# AI Final Assignment: Scientific Abstract Classification

Author: Lin Chung-Hsi

Student ID: 26002304790

Course: 31889: Artificial Intelligence (G1)

Lecturer: Assistant Professor Mate Kovacs

Submission Date: January 20th, 2025

## I. Introduction

### I.I Overview

Text classification is a classical implementation of Artificial Intelligence in general. In the process of implementing one, various techniques and preprocessing are required to achieve high accuracy of the trained model. There are generally three main procedures that are needed in the making: data collection and preprocessing, model selection and construction, training and tuning. This report will go into detail about the process of training an artificial neural network for the task of classifying journal abstracts from three fields of studies. The three fields of studies in name are Psychology, Sociology, and Political Science. This report will go on to explore various configurations of a Feed-Forward Neural Network and the hyperparameters in training to achieve optimal performance.

### I.II Background

Feed-Forward Neural Network, or FNN, is the simplest kind of neural networks. Under the context of multiclass classification, the process FNN undergoes can be condensed to mapping an array of numeric values to another array of numeric values. The input array has a label representing its true class, and the output array represents the prediction for such underlying label for the input array. The three main components to a FNN are the input, hidden, and output layers. To understand the purpose of each component (and sub-components), we need to elaborate on how a FNN produces a prediction. The process of prediction goes as such:

1. Transform the data into a fixed size array of values that represents the features of the data
2. Input layer receiving the array, and passed the data though linear transformation, aggregating the values of all neurons in the input layer. The linear transformation is parameterized by weights and biases. In FNN, all neurons between two layers are connected, so there will be n*m weights and biases in between layers. The values calculated for each neuron in the first hidden layer will then first pass through an activation function that usually limits the values to a certain range non-linearly. The neurons of the first hidden layer will take these as their value. This process is called the forward-propagation
3. For the number of hidden layers there are for a network, forward-propagation is performed sequentially one after another in between each hidden layer until the last.
4. The last forward-propagation happens between the last hidden layer and the output layer. The values the output layer holds are being represented as a prediction for the true label.

The biggest problem about FNN is the large number of parameters to tune. The initial values can be assumed to have low accuracy. We can calculate the error through a loss function and adjust the parameters according to the loss. The adjustment is performed by a process called backward-propagation. Backward-propagation uses chain rule to back track the influence of each parameter. Influence in backward-propagation is represented through calculating gradient. It is how much the loss changes with respect to changing the values of a particular weight or bias. The gradient represents the direction where this value should be adjusted towards. Finally, using these gradients, we can adjust the parameters with a learning rate, which determines how much we move towards these directions. This progress would make the network predict labels more accurately.

## II. Methodology

### II.I Data Acquisition

The data set that is used in the report is web-scraped from an online journal database organization called Crossref. For acquiring the data, a python program is created to perform HTTP conversation with Crossref's server. The list below is a generalized procedure the code performed:

1. Request 999 papers with the name of one field of study as the query. (Maximum number of papers per request is 999)
2. Eliminate responses that have invalid abstracts
3. Repeat step 1 and 2 until the accumulated valid papers reaches target number

4. Save the valid papers into a csv file titled with the field of study
5. Repeat step 1 to 4 for the two other field of study

As the program is executed, there will be three csv file generated corresponding to each field of studies. In each file, there are three columns, “DOI”, “Abstract”, “Label”. Doi is saved since removing duplicates and be easily done with matching Doi. Finally, 2400 rows the data is obtained (800 per field), however, we cannot just use these abstracts as it is for training. Preprocessing is essential for improving accuracies.

**II.II Preprocessing**

Preprocessing is the procedure of filtering, altering, and removing unnecessary components in the dataset to prepare the dataset to be used for training. Looking at the results of data acquisition, there are an abundance of artifacts that may affect the results. Preprocessing, to remove these artifacts becomes important. This report is going to implement TFIDF to vectorize the data. TFIDF, or term frequency-Inverse Document Frequency, is a method to quantify textual data. Simply, this method would turn a given set of texts into an array containing the information of all texts. Mainly, TF of TFIDF focuses on, as the name suggests, frequency of a term. For example, “sentiment” can be a frequent word in psychology abstracts, but appears less often in political abstracts, TF would quantify this information in each abstract. As for IDF, measures the rarity of a term across all texts. This would accommodate words that appear frequently in abstracts of multiple labels. However, there are artifacts that would be a bad influence on the effectiveness of this technique. These are, essentially, noises to the vectorizers. Removing these artifacts would be necessary. Here is a table of the artifacts, the reasons to remove them, and the methods used to remove them.

| Artifact | Removal Rational | Methods used for Removal |
|---|---|---|
| HTML tags | Not part of the abstracts | Capture with Regular Expression |
| Punctuations | Doesn’t benefit performance since the TFIDF value for these are meaningless | |
| Low word count | Not abstracts most of the time | Check word count |
| Duplicates | Result in inaccuracy performances | Pandas built in drop_duplicate function |
| Stop words | Doesn’t benefit performance since they are meaningless | Download English stop words from library nltk. |
| Same word in different forms | TFIDF scores would be inaccurate since they are considered different words for the TFIDF vectorizer. | Stemmer module from nltk lemmatize all terms in the data. |

After removing these artifacts, these preprocesses would allow the TFIDF vectorizer to perform vectorization adequately.

### II.III Implementation Techniques

As mentioned, the encoding technique for the input layer is TFIDF, which will transform the text data into an n-dimensional vector. TFIDF makes input layer size simply being n, but the value for n is not straightforward. It is identified that the lexicon size of this dataset is around 13000 unique terms. N must be equal to or smaller than the number of unique terms in the dataset, or corpus. However, if n is too small, we would experience loss in information. In this vector, there can also be less meaningful information, for example, "abstract" is a meaningless word in classifying abstracts. Essentially, not all 13000 terms serve a purpose to help predicting the label. Shortening the feature length can be helpful in reducing the complexity of the data and the network. However, there isn't any immediate evidence of how much we can shorten it. There needs to be experimentation to explore the optimal length of the vector.

