	202
INTP	0.51	0.45	0.48	288
ISFJ	0.00	0.00	0.00	32
ISFP	0.00	0.00	0.00	43
ISTJ	0.00	0.00	0.00	45
ISTP	0.00	0.00	0.00	65
accuracy			0.38	1735
macro avg	0.14	0.13	0.10	1735
weighted avg	0.32	0.38	0.28	1735

Custom Count Vectorizer

	precision	recall	f1-score	support
ENFJ	0.00	0.00	0.00	40
ENFP	0.00	0.00	0.00	146
ENTJ	0.00	0.00	0.00	57
ENTP	0.10	0.01	0.01	128
ESFJ	0.00	0.00	0.00	8
ESFP	0.00	0.00	0.00	6
ESTJ	0.00	0.00	0.00	8
ESTP	0.00	0.00	0.00	14
INFJ	0.30	0.53	0.38	281
INFP	0.33	0.78	0.46	372
INTJ	0.48	0.12	0.19	202
INTP	0.46	0.48	0.47	288
ISFJ	0.00	0.00	0.00	32
ISFP	1.00	0.02	0.05	43
ISTJ	0.00	0.00	0.00	45
ISTP	0.00	0.00	0.00	65
accuracy			0.35	1735
macro avg	0.17	0.12	0.10	1735
weighted avg	0.28	0.35	0.26	1735

Tf-Idf Vectorizer

	precision	recall	f1-score	support
ENFJ	0.00	0.00	0.00	40
ENFP	0.00	0.00	0.00	146
ENTJ	0.00	0.00	0.00	57
ENTP	0.00	0.00	0.00	128
ESFJ	0.00	0.00	0.00	8
ESFP	0.00	0.00	0.00	6
ESTJ	0.00	0.00	0.00	8
ESTP	0.00	0.00	0.00	14
INFJ	0.31	0.01	0.03	281
INFP	0.21	0.99	0.35	372
INTJ	0.00	0.00	0.00	202
INTP	0.00	0.00	0.00	288
ISFJ	0.00	0.00	0.00	32
ISFP	0.00	0.00	0.00	43
ISTJ	0.00	0.00	0.00	45
ISTP	0.00	0.00	0.00	65
accuracy			0.22	1735
macro avg	0.03	0.06	0.02	1735
weighted avg	0.10	0.22	0.08	1735

Figure 3: Comparison of vector representation performances

Provided the classification tables, the overall **accuracy** for all the three methods are:

- Count Vectorizer: **0.38**
- Tf-Idf vectorizer: **0.22**
- Count Vectorizer(custom): **0.35**

The Multinomial Naive Bayes classifier has classified all samples into four or less personality types (INFJ, INFP, INTJ and INTP) that have larger number of samples in the training set. Considering the per class F1 scores, it is observed that sample size has a direct relation with the F1 score of each class. The larger the sample, the higher the score. *Counter Vectorizer* will be used to search for the best classifier.

Within the four mentioned personality types, the INFP type that has the most number of samples result in a very high **recall** score. This is while the **precision** for this type is relatively low since the other types are misclassified as this INFP, resulting a poor score. In the next step, we will perform a number of re-sampling techniques as well as *Cross Validation* to reach a more balanced accuracy among all 16 personality types.

Resampling

The result of three resampling methods implemented using *imblearn* are as follows.

Results from *Random Under Sampler* suggests that all per class scores and the overall score have dropped significantly. This is not very unexpected since the resampler removes samples with respect to the size of the minority class, which here is considerably small. Therefore, a large amount of information is lost leading to poor prediction scores. The *Random Over Sampler* table indicates a noticeable improvement in the per class F1 scores and the model's overall accuracy. Personality types with **ENxx** and **ISxx** have received higher F1 scores compared to 0 scores obtained with the original data. This is while the **ESxx** types still get 0 scores. In the case of *SMOTE*, the overall model accuracy has increased to 0.43, however it has failed to yield a more balanced per class results.

