

Investigating the Use of Statistical Tests in Natural Language Processing Machine Learning Models

Jule Valendo Halim -1425567
University of Melbourne

Abstract

Objective

To investigate possible roles of statistical tests on natural language processing tasks through models for multi-label and multi-class classifiers.

Materials

Dataset containing online patient reviews of prescribed medication, provided by Zoolnori et al^{1,2}. The dataset contains annotated labels for each review sentence, along with the medication being reviewed.

Methods

Various multi-label and multi-class classification tasks were created. Afterwards, each model was evaluated using traditional machine learning metrics as well as statistical tests.

Results

When comparing traditional machine learning metrics, transformer models generally outperform their logistic regression, except for one task-feature pair. Statistical test results were reported and interpretations were provided.

Discussion

Interpretations of statistical tests were discussed in the context of model behaviour. Possible limitations and issues were also described.

Conclusion

Statistical tests provide valuable insight into model behaviour outside of traditional machine learning metrics. However, additional work is needed in creating a pipeline to investigate causes of possible model behaviour, as well as integration of assumption testing.

Introduction

Statistical tests are a popular and long-standing method in multiple fields of science. However, studies in natural language processing (NLP) and machine learning (ML) do not generally include statistical testing as part of their model evaluation. A survey on 233 published papers in the field of NLP showed that 132 of these papers did not report statistical significance³. However, more studies have begun advocating for the use of these tests to show that experimental results are not coincidental³ and argue that a combination of NLP and statistical tests can provide a framework for the development of robust, high-throughput health NLP systems⁴.

In this report, I aim to investigate the use of statistical tests on ML models that predict multi-label and multi-class classification tasks using the statistical test workflow suggested by Rainio, Tauho, and Klén⁵, shown in figure 1, and compare different feature inputs.