The output layer of this network is going to implement two techniques. First, one hot encoding is going to be implemented. It suggests that a unique class would correspond to the activation of one and that one neuron only. It determines the size of the output layer, being the number of classes. An example for one hot encoding for this network would be [1, 0, 0] representing psychology. An aspect of interpreting this encoding method is to encode data as probabilities of being a particular label. Since the output is not guaranteed to form a set of probability, a SoftMax layer can be applied after the output layer. What a SoftMax layer does is to normalize the output layer to scale each neuron to ensure the values sums to one.

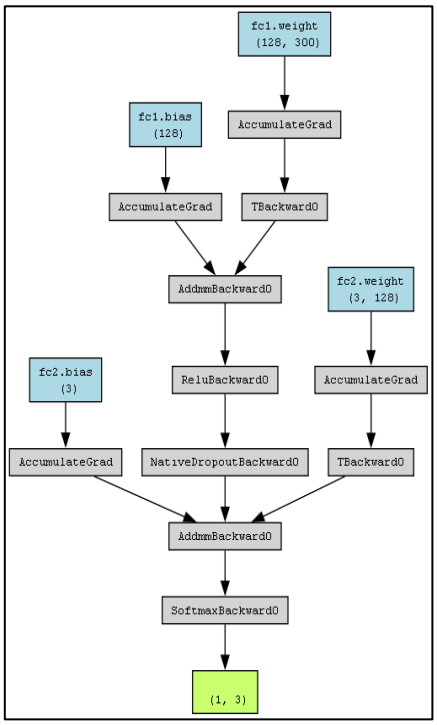
First, K-fold cross validation is going to be implemented. K-fold is a technique where the dataset is divided into k subsets. With k iterations, k models are going to be trained. For each model, one of the subsets is served as a validation set, and the rest are served as a training set. After k iteration, we would result in k models that together ensure that all data are used in both training and validation. The aggregation of these models would average performance metrics and form less biased results.

An important detail in training is in regard to fitting. During training, accuracy for the validation set would grow and peak after a certain number of epochs, then decrease afterwards. Before peaking suggests underfitting, where training can continue to boost accuracy. After peaking suggests overfitting, where any further training will only reduce validation accuracy. The objective is to stop training at the peak. To precisely terminate training, we can implement early stopping, a regularization technique that prevents further training when training does not gain improvement for the validation set. At each epoch, we evaluate the model with the validation set and record the accuracy. Then, we can compare the accuracy of the current epoch with the highest accuracy so far. If accuracy decreases under a certain tolerance from the highest accuracy score, we terminate the training.

## III. Experimentation and Exploration

### III.I Feature Length

Figure 3.1. Neural Network Structure of 1 hidden layer with 128 Neurons




As mentioned, using 13000 as the feature length for TFIDF is not entirely effective. Here is the process to explore the effect of feature length to model accuracy. First, with feature length of 13000, through testing in many network configurations, from single hidden layer with 2 neurons to 3 layers each with 8192 neurons, the best accuracy observed can only reach around 90%. Then, we can try to reduce the feature length and observe whether losing information affects accuracy. Figure 3.2 plots accuracy against the feature length trained with a particular effective network configuration (Figure 3.1). As shown, reducing feature length all the way to 100 has essentially no difference in accuracy. A probable explanation can be that after the first 100 most frequent unique terms, TDIDF scores for these terms diminish. Suggesting the first 100 terms containing the significant information that is needed to make accurate predictions.

Figure 3.2. Feature Length against Average Accuracy (100 - 10000)