Random Under Sampler

	precision	recall	f1-score	support
ENFJ	0.05	0.48	0.09	42
ENFP	0.26	0.17	0.21	138
ENTJ	0.35	0.17	0.23	40
ENTP	0.22	0.46	0.30	129
ESFJ	0.03	0.89	0.06	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.05	0.33	0.08	9
ESTP	0.07	0.07	0.07	14
INFJ	0.54	0.07	0.13	272
INFP	0.40	0.13	0.19	382
INTJ	0.65	0.11	0.18	244
INTP	0.58	0.04	0.08	251
ISFJ	0.10	0.36	0.16	33
ISFP	0.07	0.02	0.03	52
ISTJ	0.12	0.63	0.19	35
ISTP	0.40	0.33	0.36	76
accuracy			0.17	1735
macro avg	0.24	0.27	0.15	1735
weighted avg	0.42	0.17	0.17	1735

Random Over Sampler

	precision	recall	f1-score	support
ENFJ	0.50	0.12	0.19	42
ENFP	0.37	0.38	0.37	138
ENTJ	0.33	0.12	0.18	40
ENTP	0.50	0.47	0.48	129
ESFJ	0.00	0.00	0.00	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	1.00	0.07	0.13	14
INFJ	0.43	0.65	0.52	272
INFP	0.53	0.72	0.61	382
INTJ	0.62	0.47	0.54	244
INTP	0.48	0.53	0.50	251
ISFJ	0.40	0.06	0.11	33
ISFP	0.17	0.06	0.09	52
ISTJ	1.00	0.03	0.06	35
ISTP	0.67	0.21	0.32	76
accuracy			0.49	1735
macro avg	0.44	0.24	0.26	1735
weighted avg	0.50	0.49	0.46	1735

SMOTE

	precision	recall	f1-score	support
ENFJ	0.00	0.00	0.00	42
ENFP	0.55	0.08	0.14	138
ENTJ	0.00	0.00	0.00	40
ENTP	0.69	0.14	0.23	129
ESFJ	0.00	0.00	0.00	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	0.00	0.00	0.00	14
INFJ	0.31	0.64	0.42	272
INFP	0.41	0.81	0.55	382
INTJ	0.80	0.30	0.44	244
INTP	0.47	0.51	0.49	251
ISFJ	0.00	0.00	0.00	33
ISFP	0.00	0.00	0.00	52
ISTJ	0.00	0.00	0.00	35
ISTP	0.00	0.00	0.00	76
accuracy			0.41	1735
macro avg	0.20	0.16	0.14	1735
weighted avg	0.42	0.41	0.35	1735

Figure 4: Comparison of resampling method performances

Classifier

In this section, three classification algorithms will be evaluated primarily on the default parameter values and then we attempt to find the best hyperparameters through a grid search. The results will be compared to the baseline MNB in the preceding section.

Random Forest

	precision	recall	f1-score	support
ENFJ	0.40	0.50	0.44	42
ENFP	0.47	0.53	0.50	138
ENTJ	0.00	0.00	0.00	40
ENTP	0.00	0.00	0.00	129
ESFJ	0.03	0.56	0.07	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	0.00	0.00	0.00	14
INFJ	0.67	0.01	0.03	272
INFP	0.62	0.48	0.54	382
INTJ	0.59	0.45	0.51	244
INTP	0.51	0.35	0.42	251
ISFJ	0.35	0.58	0.43	33
ISFP	0.55	0.31	0.40	52
ISTJ	0.04	0.43	0.07	35
ISTP	0.36	0.45	0.40	76
accuracy			0.33	1735
macro avg	0.29	0.29	0.24	1735
weighted avg	0.49	0.33	0.35	1735

