ARTIFICIAL INTELLIGENCE FINAL ASSIGNMENT

REPORT: SCIENTIFIC ABSTRACT CLASSIFICATION

Andifallih Noor Malela – 26002304448

31889: Artificial Intelligence (G1)

INTRODUCTION

Text classification is a common task in the field of machine learning (ML) and natural language processing (NLP), a subset of ML focused on understanding language. It usually consists of preprocessing input dataset with techniques such as stemming, stop-word removal, and tokenization, followed by applying machine learning or deep learning algorithms such as random forests, support vector machines (SVM), or performing transfer learning with pre-trained language models like BERT (Bidirectional Encoder Representations from Transformers).

In this report, the author built a classifier that's able to correctly categorize scientific abstracts of one of three predefined scientific fields using BERT-base model. The three scientific fields chosen are political science, sociology, and psychology.

BACKGROUND

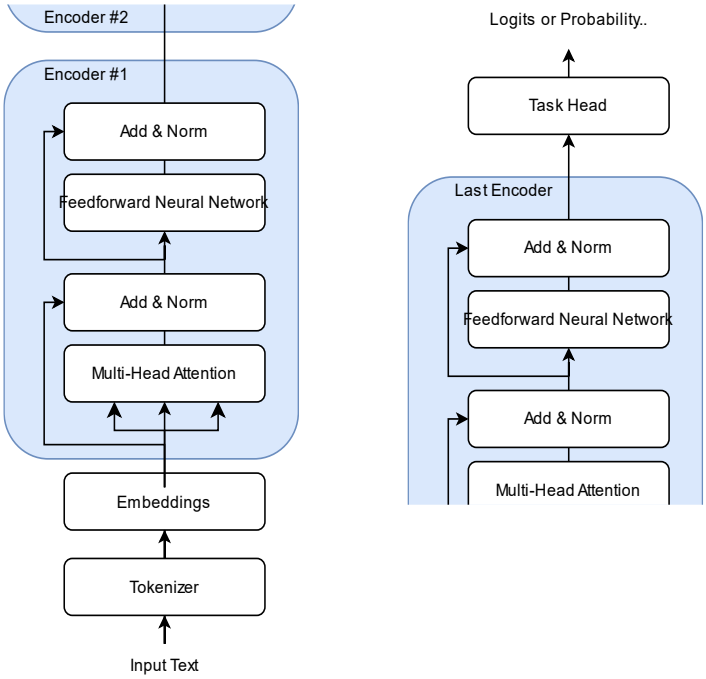
BERT (Bidirectional Encoder Representations from Transformers) is a language model pre-trained to understand the English language and can be fine-tuned for a variety of tasks, including text classification.

- *Bidirectional*: BERT simultaneously processes texts from both left and right directions of a word to consider the full "bidirectional" context of a word to its surrounding texts.
- *Encoder*: The encoder component in a Transformer architecture which BERT relies on.
- *Representations*: BERT generates representations that capture contextual relationships and meanings of words.
- *Transformers*: The deep learning architecture consisting of encoders and decoders. However, BERT only uses the encoder portion within its architecture.

Like other neural networks, BERT consists of connected layers of neurons that process and transmit numerical information through multiple layers, with various parameters associated with the neurons.

During forward pass of the training, a representation of the difference between the model's predicted output and the actual correct output in the input data is calculated by a loss function. This loss is then minimized by backpropagating the error through the network in a backward pass, adjusting the weights and biases and other parameters, allowing the model to learn and perform better on the input data for its specified task over trainings.

What makes BERT different is the architecture. From a high-level, the pipeline of BERT consists of 4 components: Tokenizer, Embedding, Encoder, and Task head. Diagram below shows their structure.



1. Tokenizer

Preprocesses English words into format BERT can understand. Breaks input sentences into sub-words "tokens" which are mapped to a unique ID based on the tokenizer's vocabulary. It also adds special tokens [CLS] for classification purposes and [SEP] for sequence separator. BERT-base's tokenizer, WordPiece, handles tokenization without requiring much data preprocessing.