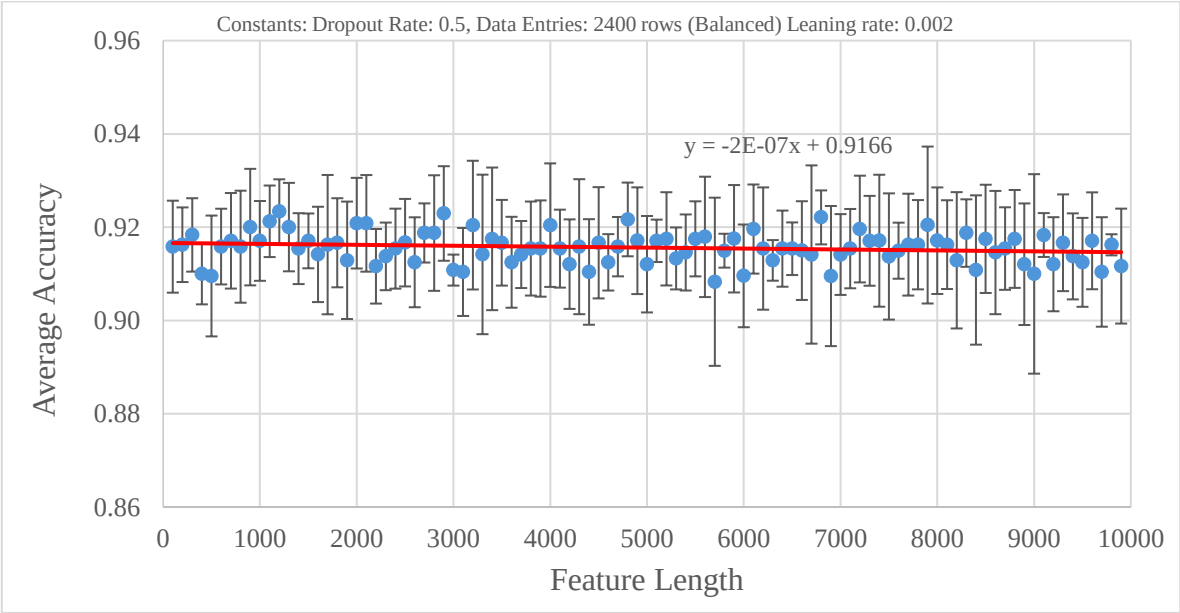
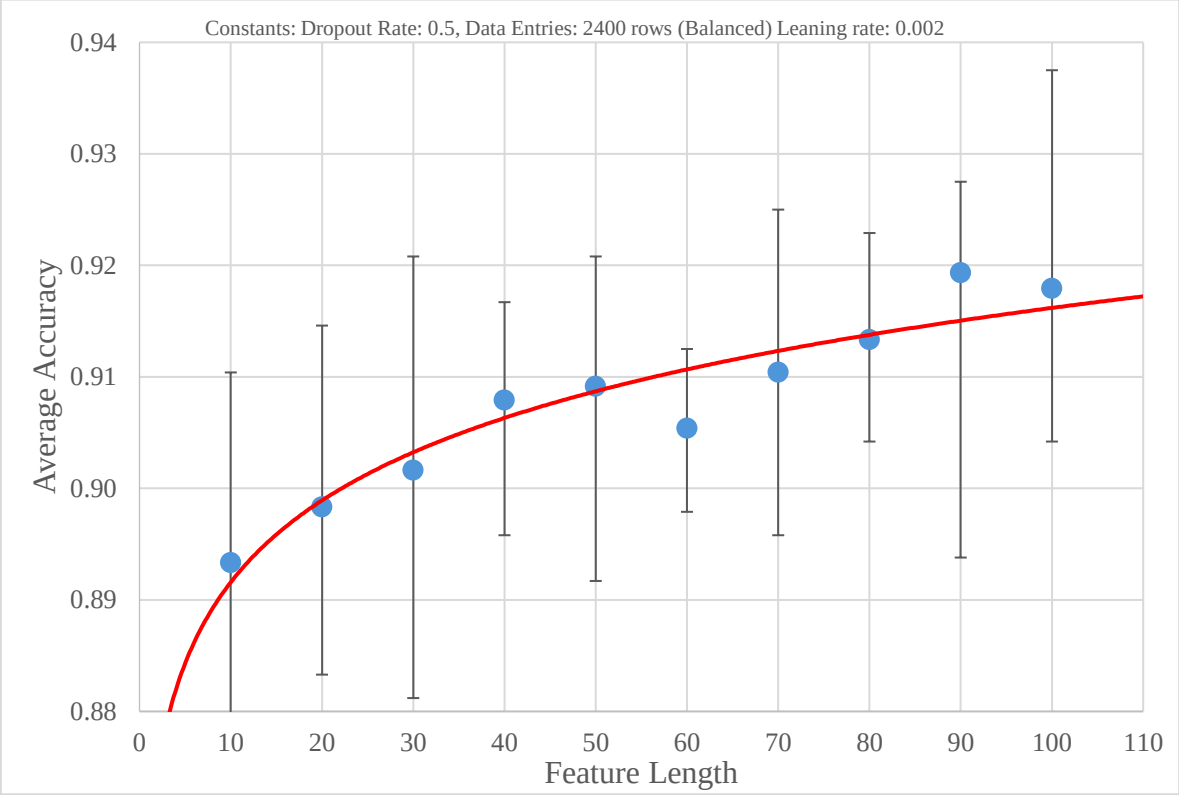


Figure 3.3. Feature Length against Average Accuracy (10 – 100)




As shown in Figure 3.3, feature length of 10 could only yield average accuracy at 89.3%, which is more performant than expected, but, as feature length increases linearly towards 100, average accuracy increases to 91.8%. Here is a clear demonstration of the network's behavior at low feature length. Then, in Figure 3.2, starting from 100, the trendline poses a downward trend of 0.1% in accuracy for every 1000 increment in feature length. Such a trend would also terminate at the lexicon size of the corpus, which is 13000. Considering the error bars, the slope isn't reliable, positive trends exist within the errors. Therefore, it can be said that between 100 to 13000, the effect of feature length on model accuracy is irrelevant. Though this causes the optimal feature length to be inconclusive, it does show that reduction in feature length causes minimal information loss. It means that we can choose feature length a lot shorter than the size of the corpus, reducing the complexity of the network. However, during training, feature length below 1000 would take more epochs to finish training (which means to reach early stopping conditions). Learning is slower for short feature length; this suggests that the network is struggling to extract patterns from the data. For what is observed from the experiment, it is safe to continue with feature length of 1000 considering both in reduction of complexity and prevention of excessive information loss.

### III.II Network Hidden Structure

In regard to the complexity, comes an important question of how the structural design of the network should be. How many hidden layers and how big should these layers be? We can solve this problem by experimentation. A primitive but powerful method is to start with one hidden layer consisting of a few neurons and gradually increase the size and number of layers. Here are graphs that demonstrate accuracies of models with one to three hidden layers varying from 4 to 4096 neurons. Figures 3.4 to 3.7 share the same constants.