Support Vector Classifier

	precision	recall	f1-score	support
ENFJ	0.70	0.38	0.49	42
ENFP	0.70	0.56	0.62	138
ENTJ	0.63	0.47	0.54	40
ENTP	0.69	0.67	0.68	129
ESFJ	1.00	0.11	0.20	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	0.55	0.43	0.48	14
INFJ	0.62	0.68	0.65	272
INFP	0.67	0.81	0.74	382
INTJ	0.61	0.70	0.65	244
INTP	0.69	0.72	0.70	251
ISFJ	0.74	0.42	0.54	33
ISFP	0.68	0.50	0.58	52
ISTJ	0.90	0.51	0.65	35
ISTP	0.69	0.49	0.57	76
accuracy			0.66	1735
macro avg	0.62	0.47	0.51	1735
weighted avg	0.66	0.66	0.65	1735

Logistic Regression

	precision	recall	f1-score	support
ENFJ	0.48	0.31	0.38	42
ENFP	0.64	0.54	0.59	138
ENTJ	0.50	0.50	0.50	40
ENTP	0.68	0.65	0.66	129
ESFJ	0.75	0.33	0.46	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.75	0.33	0.46	9
ESTP	0.53	0.57	0.55	14
INFJ	0.59	0.63	0.61	272
INFP	0.72	0.78	0.75	382
INTJ	0.66	0.67	0.67	244
INTP	0.67	0.67	0.67	251
ISFJ	0.56	0.45	0.50	33
ISFP	0.49	0.58	0.53	52
ISTJ	0.51	0.54	0.53	35
ISTP	0.59	0.55	0.57	76
accuracy			0.64	1735
macro avg	0.57	0.51	0.53	1735
weighted avg	0.64	0.64	0.64	1735

Figure 5: Comparison of classifiers on default parameter settings

Support Vector Classifier and Logistic Regression model accuracy are 0.66 and 0.64 followed by the much lower 0.33 for the Random Forest model. Performing grid search on all three algorithms, the best parameter values returned are:

- Random Forest
  - ccp_alpha: **0**
  - criterion: **gini**
  - min_samples_leaf: **3**
  - min_samples_split: **30**
  - n_estimators: **200**
- Support Vector Classifier
  - C: **10**
  - Kernel: **rbf**
- Logistic Regression
  - C: **10**
  - Penalty: **l2**
  - Solver: **Saga**

Using the selected parameters, three new models are trained with below results.

Random Forest

	precision	recall	f1-score	support
ENFJ	0.74	0.33	0.46	42
ENFP	0.61	0.53	0.57	138
ENTJ	0.67	0.60	0.63	40
ENTP	0.63	0.71	0.67	129
ESFJ	0.00	0.00	0.00	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	0.67	0.14	0.24	14
INFJ	0.65	0.69	0.67	272
INFP	0.64	0.81	0.72	382
INTJ	0.66	0.62	0.64	244
INTP	0.70	0.73	0.72	251
ISFJ	0.71	0.52	0.60	33
ISFP	0.62	0.38	0.48	52
ISTJ	0.76	0.54	0.63	35
ISTP	0.70	0.59	0.64	76
accuracy			0.66	1735
macro avg	0.55	0.45	0.48	1735
weighted avg	0.65	0.66	0.65	1735

Support Vector Classifier

	precision	recall	f1-score	support
ENFJ	0.71	0.36	0.48	42
ENFP	0.69	0.51	0.59	138
ENTJ	0.63	0.42	0.51	40
ENTP	0.67	0.64	0.66	129
ESFJ	1.00	0.11	0.20	9
ESFP	0.00	0.00	0.00	9
ESTJ	0.00	0.00	0.00	9
ESTP	0.55	0.43	0.48	14
INFJ	0.59	0.69	0.64	272
INFP	0.66	0.81	0.73	382
INTJ	0.65	0.70	0.67	244
INTP	0.66	0.72	0.69	251
ISFJ	0.72	0.39	0.51	33
ISFP	0.75	0.46	0.57	52
ISTJ	0.89	0.46	0.60	35
ISTP	0.68	0.45	0.54	76
accuracy			0.65	1735
macro avg	0.61	0.45	0.49	1735
weighted avg	0.65	0.65	0.64	1735