Task of the models

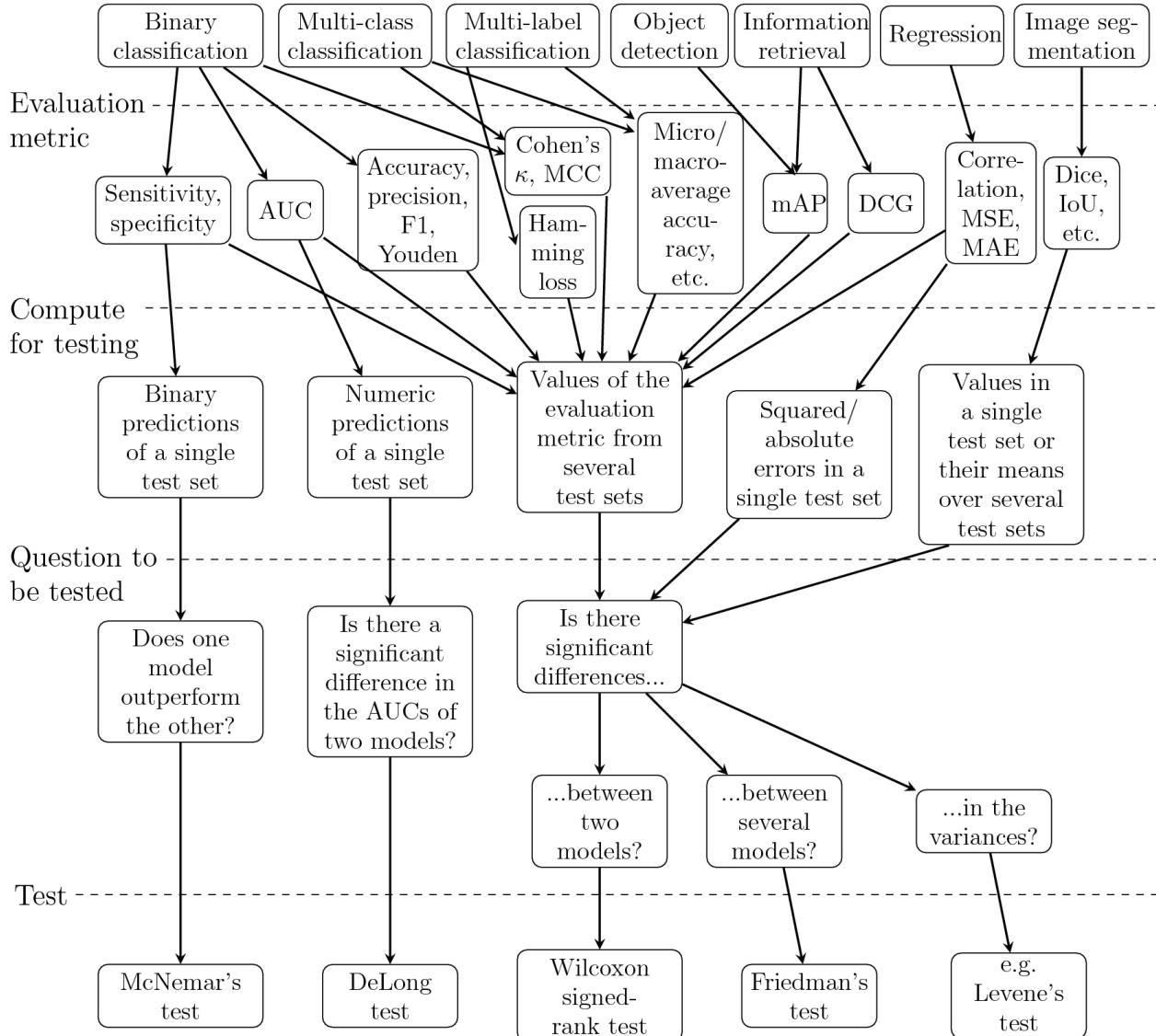


Figure 1. Statistical Workflow Provided by Rainio, Tauho, and Klén⁵

The Sentence_Labeling sheet of the PsyTAR dataset used in this study is provided by Zoolnoori et al¹². This sheet contains three sections of interest. The first is sentences from online patient reviews of certain drugs. The second is annotations of certain labels, described below in table 2. The third is drug labels, which indicates which drug each sentence is reviewing. Table 1 shows descriptive statistics of the dataset.

Two classification tasks were performed using a baseline logistic regression model and a transformer model. The multi label class predicts six binary annotations, while the multi-class predicts four different drugs. For each model, different features will be used as inputs. The resulting predictions will be used for statistical tests to determine their statistical significance.

Number of Sentences	6009
Sample Review Text	extreme weight gain, short-term memory loss, hair loss.
Drug Class Distribution	
Cymbalta	1707
Lexapro	1492
Effexorxr	1549
Zoloft	1261
Review Text Label (Positive)	
ADR (Positive)	2169
WD (Positive)	439
EF (Positive)	1088
INF (Positive)	338
SSI (Positive)	790
DI (Positive)	622
Review Text Label (Negative)	
ADR (Negative)	3840
WD (Negative)	5570
EF (Negative)	4921
INF (Negative)	5671
SSI (Negative)	5219
DI (Negative)	5387

Table 1. Descriptive Statistics from Sentence Labeling Dataset - Description of Review Text Labels Are Found in Table 2.

Methods

Preprocessing

Two main preprocessing steps were done. Firstly, as the drug labels in the data were concatenated with review ID, the review ID was stripped and only the drug label was added to a new column. Secondly, the review text was preprocessed through tokenization, as well as stopword and punctuation removal. The data was split into train, validation, and test sets using a 45:45:10 ratio. Additional feature processing specific to each task is described in their respective sections.

Multi-Label Classification

Multi-label classification involves predicting six labels, as shown in table 2. Each label is a binary task (1 for present and 0 for absent), which has been manually annotated. This task aims to predict annotations for a given review text.