- Text: "I am training NLP"
- Tokens: "[CLS], i, am, train, ##ing, nlp, [SEP]" – *lowercases if the tokenizer used is uncased*
- Token IDs: "[101, 1045, 2572, 4735, 2075, 17953, 102]"

Tokenizer also truncates sequences that are too long and adds padding [PAD] tokens for sequences that are too short, making all input equal to a specified length (e.g., 512). Real tokens are marked with 1 and padding tokens with 0 (attention masking). Only real tokens will be focused by the model in computation.

2. Embedding

Embeddings are dense vectors (vectors of mostly non-zero, real-numbered values) with the goal to represent higher dimensions of textual information from the tokenized input data. Each token is converted to embeddings consisting of three types: token, segment, position.

Token Embedding

The token IDs access a dense vector in an embedding matrix: an array of dense vectors where each vector represents a unique token in the vocabulary. WordPiece has vocabulary size of 30000 and embedding size of 768. So, the token embedding matrix size is 30000×768 with each token embedding being a 768-dimensional dense vector.

- [CLS] token with ID [102] accesses vector in row 102 of the matrix, which becomes its token embedding $E_{[CLS]}$.

Segment Embedding

Segment embeddings differentiate tokens in multiple-input sequences. Tokens in the same segment share the same segment embedding. There are two segment embeddings in BERT, E_A and E_B making a 2×768 segment embedding matrix.

- Tokens: "[CLS], how, are, you, ?, [SEP], good, [sep]"
- Segment embedding: "$E_A, E_A, E_A, E_A, E_A, E_A, E_B, E_B$"

This however is only for tasks with multiple-input sequences such as question-answering and next-sentence prediction (notice the middle [SEP] added to indicate differing segments). Otherwise, for single-input tasks such as text classification, all tokens belong to the same segment.

- Text: "I am training NLP. It is fun"
- Tokens: "[CLS], i, am, train, ##ing, nlp, ., it, is, fun, [SEP]"
- Segment embedding: "$E_A, E_A, E_A, E_A, E_A, E_A, E_A, E_A, E_A, E_A, E_A$"

Position Embedding

For understanding token position in their sequence. Each token is assigned an embedding from a position embedding matrix according to its sequence index. The maximum input sequence length in BERT-base is 512. Therefore, the max size of the position embedding matrix is 512×768.

- Tokens: "[CLS], i, am, train, ##ing, nlp, [SEP]"
- Position embedding: "$E_0, E_1, E_2, E_3, E_4, E_5, E_6, E_7$"

When training the model, values in all the embedding matrices and corresponding embeddings are updated though backpropagation as the model understand contextual information within input data.

Finally, an input embedding for each token is computed:

$$Input\ embedding\ =\ Token\ embedding\ +\ Segment\ embedding\ +\ Position\ embedding$$

The full sequence of input embeddings is a X matrix with size $n \times d$. n is sequence length and d is embedding size. This matrix is then processed through the encoder layers.

3. Encoder

Encoder layers create contextual representations for every token in the input sequence. Each encoder consists of two major components: a multi-head attention mechanism and a feedforward network (FFN) each followed by an add & norm module.

Multi-Head Attention

Self-attention mechanism determines relative importance of each token to every other token in the sequence, allowing entire context surrounding each word be fully considered giving a bidirectionality aspect to the model. Because in natural language, word meanings depend on other words. A single attention mechanism might only focus on one type of relationship between tokens. So, multi-head attention aims to capture multiple different types of relationships and features by splitting the self-attention mechanisms to multiple attention heads (12 heads in BERT-base) and each head running parallelly. Here's how it works:

Queries Q, keys $K$and values V matrices are created from input embedding X:

$$Q = XW^Q$$
$$K = XW^K$$
$$V = XW^V$$

W^Q, W^K, W^V are weight matrices with values tuned during training.