Logistic Regression

	precision	recall	f1-score	support
ENFJ	0.63	0.42	0.51	40
ENFP	0.63	0.63	0.63	146
ENTJ	0.67	0.54	0.60	57
ENTP	0.56	0.64	0.60	128
ESFJ	0.67	0.25	0.36	8
ESFP	0.20	0.17	0.18	6
ESTJ	0.75	0.38	0.50	8
ESTP	0.44	0.29	0.35	14
INFJ	0.64	0.69	0.66	281
INFP	0.68	0.73	0.70	372
INTJ	0.67	0.68	0.67	202
INTP	0.70	0.70	0.70	288
ISFJ	0.56	0.47	0.51	32
ISFP	0.50	0.30	0.38	43
ISTJ	0.69	0.60	0.64	45
ISTP	0.58	0.54	0.56	65
accuracy			0.65	1735
macro avg	0.60	0.50	0.54	1735
weighted avg	0.65	0.65	0.65	1735

Figure 6: Comparison of classifiers with selected parameter values

There is a considerable increase in the overall accuracy of the *Random Forests* model compared to its base settings. The per class F1 score has also improved. However, the score for all the 4 **ESxx** types are 0. The results for SVC indicate a slightly lower overall accuracy of **0.64** compared to base settings. Logistic Regression gets a 1 percent increase from **0.64** to **0.65** with the selected parameter values.

LSTM

In this next phase, three configurations of RNN with LSTM (*see methods/word embeddings and LSTM*) will be tested.