Figure 3.4. Number of Neurons against Accuracy for one Hidden Layer

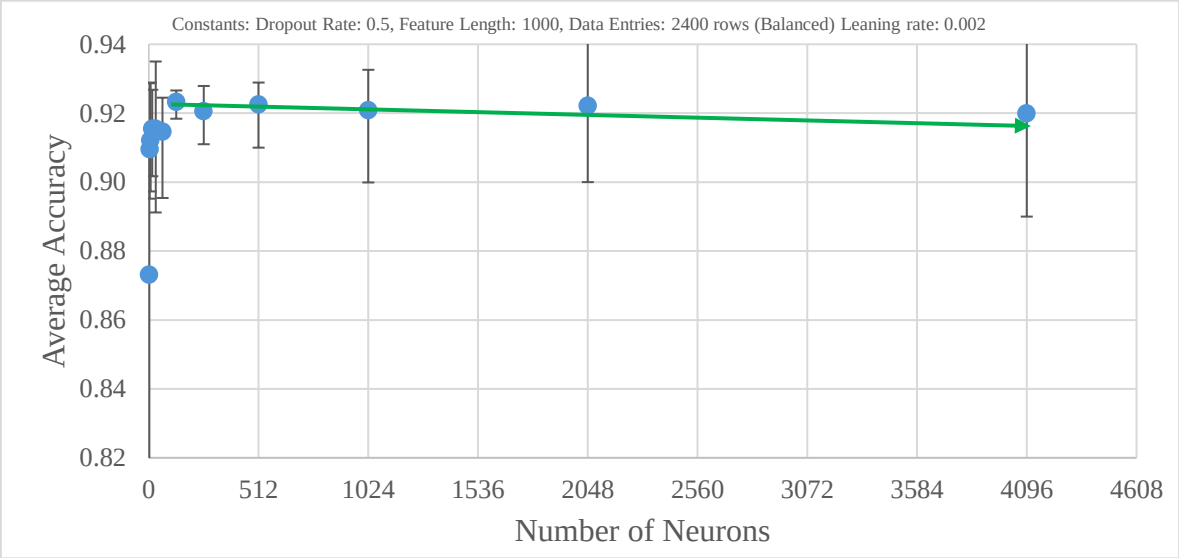



Figure 3.5. Number of Neurons against Accuracy for Two Hidden Layer

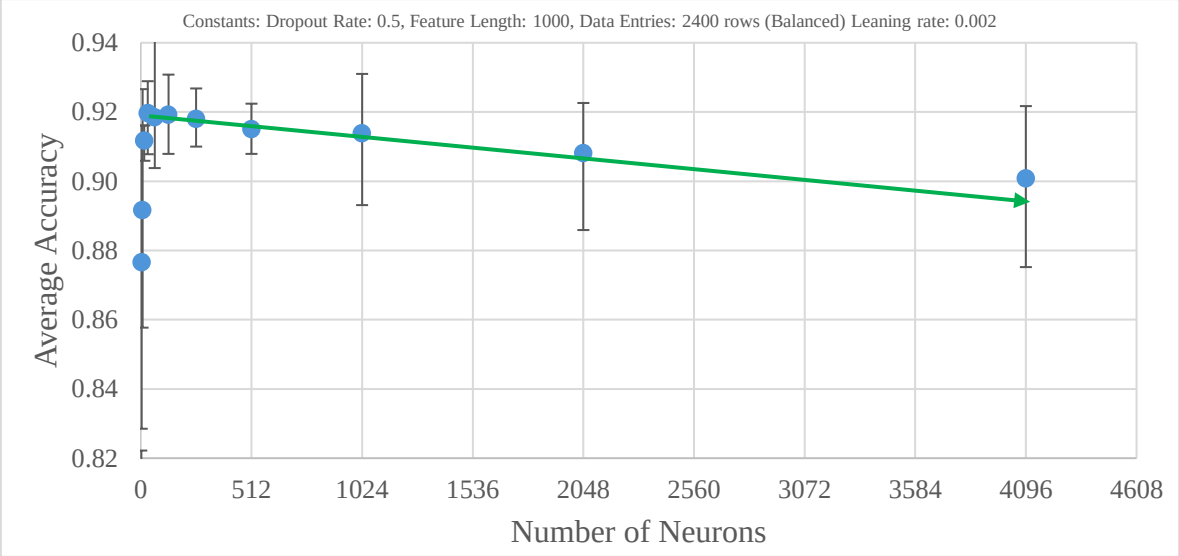
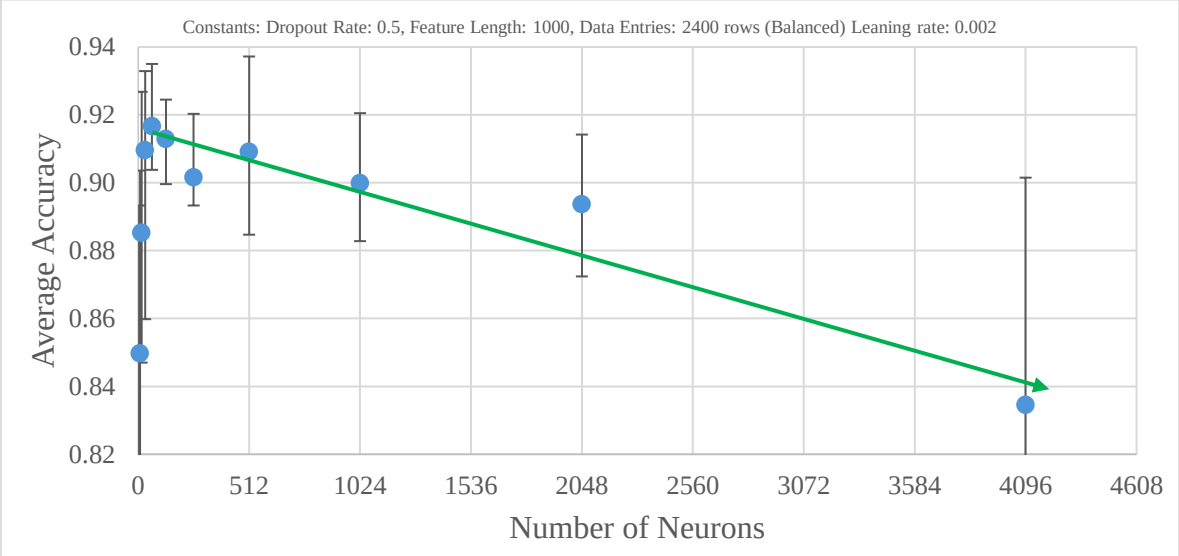