For *each head*, scaled-dot product attention calculates how much each token attends to all other token in the sequence:

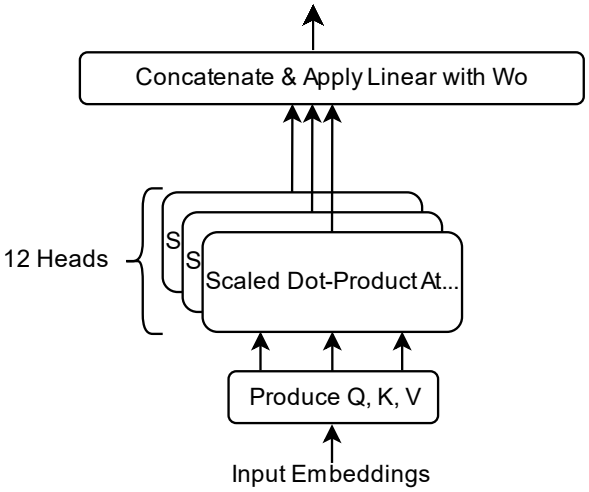
$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_h}}\right)V$$

$$d_h = \frac{d}{h} = \frac{embedding\ size}{number\ of\ attention\ heads}$$

Then the multi-head output for all h number of heads is calculated with:

$$MultiHeadOutput = Concat(head_1, head_2, \dots, head_h)W^O$$

W^O is projection matrix learned during training.



The concatenation combines different types of relationship representations from each head, while the W^O acts like an editor which learns how to combine them meaningfully as the model is trained for a certain task.

Add & Norm

There is add & norm module after each multi-head attention and FFN layers. Here are the functions:

- Add: Preserves original information in input data, directly combining them to the new learned transformation (after the multi-head attention and FFN layers).
- Norm: Normalizes output of transformations, ensuring values remain stable, preventing exploding or vanishing gradients during training.

After multi-head attention:

$$Output\ =\ LayerNorm(Input + MultiHeadOutput)$$

After FFN:

$$Output\ =\ LayerNorm(Input + FNN\ Output)$$

FFN (Feedforward Network)

Multi-head attention gives linear context to each token representations based only off relationships captured by attention mechanism. FFN then introduces non-linearity using activation functions allowing the model to learn even more complex patterns and refine the token representations.

There are two layers in each FFN module. First layer expands dimensionality of embeddings enabling it to learn more complex relations and patterns. The second layer reduces the dimensionality back to its original size retaining while its learned information. Here's the formula:

$$FFN\ Output = \varphi(XW_1 + b_1)W_2 + b_2$$

X is a normalized multi-head attention output matrix and φ is activation function.

Then it goes through add & norm module again before going to another encoder layer or a task head.

BERT-base contains 12 encoder layers. Each layer refines token representation capturing more complex relationships between tokens.

4. Task Head

BERT-base contains 12 encoder layers. Each layer refines token representation capturing more complex relationships between tokens. After the last encoder, the output is a matrix consisting of contextualized embeddings of tokens. This is then processed to a task head that are specific to the model's task positioned after the last encoder layer.

BERT-base was pre-trained on Masked language modeling (MLM) task and next sentence prediction (NSP) task.

- During MLM task, the task head is an un-embedding layer. Random tokens will be replaced with [MASK] token. The training objective is to predict these masked tokens by projecting embeddings back to vocabulary and generating predicted token probabilities.
- During NSP task, the task head is a binary classification layer. Given two sequences of input data, the model tries to predict if the second sequence logically follows the first.

Multi-class text classification, the fine-tuning goal of this report is like the NSP task. However, it uses a single sequenced input data.

In more details, it works by using the [CLS] token embedding, which aggregates information about the entire input sequence. The embedding is passed through a dropout layer, which randomly drops a certain number of neurons to induce generalization, ensuring the model can generalize to unseen data and prevents overfitting. Then it's passed through a fully connected layer network, a neural network where each input neurons is fully connected to every output neurons with learnable weights, and a softmax function that converts raw output scores aka logits to probability distribution over the classes. The softmax function can be replaced with an argmax function which selects largest possible values in the classes.