Predicted Class	Description
Adverse Drug Reactions (ADR)	Adverse reactions to the drug
Withdrawal Symptom (WD)	Withdrawal symptoms after they stopped using the drug
Effective (EF)	Drug is effective
Ineffective (INF)	Drug is ineffective
Sign/Symptom/Illness (SSI)	Text contains explicit SSI as a result of the drug
Drug Indication (DI)	Text contains SSI that is currently being addressed by the drug

Table 2. The Six Labels Predicted by Multi-Label Classifiers

Selected Features Two features were used for multi-label classification. The first is the preprocessed review texts without any additional features. From here on, references to this task-feature pair will be referred as

task 1 feature 1. The second contains the drug name added to the start of the preprocessed review text. This will be referred to as task 1 feature 2.

Feature 1	Feature 2
extreme weight gain short-term memory hair loss	lexapro extreme weight gain short-term memory hair loss

Table 3. Sample Inputs for Each Feature in Multi-Label Classification

Logistic Regression The logistic regression model performed Term Frequency Inverse Document Frequency (TF-IDF) vectorization on the input text. A multi-output classifier was built upon a logistic regression model, which was then tuned using hyperparameter tuning.

Feature 1: Best Parameters Found	Value	Feature 2: Best Parameters Found	Value
estimator_C	10	estimator_C	10
estimator_penalty	l2	estimator_penalty	l2
estimator_solver	saga	estimator_solver	lbfgs

Table 4. Best Parameters for Logistic Regression (Feature 1 and Feature 2)

Transformer The transformer model uses a pre-trained BERT model, which was trained on a downstream task in order to create a multi-label transformer classifier. Binary cross-entropy loss (BCE) with logits (a sigmoid layer) loss was used for loss calculation. BCE is well-suited for binary classification⁶.

Parameter	Value
hidden_size	768
num_hidden_layers	24
num_attention_heads	12
intermediate_size	3072
num_labels	6
optimizer	AdamW
loss_calculation	BCE With Logits Loss
epochs	15

Table 5. BERT Configuration for Multi-Label Classification

Statistical Tests Multi-label classification will be tested using macro averaged precision, recall, and F1-scores. The choice to use macro instead of micro was to give equal weights to each class, as some labels were more prevalent than others. Accuracy will also be reported.

The Hamming Loss (HL) of the predictions are also calculated. HL measures the fraction of labels that are incorrectly predicted, on average, across all samples and is used to evaluate multi-label classification tasks⁷. HL ranges from 0 to 1, where 1 means all predictions are erroneous, while 0 means perfect predictions. Appendix A contains statistical test equations.

Multi-Class Classification

Multiclass classification aims to take in varying inputs and predict the drug that is being reviewed. There are four classes of drugs to predict; lexapro, cymbalta, effexorxr, and zoloft.

Selected Features Three features were selected for multi-class classification. The first is the preprocessed sentence without any additional features, referred to as task 2 feature 1.

The second is the preprocessed text along with its annotations. For the logistic regression model, the binary annotations are simply added onto the end of the sentences as 1s and 0s.

The transformer's input has the review text concatenated with the predicted class names. If the class is labelled 1, a [POS] token was placed in front of it. If the class is labelled 0, a [NEG] token was placed instead. These tokens identify positive(1) and negative(0) labels respectively. This feature will be referred to as task 2 feature 2.

The third feature is to only use the annotation inputs. This will be referred to as task 2 feature 3. The logistic regression model takes in only the binary annotations, while the transformer only takes in the predicted class name along with the described tokens.

Feature 1 Transformer	Feature 2 Transformer	Feature 3 Transformer
extreme weight gain short-term memory hair loss	[POS] adverse drug reaction [NEG] withdrawal symptoms...extreme weight gain short-term memory hair loss	[POS] adverse drug reaction [NEG] withdrawal symptoms...[NEG] drug indication
Feature 1 Logistic Regression	Feature 2 Logistic Regression	Feature 3 Logistic Regression
extreme weight gain short-term memory hair loss	extreme weight gain short-term memory hair loss	1 0 1 1 0 0

Table 6. Sample Inputs for Each Feature in Multi-Class Classification

Logistic Regression TF-IDF vectorization was used on the input text. However, in contrast to using a multi-label classifier built on top of a logistic regression model, multi-class classification uses logistic regression directly.