Configuration 1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1945, 64)          10246080
_________________________________________________________________
bidirectional_1 (Bidirection (None, 1945, 128)         66048
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                2064
=================================================================
Total params: 10,314,192
Trainable params: 10,314,192
Non-trainable params: 0
```

In this step, the *random over sampled* data will be used for training the model.

Plots indicate overfitting of the model and a considerable gap between train and validation loss and accuracy.

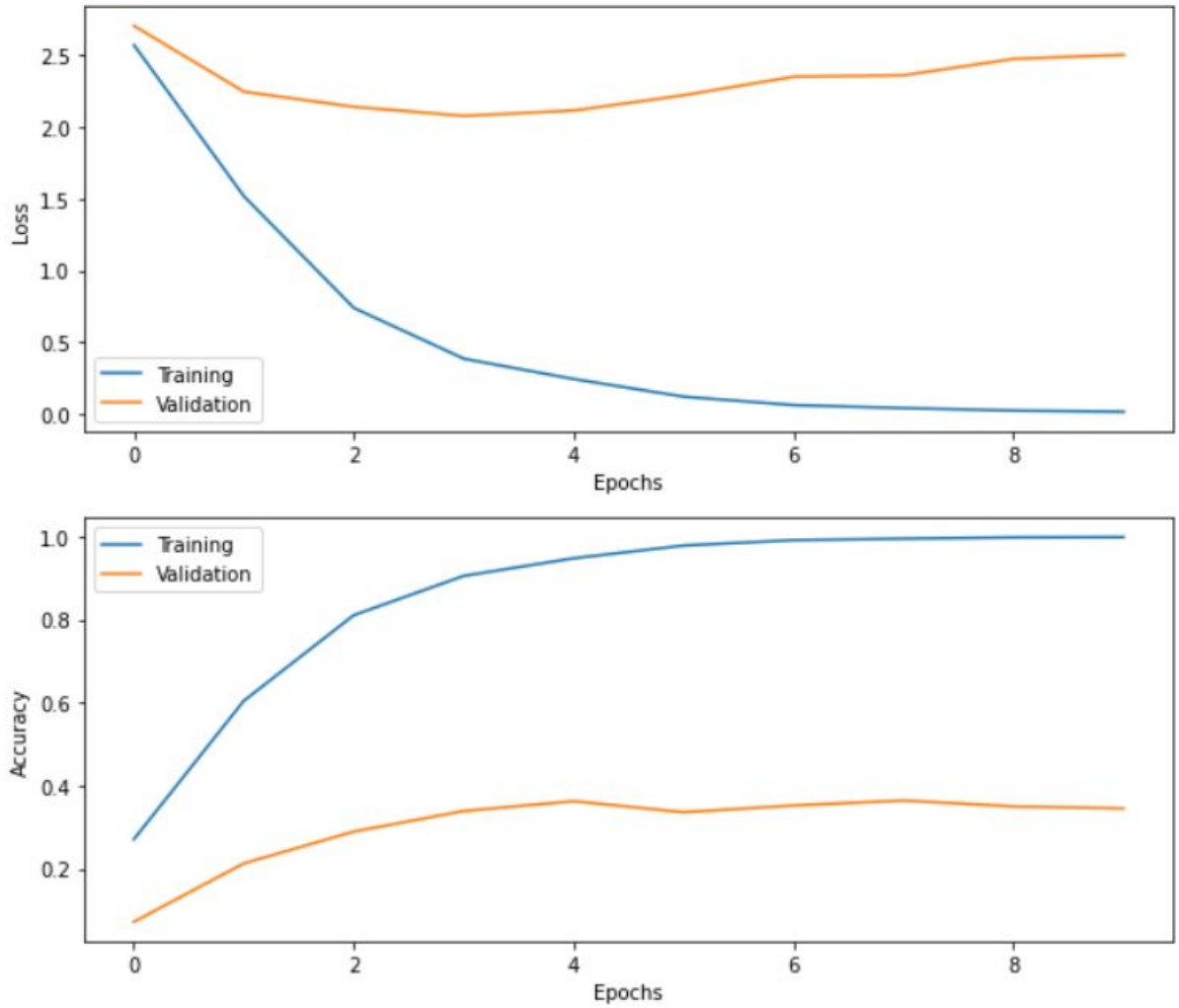


Figure 7: Configuration 1 Loss/Accuracy

Configuration 2