METHODOLOGY

1. Dataset Acquisition, Cleaning, Preprocessing

The abstracts were obtained by scraping scientific journals specific to a given field using CrossRef API, with the script written in Python. CrossRef API allows access to metadata of scholarly articles, including abstracts. The script makes a request to CrossRef API, filtering the results based on a specified journal that retrieves an abstract. It then saves them to a CSV file each with a label corresponding to the field of the specified journal (e.g., "sociology"). The scraper performs this for each field. Here is the list of journals used:

- Political Science: Annual Review of Political Science, British Journal of Political Science.

- Psychology: Annual Review of Clinical Psychology, Psychological Science.
- Sociology: Sociology, Annual Review of Sociology, American Journal of Cultural Sociology.

After scraping, to make sure that the datasets only contain texts relevant to the abstract cleaning involved removing:

- Duplicates.
- Unicode characters.
- Empty entries like "Not available" and blank strings.
- HTML artifacts.
- Whitespaces and line breaks.
- Text artifacts such as "Abstract:" and "Conclusion:" to prevent interference with training – certain journals returned this text in the abstracts while others did not, the model might learn to associate on these terms instead of focusing on the abstract content.

One thing to note was that the author performed data acquisition before deciding on which models to use for text classification. Therefore, it included steps that are unnecessary when using BERT tokenizer such as removing Unicode characters and whitespaces.

The resulting dataset is a collection of 900 total abstracts from three scientific fields with each field having 300 abstracts.

2. Learning Tools

PyTorch was used as the foundational machine learning framework in training and fine-tuning BERT.

K-fold cross-validation technique was employed during training to evaluate the model's performance on different subsets of the data. How it works is that there are k number of folds, and the dataset was divided to k number of splits. The model trains on k-1 splits with the remaining split used for validation, and the splits were rotated for each fold. This ensured that every datapoint is used for both training and validation. The folds were stratified using library to ensure that proportion of the class labels in each split dataset is consistent with their proportion in the original dataset. $1/3^{rd}$ of the data being psychology labels, $1/3^{rd}$ being sociology labels, and $1/3^{rd}$ being political science labels. To do these tasks, StratifiedKFold library was used.

Early stopping with a patience counter was implemented using PyTorch's built-in functionalities to stop the training process whenever the validation loss stagnated or increased. The goal was to allow the model to learn from the training data in each fold as much as possible while preventing unnecessary training when it has reached its optimal state (when validation loss stopped decreasing) and avoiding overfitting (when validation began to rise).

ReduceLROnPlateau library was used to adjust the learning rated during training based on the performance of validation loss metric. If the loss does not improve after a certain number of epochs, learning rate is reduced by a certain factor.

AdamW (Adaptive Moment Estimation with Weight decay) was used to optimize the model's parameters during training. It combines gradient decent with momentum and RMSProp (Root Mean Square propagation) algorithms to descent the loss landscape. Gradient descent with momentum removes sudden changes in parameter values, smoothing it and fastens training, while RMSP adapts the learning rate for each parameter based on previous gradients. Additionally, weight decay is applied as regularization technique to improve generalization performance.

Fixed seed was also implemented to ensure reproducibility of random data shuffling, data splitting, and numerical operations.