Features 1-3 (Same Parameters Found): Best Parameters Found	Value
C	0.1
penalty	l1
solver	saga

Table 7. Best Parameters for Logistic Regression (Features 1-3)

Transformer The transformer model is identical to the multi-label transformer, except it was trained on a multi-class downstream task. The number of hidden layers was also decreased due to long training times. The loss calculation was also changed to cross-entropy as it is a better fit for classification tasks⁸.

Parameter	Value
hidden_size	768
num_hidden_layers	8
num_attention_heads	12
intermediate_size	3072
num_labels	1
optimizer	AdamW
loss_calculation	Cross-Entropy Loss
epochs	20

Table 8. BERT Configuration for Multi-Class Classification

Statistical Tests Multi-class classification will be tested using macro averaged precision, recall, and F1-scores as described previously. The use of macro averaging was due to class imbalance (zoloft had a test count of 565 while cymbalta had a test count of 791). Accuracy will also be reported.

Additionally, following figure 1, Cohen's Kappa⁹ (Cohen's K) and Matthews Correlation Coefficient (MCC) will be used. Cohen's K is used to calculate inter-model agreement on predictions. It returns a value between 1 and -1, where 1 means a perfect agreement, 0 means no agreement above chance, and -1 indicating less agreement than random chance.

Absolute Cohen's Kappa Range	Interpretation
$ \kappa \leq 0$	No agreement
$0.01 \leq \kappa \leq 0.20$	None to slight agreement
$0.21 \leq \kappa \leq 0.40$	Fair agreement
$0.41 \leq \kappa \leq 0.60$	Moderate agreement
$0.61 \leq \kappa \leq 0.80$	Substantial agreement
$0.81 \leq \kappa \leq 1.00$	Almost perfect agreement

Table 9. Cohen's Kappa Cutoff Points Based on McHugh's Research¹⁰

MCC has been adapted for multi-class classification by considering all the true and false positive as well as true and false negatives for each class¹¹. This adaptation of the MCC can indicate whether model performance across all classes.

MCC returns a value between 1 and -1, where 1 means perfect predictions, 0 means a prediction that is no better than random, and -1 means total disagreement between predictions and ground truth. MCC follows cutoff point selection of graphs such as an area under a receiver operating characteristic (AUROC) curve^{12 13}. However, this report will follow arbitrary cutoff points.

Absolute MCC Values	Interpretation
$0 \leq MCC \leq 0.1$	Very Poor Predictions
$0.1 < MCC < 0.3$	Poor Predictions
$0.3 \leq MCC < 0.5$	Moderate Predictions
$ MCC \geq 0.7$	Good Predictions

Table 10. MCC Cutoff Points

Friedman's Test of Significance

In order to investigate whether the predictions are significantly different from each other, Friedman's test will be attempted to be performed on both tasks. Friedman's test^{14 15} is a hypothesis testing method, where the null hypothesis is that there is no significant difference between two samples of predictions¹⁶. Meanwhile, the alternative hypothesis is that a significant difference does exist. A cutoff point (α value) of 0.05 will be used, meaning that if the p-value is <0.05 , the null hypothesis will be rejected.

This test was found to not be suitable for the multi-class classification task due to the output. Additional discussion on the impact of this will be discussed in the discussion and conclusion section. For multi-label classification, each label had the Friedman's test performed.

- **Null Hypothesis (H_0):** There is no significant difference between the models' predictions for the label.

- **Alternative Hypothesis (H_1):** There is a significant difference between the models' predictions for the label.
- $\alpha = 0.05$

Ethics Statement

One ethical consideration for this study is that interpretations of statistical test results should only be seen as possible recommendations and indicators of possible model behaviour. As such, additional testing and observations need to be done to confirm these interpretations.