For this step we use configuration 1 on the original data.

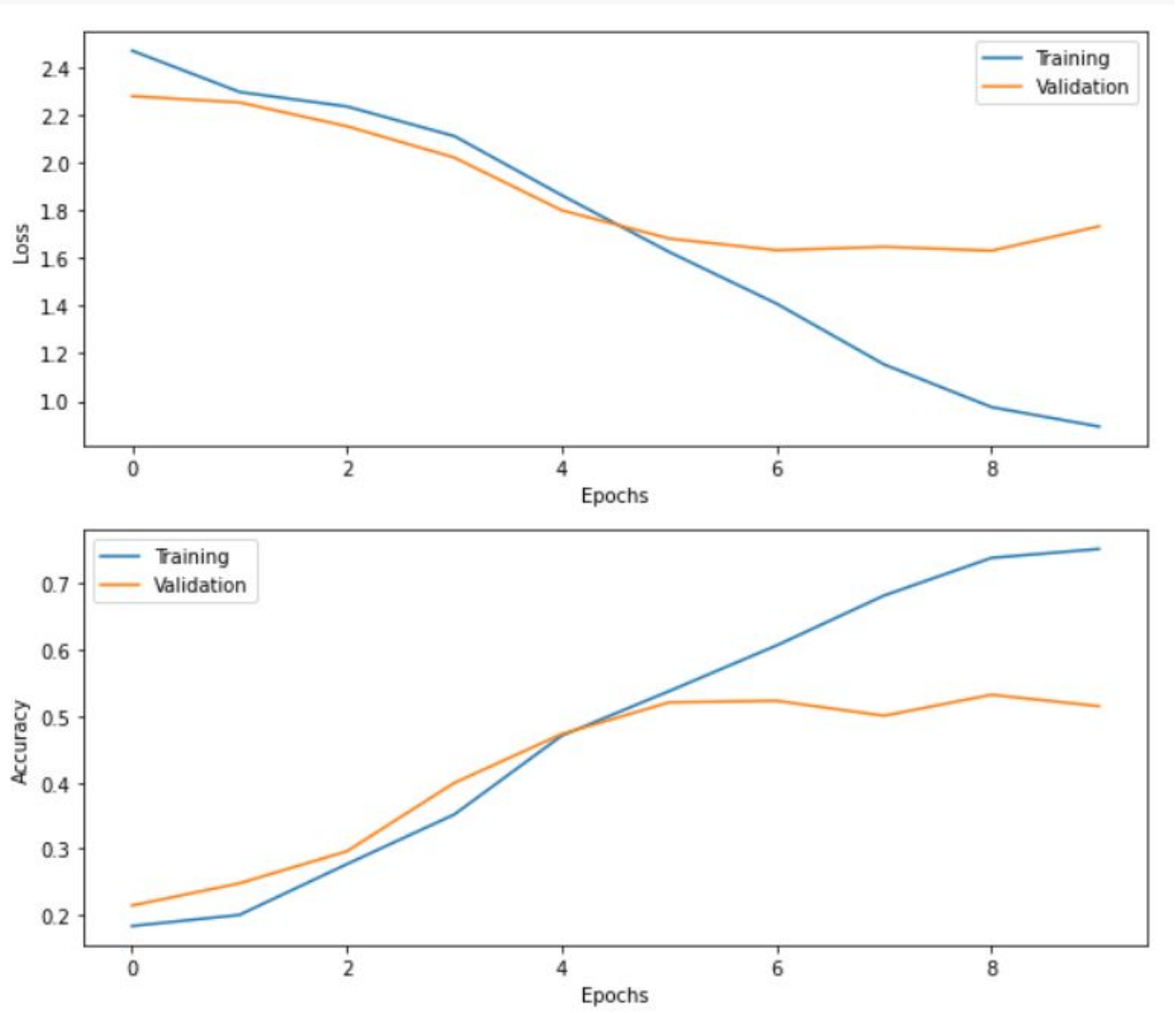


Figure 8: Configuration 2 Loss/Accuracy

There is an obvious improvement in the validation loss and accuracy until the sixth epoch. The model overfits thereafter.

Configuration 3

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 1945, 100)	16009500
dropout_2 (Dropout)	(None, 1945, 100)	0
bidirectional_2 (Bidirection	(None, 1945, 200)	160800
global_max_pooling1d_2 (Glob	(None, 200)	0
dropout_3 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 16)	3216

Total params: 16,173,516
Trainable params: 16,173,516
Non-trainable params: 0

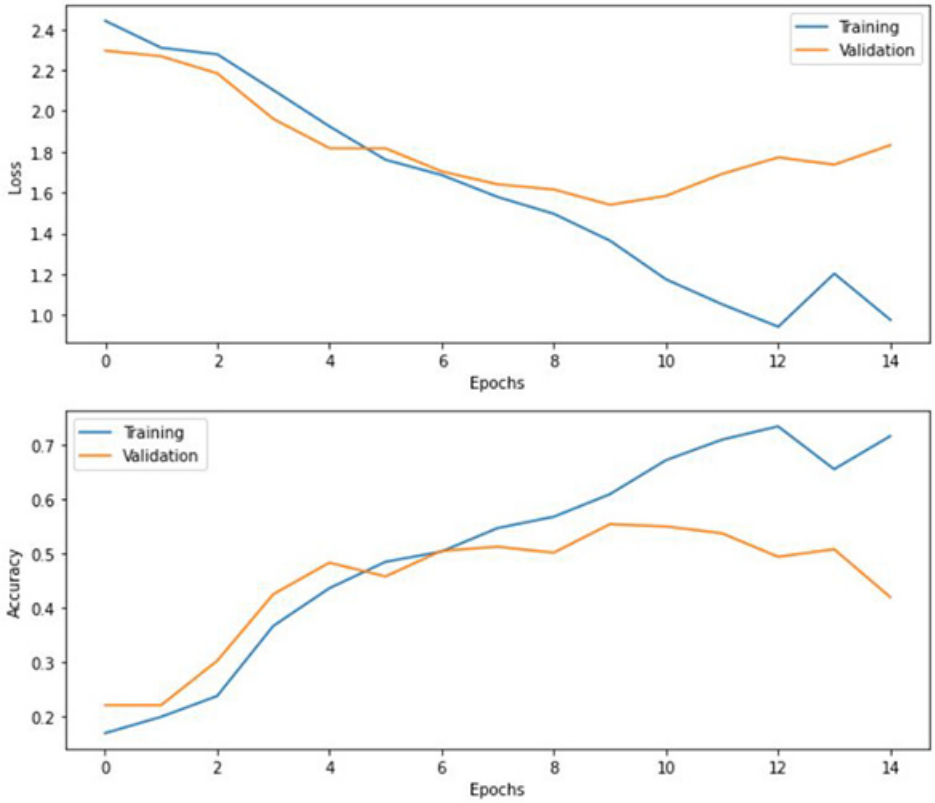


Figure 9: Configuration 3 Loss/Accuracy

Overall behavior of the model is similar to configuration 2. The training process seem to be more unstable. This configuration was trained for 15 epochs (as opposed to 10 epochs for configuration 1 and 2) but the plots do not indicate model improvement.

	Config.1	Config.2	Config.3
Train Accuracy	1.00	80.7	68.5
Test Accuracy	0.36	51.5	43.7

Personality type prediction using Twitter data

Considering close accuracy scores of all classifiers the **Random Forest Classifier** will be selected as the best model given its considerably faster training speed compared to SVC and Logistic Regression as well as it's higher accuracy compared to LSTM models.

We use the classifier on +43,000 tweets from Donald Trump to predict his personality type.