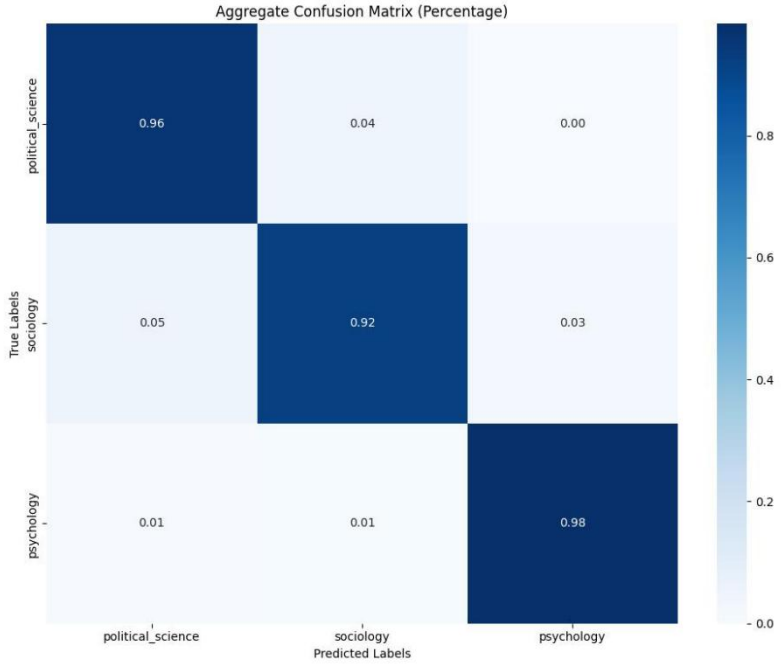
EXPERIMENTATION & RESULTS

Initial hyperparameters were initially set as follows:

- **Batch size: 16** – for reasonable balance between computational efficiency and convergence speed.
- **K-folds: 5 folds** – this gives 900/5=180 total samples per fold for validation, with 180/3=60 samples per class label for the model to predict on.
- **Early stopping patience: 3 epochs** – model will stop after 3 epochs of no visible loss improvement.
- **Sequence length: 512 (max)** – scientific abstracts are long texts therefore the maximum length is used.

Here is result of initial run with uncased version of BERT-base. Uncased meaning it ignores uppercase letters.

Confusion Matrix 1. BERT-base-uncased



Average Metrics Across All Folds:

Accuracy: 0.9533, Precision: 0.9540, Recall: 0.9533, F1: 0.9532

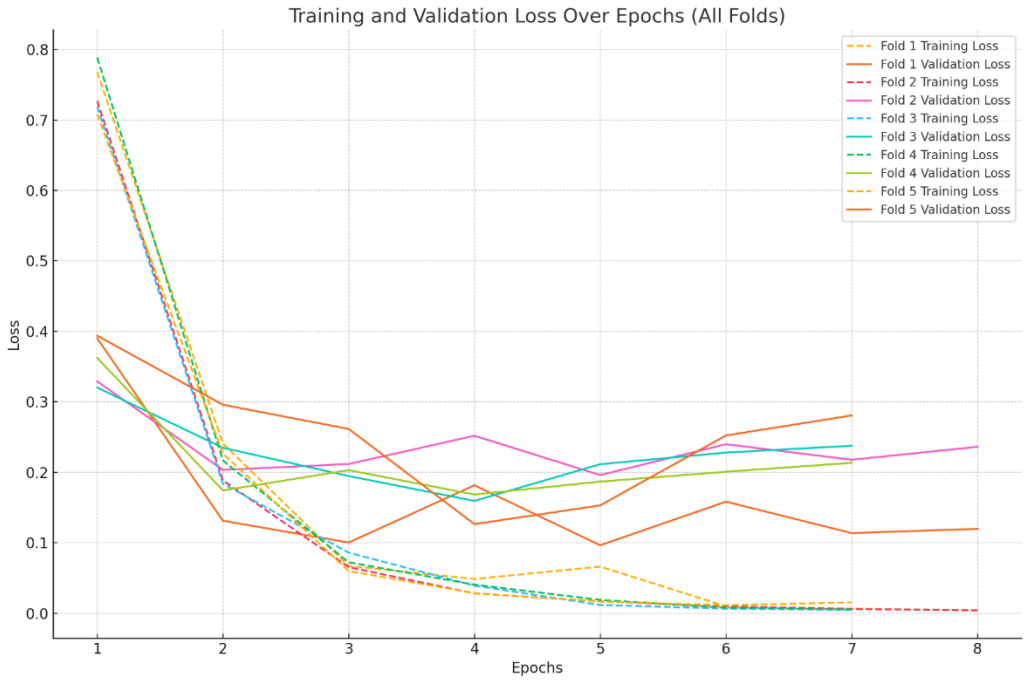
It reached an average accuracy of 95.33%. This is the percentage of all samples were identified correctly across all folds. 95.40% precision means that it has a low rate of false positives across all the classes prediction. 95.33% recall value means that the model has a low rate of false negatives (captures mostly true predictions). 95.32% F1 score means that the model performs with a good balance between precision and recall.