Secondly, this study proposes some potential issues regarding the statistical workflow by Rainio, Tauho, and Klén⁵. However, this should not be seen as a direct criticism of their study. Rather, it serves to highlight possible difficulties in the interpretations of how to apply statistical tests to natural language processing models.

Finally, this study proposes changes to the way machine learning models are evaluated by the wider scientific community. However, these proposed changes require significantly more robust investigation into their advantages before being taken as the norm for evaluating machine learning tasks. Instead, this study serves as an investigation into current methods and pipelines suggested by other studies in the field of combining statistical tests with traditional machine learning metrics.

Results and Analysis

Multi-Label Statistical Tests

Accuracy, Precision, Recall, and F1-Scores Table 11 shows the accuracy along with the macro precision, recall, and F1 scores. Transformer models also show training and validation loss.

Model	Accuracy	Macro Precision	Macro Recall	Macro F1	Training Loss	Validation Loss
Task 1 Feature 1 Transformer	0.98	0.95	0.91	0.93	0.0134	0.0789
Task 1 Feature 1 Logistic Regression	0.5083	0.64	0.38	0.47	N/A	N/A
Task 1 Feature 2 Transformer	0.98	0.92	0.93	0.92	0.0115	0.0772
Task 1 Feature 2 Logistic Regression	0.5249	0.65	0.38	0.47	N/A	N/A

Table 11. Accuracy and Macro-Averaged Precision, Recall, F1-Scores, Training Loss, and Validation Loss for Task 1

Transformers show much higher performances compared to logistic regression for both features. Change in features does not appear to significantly affect model performance for all models, with the highest change being accuracy in logistic regression models (1.66% difference).

Hamming Loss The results of calculating HL for multi-label classification are shown in table 12. Each model was compared to the ground truth.

The transformer models have significantly lower HL than the logistic regression models. This is as expected, as a higher accuracy indicates a higher chance of correct predictions. However, HL can show a more detailed view on model performance. For example, feature 1 using logistic regression has an accuracy of 50.83%. This accuracy metric is strict as it only considers an instance as correct if all labels were correctly predicted. However, the HL for this model suggests that only about 11.16% of the predictions were incorrect, indicating that the model performs better when considering individual label predictions.

Model	Hamming Loss Against Ground Truth
Task 1 Feature 1 Transformer	0.016057586
Task 1 Feature 1 Logistic Regression	0.11517165
Task 1 Feature 2 Transformer	0.019379845
Task 1 Feature 2 Logistic Regression	0.109634551

Table 12. Hamming Loss for Multi-Label Task (Task 1)

Friedman’s Test of Significance Friedman’s test was done on all the models simultaneously. It returns two values, a test statistic and a p-value. Results are shown in table 13.

Label	Statistic	p-value
ADR	16.373	0.003
WD	10.667	0.031
EF	64.681	3.00×10^{-13}
INF	12.653	0.013
SSI	34.712	5.32×10^{-7}
DI	27.855	1.33×10^{-5}

Table 13. Friedman’s Test of Significance for Multi-Label Task (Task 1)

The results of Friedman’s test statistic shows the difference in predictions for all models. These differences are statistically significant, as each of the p-values are less than the determined cutoff point (0.05). As such, we reject the null hypothesis. These results provide evidence in favor of the alternative hypothesis that each model’s predictions are significantly different.

Multi-Class Statistical Tests

Accuracy, Precision, Recall, and F1-Scores Table 14 shows the accuracy along with the macro precision, recall, and F1 scores. Transformer models also show training and validation loss.