```
array(['ESTJ'], dtype=object)
```

Figure 10: Predicted Trump's personality type

Discussion

Results provided by practicing resampling techniques in this report were not unexpected. Undersampled dataset decreased accuracy provided that the number of samples in every class was reduced to the number of minority class, which in our dataset is significantly small, resulting a massive loss of information. One might expect to obtain better result with SMOTE oversampling compared to random oversampling but study shows "while in most cases SMOTE seems beneficial with low-dimensional data, it does not attenuate the bias towards the classification in the majority class for most classifiers when data are high-dimensional, and it is less effective than random undersampling." (Blagus and Lusa 2013)

Naive Bayes Classifier saw the biggest improvement after over sampling the data because with more instances of minority classes, their prior probabilities were improved, thus improving classification results. *Support Vector Classifier* and *Logistic Regression Classifier* performed almost identical using either default or selected hyperparameter values returned by grid search. However, they both performed relatively well with respect to observed class F1 scores and the model's accuracy on baseline settings. The performance of *Random Forest Classifier* on the other hand was doubled after training with the optimized model. The range of parameters in random forests is large and thus the algorithm allows for a model that is tailored to the specifics of the task at hand. Not only there is high control over how each tree is grown, a fine control is also available over the bootstrapping step of the algorithm. All three classifiers performed almost identically in terms of overall accuracy. As a result, the training speed was observed to reach a conclusion. From the fastest to slowest:

1. Random Forest Classifier
2. Logistic Regression Classifier
3. Support Vector Classifier

The oversampling strategy created a negative effect when training the LSTM network, resulting in a clearly overfit model. Although neural networks are known to perform well with larger sets but since the oversampled data here simply contains repeated samples of the original instances, the model overfits to the training data. As for the second and third configurations, the same model's accuracy on the test set improves. However, the validation does not improve beyond the sixth epoch and overfits thereafter.

For prediction of Trump's personality type, one should note that a person's MBTI personality type is not assigned but rather agreed upon by the community. Donald Trump's personality type is believed to be **ESTP**. Using more than 43,000 of Trump's tweets the final model predicted his personality type as **ESTJ**. Assuming ESTP to be the true type, the best model has managed to predict 3 out of 4 of his personality functions correctly.

A similar study by Stanford University (Hernandez and Knight, n.d.) on the same dataset predicts Trump's personality as ESTP. The paper explores different variations of LSTM networks with a segmented dataset, as

opposed to oversampled set used in this paper. The final solution provided by the paper is a regular LSTM. Given a text from a user, the model calculates the mean of the predicted probability for every segment/pair (i.e. Introvert/Extrovert etc) and rounds it to either 0 or 1 at a certain threshold (i.e. 0 for Introvert, 1 for Extrovert). This probabilistic model has an advantage of providing probabilities rather than assigning a class deterministically. However, a bad choice of threshold to round the probabilities can lead to erroneous results. In addition, the model accuracy for every pair, except for the *Feeling/Thinking* pair, is between 63 to nearly 68 percent that is almost on par with results obtained by this report.

Conclusion

The MBTI dataset comes with two major drawbacks. It is relatively small and imbalanced. Every attempt to classify personality types based on this data is bound to resolve these two shortcomings. This however, has provided room to countless creative methods that try to overcome this issue.

This paper's aim was to find the simplest model with the best performance. It began with basic pre-processing techniques and simple models settings and gradually increased complexity to reach a conclusion. The findings of this paper suggest that less complex models can at times, perform as well or even better than the more popular deep networks while remaining more efficient with respect to implementation and computational cost. This is not to claim that the final model cannot be improved. Provided the paper's approach to try a large spectrum of techniques rather than focusing on a single field, it trades off deeper exploration of certain methods over providing a wider vision on how various techniques perform with this dataset.

Codes

Code documentation of this paper can be found at: https://github.com/alieti/mbti_classification

References

Amidi, Shervine. n.d. "RNN." Stanford. *Recurrent Neural Networks Cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

"Big Five Personality Traits." 2021. *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Big_Five_personality_traits&oldid=999779216.

Blagus, Rok, and Lara Lusa. 2013. "SMOTE for High-Dimensional Class-Imbalanced Data." *BMC Bioinformatics* 14 (1): 106. https://doi.org/10.1186/1471-2105-14-106.

Brownlee, Jason. 2017a. "How to Develop a Bidirectional LSTM for Sequence Classification in Python with Keras." *Machine Learning Mastery*. https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/.

———. 2017b. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning." *Machine Learning Mastery*. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

———. 2020a. "Random Oversampling and Undersampling for Imbalanced Classification." *Machine Learning Mastery*. https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/.

———. 2020b. "One-Vs-Rest and One-Vs-One for Multi-Class Classification." *Machine Learning Mastery*. https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/.

Chawla et al, Nitesh V. 2002. "Smote: Synthetic Minority over-Sampling Technique. Journal of Artificial Intelligence Research."

Drenth, A. J. n.d. "Myers-Briggs / MBTI in the Age of the Big Five." *Personality Junkie*. Accessed January 14, 2021. https://personalityjunkie.com/04/myers-briggs-mbti-vs-big-five/.

Hernandez, Rayne, and Ian Scott Knight. n.d. "Predicting Myers Briggs Types Indicator with Text Classsification." Stanford University. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6839354.pdf.

Malik, Usman. n.d. "Python for NLP: Introduction to the Pattern Library." *Stack Abuse*. Accessed January 14, 2021. https://stackabuse.com/python-for-nlp-introduction-to-the-pattern-library/.

"Max-Pooling." n.d. Wiki. *Max-Pooling/Pooling*. https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.

"(MBTI) Myers-Briggs Personality Type Dataset." n.d. Accessed January 14, 2021. https://kaggle.com/datasnaek/mbti-type.

Olah, Christopher. 2015. "Understanding LSTM Networks." Github. *Understanding LSTM Networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

"Personality Cafe." n.d. *Personality Cafe*. Accessed January 14, 2021. https://www.personalitycafe.com/login/.

"TfIdf Transformer." n.d. *TfIdf Transformer*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#sklearn.feature_extraction.text.TfidfTransformer.

"Trump Tweets." n.d. Accessed January 14, 2021. https://kaggle.com/austinreese/trump-tweets.