98% of psychology samples were correctly classified, 96% of political science samples were correctly classified, and 92% of sociology samples were correctly classified.

Sociology label has the highest incorrect prediction, with 5% being misclassified as political science, and 3% as psychology. Probably due to overlap in the contextual features of the abstracts.

Here is deeper look at the loss values to understand what's going on during training and validation:

Graph 1. Validation and Training Loss over Epochs (All Folds) BERT-base-uncased



Validation loss is seen decreasing along with training loss, but after around Epoch 3 & 4, validation loss stopped decreasing fluctuated.

Below is a cleaner look of it happening in Fold 2:

Graph 2. Validation and Training Loss over Epochs (Fold 2)

This means that the model is learning well on its training data but stops doing so well on validation after some Epochs. Combined with the quite significant gap between training loss and validation loss, this suggests that the model is starting to overfit starting in Epoch 3. Where the model starts to memorize the training data rather than generalizing to the validation set.

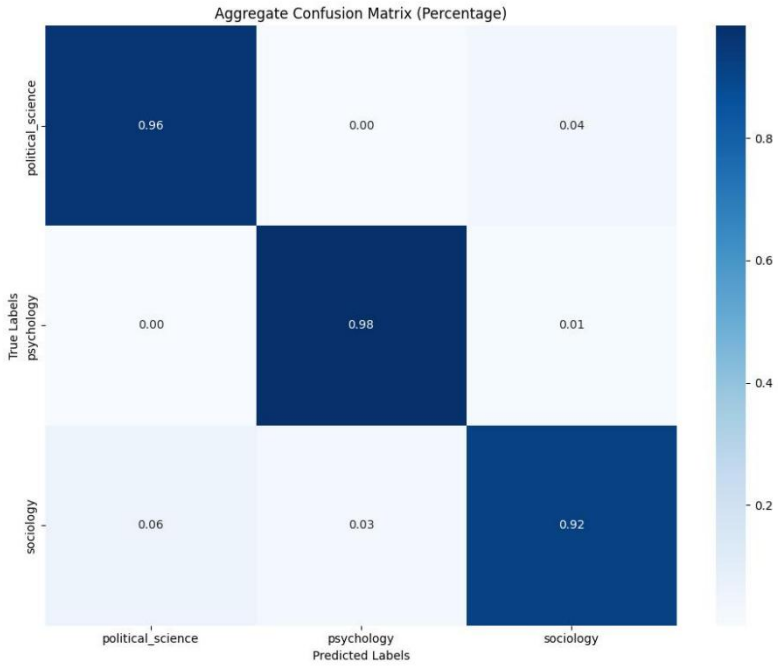
But how does the validation loss trend suggest overfitting even though other metrics (accuracy / precision / recall / F1) showed high values? Probably since validation loss is more sensitive than metrics used. Loss function captures subtle differences in model's prediction confidence while the metrics discretely treat

predictions as correct or incorrect. Say a model predicts a sample class correctly with 90% confidence, but later predicts the same class with 60% confidence. This drop of confidence increases the loss even though its prediction is correct and doesn't affect the metric values.

Overall, high metrics are a good sign. It means mean that the model has learned contextual patterns between the classes effectively.

Next, comparison attempt between cased and uncased BERT-base model was done to see how case sensitive the abstracts were. Note that label orders are different in the confusion matrix.