Model	Accuracy	Macro Precision	Macro Recall	Macro F1	Training Loss	Validation Loss
Task 2 Feature 1 Transformer	0.84	0.84	0.84	0.84	0.044	0.286
Task 2 Feature 1 Logistic Regression	0.3688	0.78	0.35	0.29	N/A	N/A
Task 2 Feature 2 Transformer	0.834	0.84	0.84	0.83	0.031	0.289
Task 2 Feature 2 Logistic Regression	0.3688	0.78	0.35	0.29	N/A	N/A
Task 2 Feature 3 Transformer	0.322	0.33	0.31	0.26	0.696	0.696
Task 2 Feature 3 Logistic Regression	0.3422	0.28	0.31	0.24	N/A	N/A

Table 14. Accuracy and Macro-Averaged Precision, Recall, F1-Scores, Training Loss, and Validation Loss for Task 2

The resulting predictions of the model for multi-class classification generally show a high accuracy for transformers, compared to their logistic regression baselines. However, feature 3 showed a different trend, where the logistic regression baseline performed better. There was also a large change in accuracy compared to other transformer models (with feature 3 having roughly 50% lower accuracy than other transformer models). This could be due to how transformers require rich language information, which feature 3 does not provide, as the inputs are only a series of token-word pairs.

Cohen's Kappa A heatmap of Cohen's K for each model tested against each other is shown in figure 2.