Figure 3.6. Number of Neurons against Accuracy for Three Hidden Layer




As we can see, there are obvious trends that related to both increment of neurons and increament of layers. In all three graphs, networks that has 2 and 4 neurons carries low accuracy. Then, as neurons increases to 128, accuracy peaks. When we keep increasing neuron after 128, accuracy decreases linearly. Note that the green arrows are used to illustrate the general direction of the trend, but they not calculated approximation curves. Among the three figures, as layer increases, accuracy decreases with higher rates. It can be speculated that there is a limit to the network's strucutural complexity. If a network is more complex than the data being fed in, the network can, instead of learn, "rememeber" the data. Especially, it is observed for large networks, training completes after just 1 or 2 epochs. This essentially suggests that overfitting is inevitable, as the network memorizes the data running through the it once or twice. Generalization is harder to happen since the network has high capacity for complex data, but data complexity failes to matach to utilize the capacity. The contrary, where complexity of the network undermatches data complexity, generalization becomes harder because the model cannot encapsulate informations the differentiate the data. An ideal senario is when the complexity of the data and the network are similar. Looking for an optimized configuration, networks with single layer configuration between 128 to 512 neurons has average accuracy consistently over 92%. Focussing on the error bar for these data points, increament in error is correlated with high neuron counts. Single layer with 128 and 256 neurons has generally more stable and consistent accuracy results. Both have the same standard deviation of 0.3848%, where other data points can easily exceed 1%.

Figure 3.7. Number of Neurons to Average Accuracy for one Hidden Layer (128 – 256)

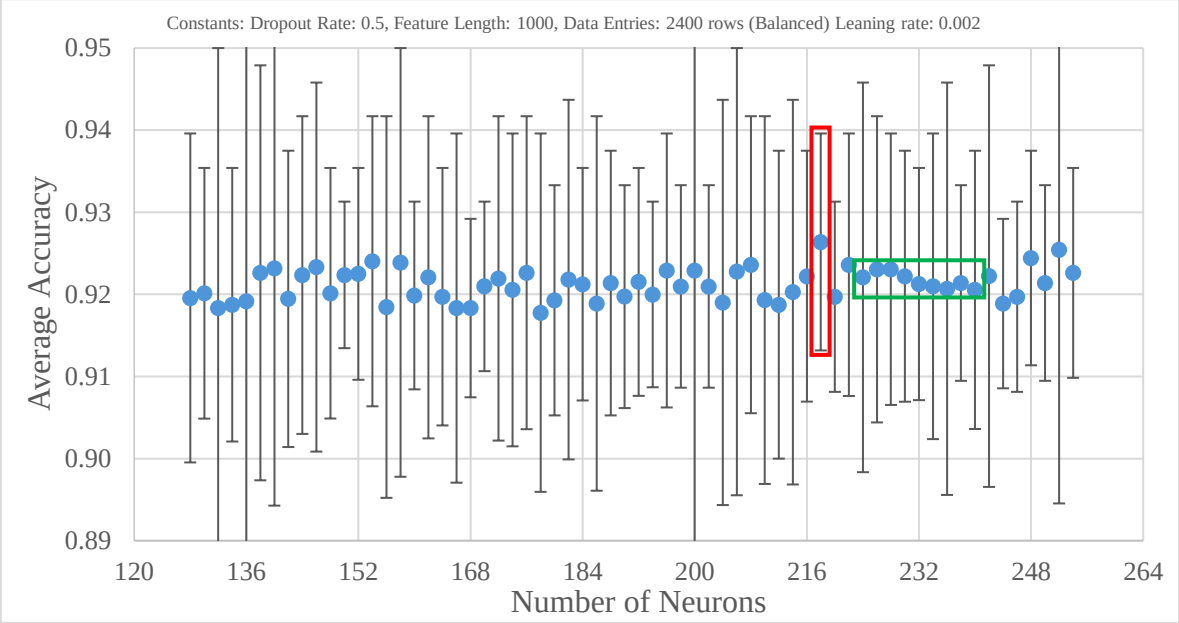



Figure 3.8. Render of Resulted Neural Network Structure

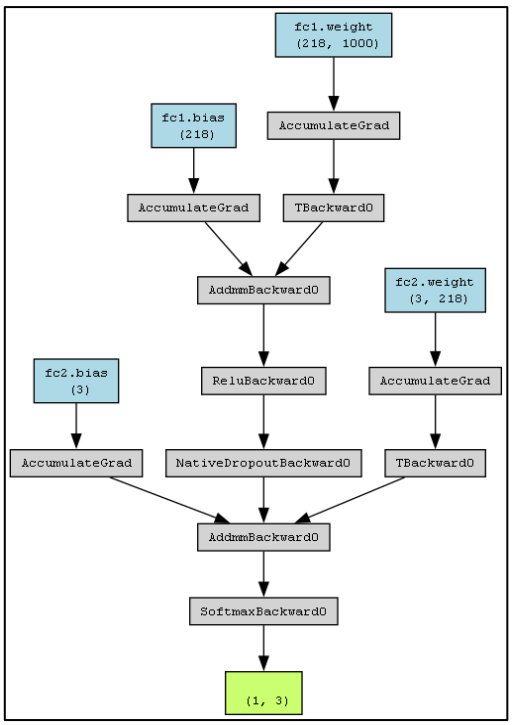
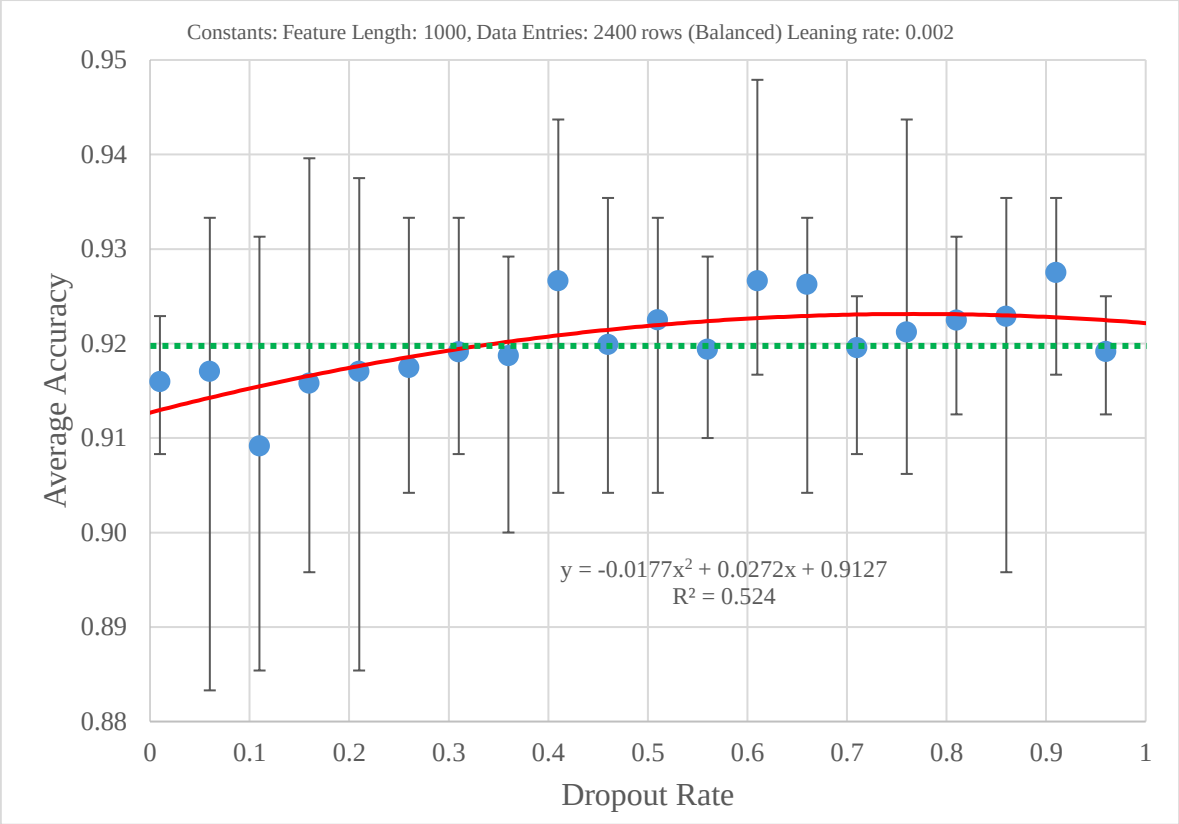