Confusion Matrix 2. BERT-base-cased



Average Metrics Across All Folds:

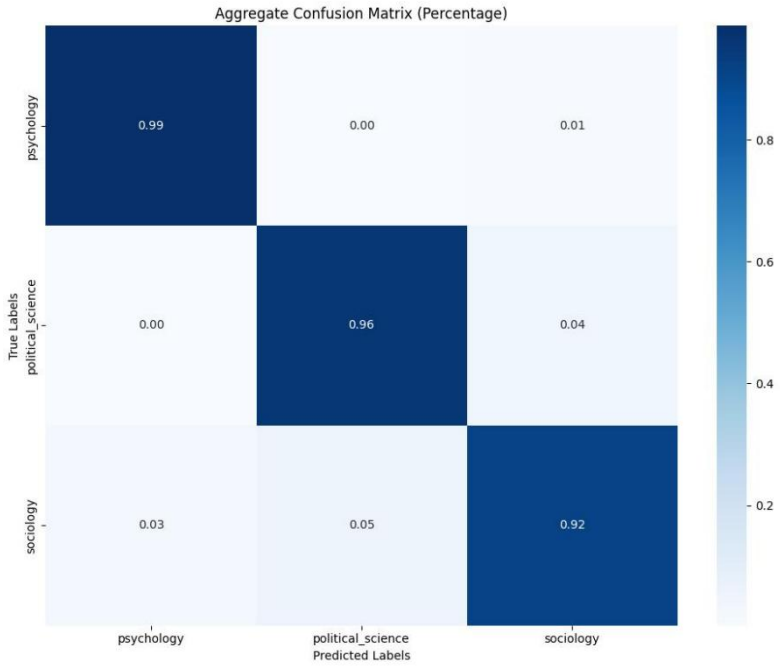
Accuracy: 0.9544, Precision: 0.9551, Recall: 0.9544, F1: 0.9543

BERT-base-cased metrics is slightly higher but there is no big difference between the two here and the confusion matrix result is the same. These are probably because vocabulary in scientific literatures don't rely on distinctions of letter cases to differentiate meanings and context too much, but still provide some contexts here and there either way. For example, when uppercased scientific acronyms pops up.

Going forward, cased version is used since the metrics are a bit higher.

From **Graph 1.** Learning is stopped at Epoch 7 or 8. A higher early stopping patience was implemented to see if the model could learn more about the data from more Epochs.

Confusion Matrix 3. BERT-base-cased, early stopping patience=6



Average Metrics Across All Folds:

Accuracy: 0.9533, Precision: 0.9539, Recall: 0.9533, F1: 0.9531

At patience=6, learning stopped at around 2 or 3 epochs later than at patience=3. Metrics are identical but slightly lower, and there is an odd 1% increase in the confusion matrix for psychology labels.

This difference is perhaps since the metrics aggregates overall performance across all classes whereas confusion matrix shows raw counts in prediction for each class. So although from the confusion matrix the it performed slightly better for psychology samples, there might be trade-offs on other classes suggested by the metrics that are unseen on the matrix.

A 1% improvement in predictions for a single class is also unlikely to be impactful for the model generalized performance at text classification. Especially since all three classes here have equal importance and the dataset are balanced.

Proceeding experiments will stick to lower patience to enhance speed of training since higher patience did not yield significant performance increase and took more time for the model to train.

A lower patience (patience=1) is also attempted but yielded a small, but worse results. With metrics below:

Accuracy: 0.9400, Precision: 0.9428, Recall: 0.9400, F1: 0.9397

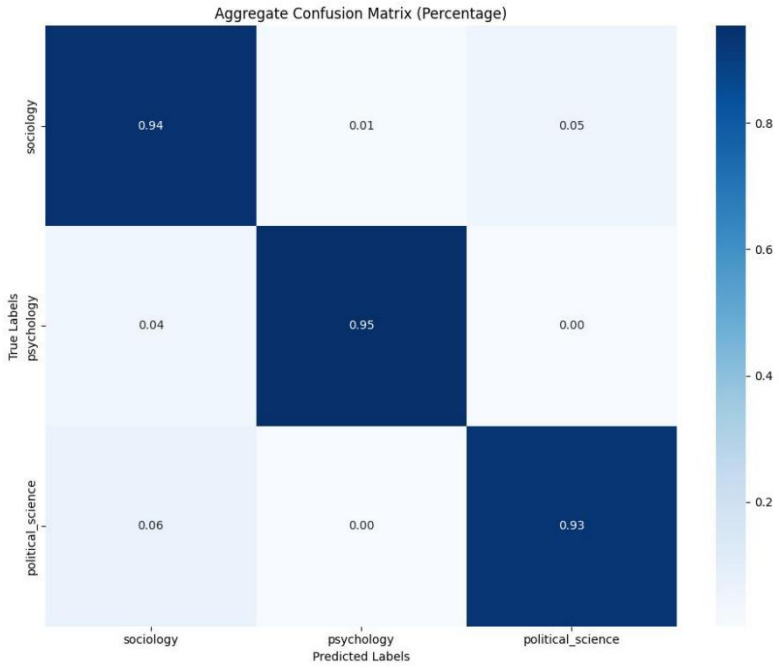
This is because when the patience counter is only 1, training will stop as soon as validation loss doesn't improve in a single epoch. This halts training before the model fully learned patterns in the data, leading to suboptimal performance.

Seeing through all the confusion matrices so far, correct sociology labels prediction percentage are still lagging (~92%). The next experiment aimed to solve this by assigning higher importance to the sociology class.

This is done by adding a generated class weights to the loss function calculation. Errors made on sociology predictions will contribute more to the overall loss during training. This change hopefully ensures that the model pay more attention to textual patterns that are specific to sociology samples.

Here is the result:

Confusion Matrix 4. BERT-base-cased, early stopping patience=3, weighted class implementation



Average Metrics Across All Folds:

Accuracy: 0.9422, Precision: 0.9456, Recall: 0.9422, F1: 0.9425

This model was trained with class weighting of {Political Science: 1.0, Sociology: 1.5, Psychology: 1.00}. From the confusion matrix, sociology class prediction accuracy is seen to be improved (2% correct prediction increase) but other class predictions took a hit and so did overall metrics.

Here's the comparison between the performance of this model and the model plotted in Confusion Matrix 2. Both models are BERT-cased with patience=3.

Metric/Class	Weighted	Unweighted	Difference
Sociology Sample Accuracy	94%	92%	**+2% (weighted better)**
Psychology Sample Accuracy	95%	98%	**-3% (unweighted better)**
Political Science Sample Accuracy	93%	96%	**-3% (unweighted better)**
Accuracy	94.22%	95.44%	**-1.22% (unweighted better)**
Precision	94.56%	95.51%	**-0.95% (unweighted better)**

Recall	94.22%	95.44%	**-1.22% (unweighted better)**
F1-Score	94.25%	95.43%	**-1.18% (unweighted better)**

The table visualizes that though the weighted model prioritizes sociology at the expense of other classes, it performs worse overall compared to the unweighted model.

Author concludes that **unweighted model is overall better** for this task of multiclass text classification**.** It still attained higher metrics of accuracy, precision, recall, and F1. It also generalized more and had better performance in predicting the other two classes.

CONCLUSIONS

In the task of building a classifier that classifies scientific abstracts to their specified field of sociology, political science and psychology, a path of using a pre-trained language model BERT was taken. Author performed transfer learning for the task, fine-tuning the model for classifying texts.

The overall objective has been achieved with suitably good results. The best model was able to predict the labels an accuracy of 95.44%, precision of 95.51%, recall of 95.44%, and F1-score of 95.43%. The confusion matrix analysis revealed that the model maintained high class-specific accuracies, with 92% for sociology, 98% for psychology, and 96% for political science.

The saved model could theoretically be used to classify unseen abstracts that belong the three scientific fields.

(3470 words everything according to MS Word)