Figure 3.7 displays further experiementation with neuron counts of single hidden layer with smaller steps. Each data points are averaged between three 5-fold trainings, or 15 models. The red box represents a particularly well performing configuration of 218 neurons. Right next to the red box, a green box represents a range of configuration that is particularly consistant, between 214 to 240 neurons. The fact of having high performance and stability around this range is evident that complexity between the model and data reached some abitrary balance. We can assume the configuration of 218 neruons is optimal. To assure this assumption, one hundred 5-Fold training of identical training on this configuration is executed. Average accuracy between all 500 models is 92.659% with one particular 5-fold set exceeding average of 93%. The result is acceptable, so further experiments will continue with this configuration. The resulting neural network structure is illustrated through Figure 3.9, with dimensions of weight and biases adjusted to the new layer size.

### III.III Dropout Rate

Dropout rate is an important factor affecting the training process. Explained in the background section, dropout rate determines the proportion of neurons in a layer that is going to be randomly deactivated. Choosing the appropriate value can prevent overfitting, achieving higher performances. With the neural network's structure properly defined, we can start experimenting with varying dropout rates.

Figure 3.8. Dropout Rate against Average Accuracy

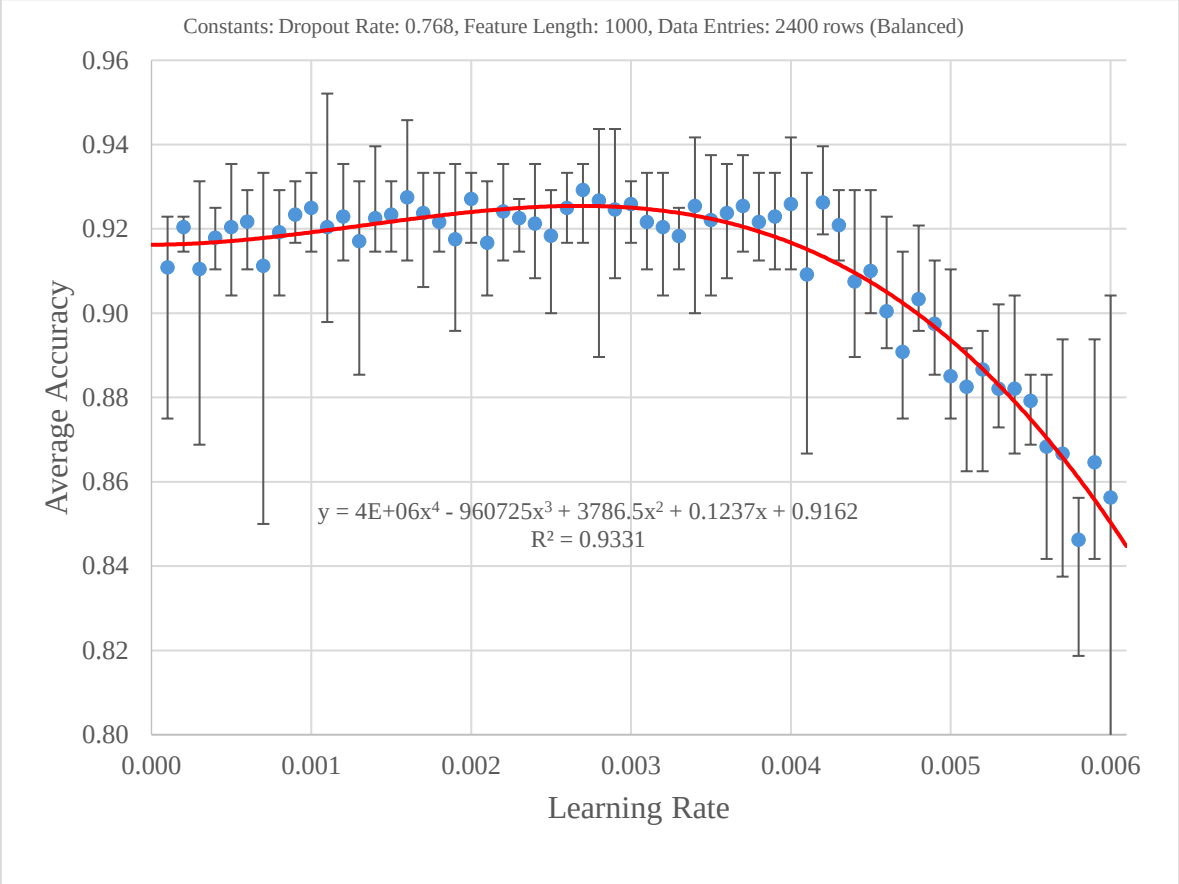



Evaluating the optimal dropout value is straightforward. Dropout rate approaching 0 presents no benefits to training and approaching 1 causes training to essentially halt (when all neurons are deactivated). Therefore, we can assume the effect of dropout rate can be represented through approximating to a parabolic curve, and the vertex presents the optimal value. The vertex of the approximation curve in Figure 3.8 suggests the optimal dropout value as 0.76836. Another information we can extract is that accuracy for dropout rate between 0.41 to 0.91 is greater than the rest of the data points, separated by the horizontal green line. This is also evident that the optimal value should exist between this range, which the vertex does.

### III.IV Learning Rate

Learning rate is the hyperparameter that determines the significance of changes applied to weights and biases during training. Proceeding with the wrong values may cause the network to be potentially stuck in a local extremum. To push the network out of local extremums, we need the learning rate to be large enough. However, having a large learning rate may cause the network to never converge at an extremum. This presents a similar story with dropout rate, deviating from the optimal value causes predictable decrease in accuracy, that forms a parabolic trend.

Figure 3.9. Learning Rate Against Average Accuracy




Interesting, the data formed a trend where a quartic expression better approximates the results. It can be speculated that overshooting learning rate does present a parabolic decreasing trend, but undershooting does not. Even with learning rate approaching 0, the vector space formed by the TFIDF vector does not have a lot of local extrema that is hard to overcome. Since there is no need to explore the learning rate to plummet the accuracy, the vertex of this quartic curve is 0.00264376.

## IV. Results

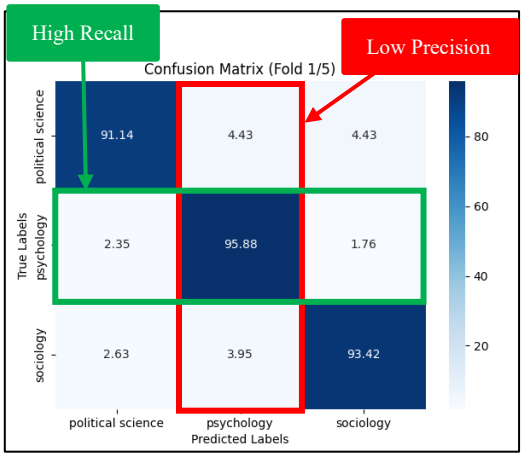
Figure 4.1. Console Output of Training the Optimized Model

```
(wenv) PS C:\Users\LeoLi\OneDrive - 学校法人立命館\Ritsumei\4th_Semester\Artificial_Intelligence\final_project\code> python FNN
/LinChungHsi_aifinal.py
Using device: NVIDIA GeForce RTX 4060 Ti
Training with Vectorized Data 1000 features. Dropout Rate 0.768. Dataset Size 2400. 218 neurons. Learning Rate 0.00264
        Fold 1 Validation Metrics - Accuracy: 0.9354, Weighted Precision: 0.9357, Weighted Recall: 0.9354, Weighted F1: 0.9353
        Fold 2 Validation Metrics - Accuracy: 0.9354, Weighted Precision: 0.9354, Weighted Recall: 0.9354, Weighted F1: 0.9354
        Fold 3 Validation Metrics - Accuracy: 0.9333, Weighted Precision: 0.9335, Weighted Recall: 0.9333, Weighted F1: 0.9332
        Fold 4 Validation Metrics - Accuracy: 0.9250, Weighted Precision: 0.9256, Weighted Recall: 0.9250, Weighted F1: 0.9251
        Fold 5 Validation Metrics - Accuracy: 0.9146, Weighted Precision: 0.9168, Weighted Recall: 0.9146, Weighted F1: 0.9145
Average Validation Accuracy (Over 5 folds): 0.9287500000000002
Confussion matrices and ROC curves are saved in: C:\Users\LeoLi\OneDrive - 学校法人立命館\Ritsumei\4th_Semester\Artificial_Inte
lligence\final_project\code\FNN/model/2025-01-17/05-58-05//figures/

Model 1 - 5 :
        Accuracy of folds:      [0.9354, 0.9354, 0.9333, 0.9250, 0.9146, ]      Average accuracy: 0.9288
        Metrics for Fold 1
                political science :     F1 score: 0.9290, Recall: 0.9114, Precision: 0.9474
                psychology :           F1 score: 0.9422, Recall: 0.9588, Precision: 0.9261
                sociology :            F1 score: 0.9342, Recall: 0.9342, Precision: 0.9342
        Metrics for Fold 2
                political science :     F1 score: 0.9367, Recall: 0.9427, Precision: 0.9308
                psychology :           F1 score: 0.9317, Recall: 0.9259, Precision: 0.9375
                sociology :            F1 score: 0.9379, Recall: 0.9379, Precision: 0.9379
        Metrics for Fold 3
                political science :     F1 score: 0.9245, Recall: 0.9130, Precision: 0.9363
                psychology :           F1 score: 0.9437, Recall: 0.9618, Precision: 0.9264
                sociology :            F1 score: 0.9317, Recall: 0.9259, Precision: 0.9375
        Metrics for Fold 4
                political science :     F1 score: 0.9154, Recall: 0.9012, Precision: 0.9299
                psychology :           F1 score: 0.9461, Recall: 0.9405, Precision: 0.9518
                sociology :            F1 score: 0.9121, Recall: 0.9333, Precision: 0.8917
        Metrics for Fold 5
                political science :     F1 score: 0.8982, Recall: 0.9259, Precision: 0.8721
                psychology :           F1 score: 0.9384, Recall: 0.9580, Precision: 0.9195
                sociology :            F1 score: 0.9102, Recall: 0.8686, Precision: 0.9560
```

With the hyperparameters of the model settled, we can look at the performance in detail. Figure 4.1 is a detailed report of one training result that, on average, has 92.88% accuracy predicting the correct classes.

Figure 4.2, Confusion Matrix for Fold 1




Investigating the model with Figure 4.2, confusion matrix for fold 1, the recall score for psychology is particularly high. Among 5 folds, psychology holds the highest recall score in 4 folds, sometimes with great margin. However, the same cannot be said for psychology precision scores. Although the scores are generally high but does not outperform the other classes by much. General performance regarding psychology has overwhelming contributions from recall scores and stable precision scores, F1 scores for

psychology are mostly the highest out of the three classes. By having high recall scores, the model is more accurate when it is predicting the label of data that is truly psychology abstracts.

Figure 4.3 Confusion Matrix of Fold 2

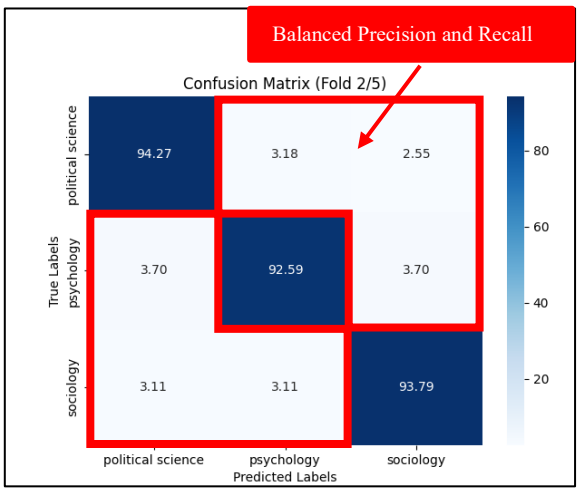



Figure 4.4. ROC Curves of Fold 2

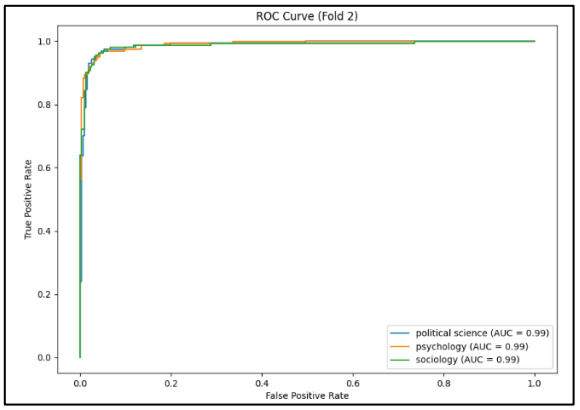
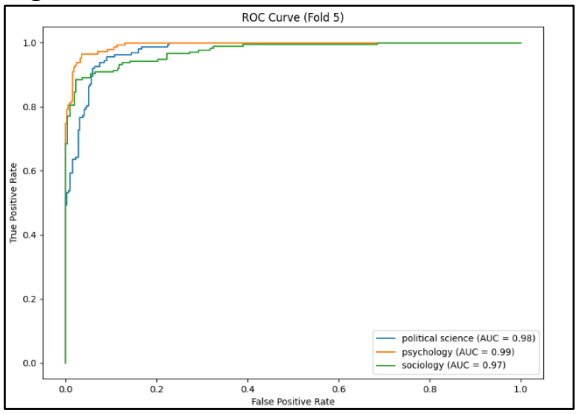



Figure 4.5. ROC Curves of Fold 5




Shifting focus to sociology. Precision score is identical to recall score for fold 1 and 2. Although there are imbalances between recall and precision score, it is not by a large margin. This is also evident from Figure 4.3 showing balanced false predictions amongst the classes. Coincidentally, fold 1 and 2 are the two most accurate models. Potential connection of balanced precision and recall score to a high performant model can be drawn here, inciting potential investigations.

Looking at the ROC curve of fold 2 tells a similar story. The AUC value, or area under the curve value, for all three classes are above 0.99. However, just looking at the better performing fold defeats the purpose of implementing K-fold cross validation. We should pay attention to low performing folds as well. The ROC curve for fold 5, the least performing fold, can clearly show use about the tradeoffs between precision and recall for these three classes. Psychology has good balance between the two, but the model prioritized precision rather than recall for sociology and reversed for political science. Even with these tradeoffs sociology and political science still achieved 0.97 and 0.98 AUC value respectively. Overall, the result from optimizing hyperparameters is satisfactory, increasing from 87% on average before investigation, to 92.8%. Diving into the performance metrics shows acceptable stability.

## V. Conclusion

This report was set to explore multiclass text classification where there is strong emphasis on experimentation and interpreting the results. First, data acquisition and preprocessing were discussed to lay out the foundation, the dataset for exploration. Second, the techniques that are implemented in the experiments were established. Techniques include K-Fold cross validation, TFIDF, one hot encoding, and early stopping. Although there were some heuristic choices, the process of the exploration was mostly experiments. Particularly, this report investigated hyperparameters of a FNN; feature length, network structure, dropout rate, and learning rate. Despite the explorations, there are multiple aspects of a FNN that wasn't discussed, activation function and batch size, for example. Lastly, as the optimal configuration is established through the experiments, this report discussed the strengths and weaknesses with performance metrics of the final model concluded the effectiveness of the model.