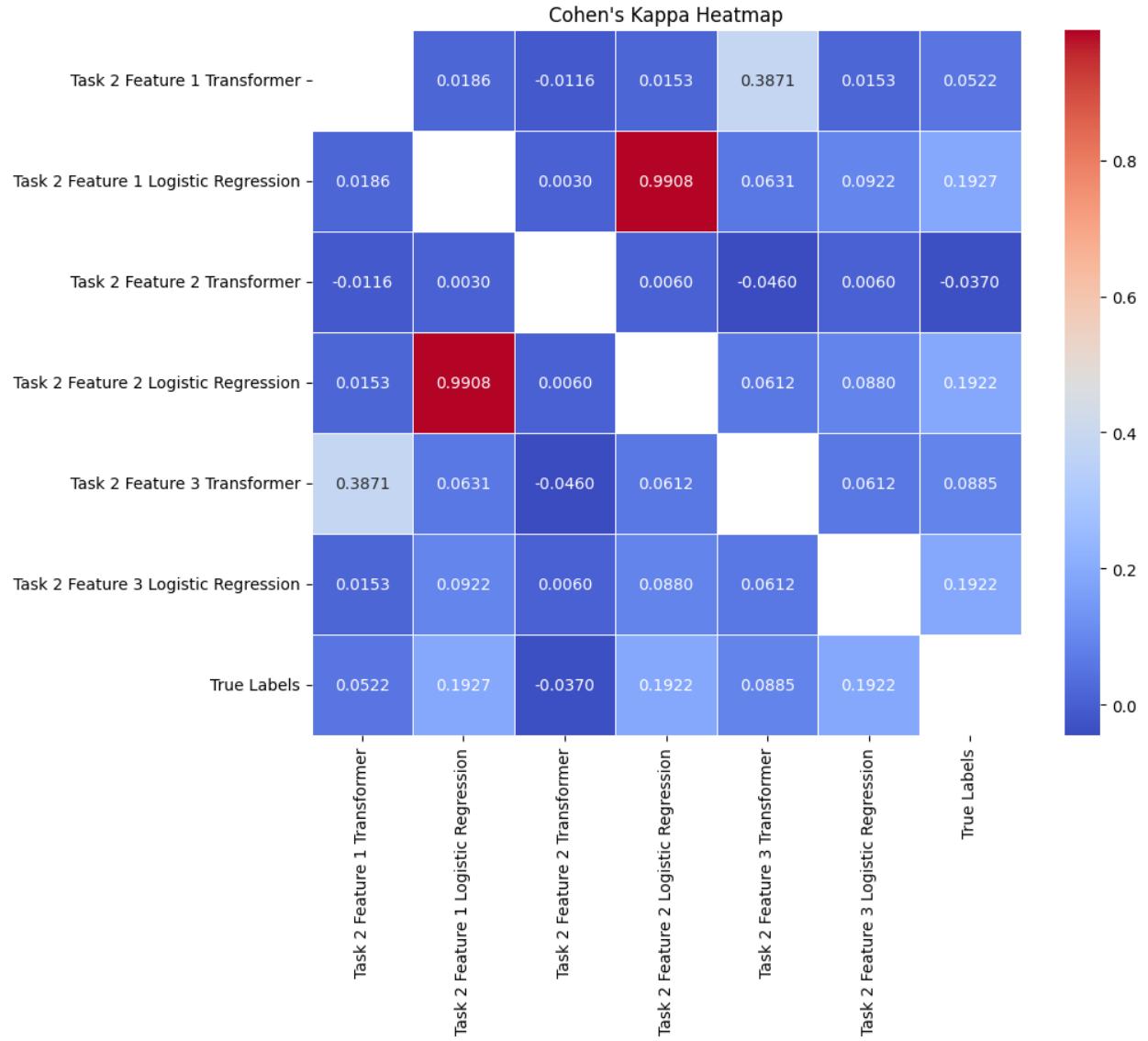


Figure 2. Heatmap of Cohen's Kappa Results on Multi-Class Classification Task

The results of Cohen's K shows that for most models, the agreement between predictions are low. This is true even for the models with high accuracy. For example, the transformer models for features 1 and 2 have similar accuracies (84% and 83.4% respectively). However, they have a Cohen's K value of -0.0116. This indicates that on the wrong predictions, these models do not predict the same incorrect class even if they have high accuracy on correct predictions.

Two Cohen's K values are of note. The first is between the logistic regression models of feature 1 and 2, which has a Cohen's K of 0.99, indicating an almost perfect positive agreement. This could be due to how the inputs for feature 1 and 2 are similar, and logistic regression models predict similar patterns for each of these. As such, the models predict similar classes, even for incorrect predictions.

The second is between the transformer models of feature 1 and feature 3 with a Cohen's K of 0.39, indicating substantial positive agreement. Upon inspection, predictions of the transformer models for feature 3 only

predict mainly one class. This could cause a higher than expected agreement with the transformer model for feature 1, depending on how the model for feature 1 predicts incorrect predictions.

Matthews Correlation Coefficient Each model was compared against the ground truth to calculate their MCC values. The resulting values are shown in table 15.

	Task 2 Feature 1 Transformer	Task 2 Feature 1 Logistic Regression	Task 2 Feature 2 Transformer	Task 2 Feature 2 Logistic Regression	Task 2 Feature 3 Transformer	Task 2 Feature 3 Logistic Regression
MCC Score	0.066509038	0.194445905	-0.043092423	0.19418937	0.103772985	0.124725893

Table 15. MCC Scores for Each Model, Compared Against Ground Truth

The results of the MCC indicate that most models are not able to correctly predict across all classes evenly. For example, the transformer models for features 1 and 2 have MCC scores that indicate very poor predictions when considering performance across all classes. This could suggest that the model performs significantly worse when considering a class-to-class basis, possibly indicating bias towards a certain class (e.g., defaulting to a majority class for incorrect predictions).

However, for some models that do not have high accuracy, they appear to have higher MCC scores. For example, the transformer for feature 3 has an MCC score of 0.10. This particular transformer model predicted only the majority class, allowing it to minimize false positives and false negatives. As such, although they do not have a good accuracy, they are able to minimize misclassifications, leading to a decent MCC score.

Discussion

Multi-Label Classification Task

Interpretation of the statistical tests on the multi-label task can be considered in three stages. First, the traditional machine learning metrics show that transformer models are able to outperform their logistic regression baselines consistently. Secondly, the HL values provide further support that transformers are able to outperform logistic regression models, even when considering individual label predictions. However, the HL shows that logistic regression models perform better on a label-by-label basis than suggested by their accuracy. Finally, Friedman's test shows that the difference in model predictions are statistically significant.

Multi-Class Classification Task

Investigation on Cohen's K supports how logistic regression models learn similar patterns given similar inputs, while transformers might learn different weights with similar inputs. Furthermore, Cohen's K can also provide insights into what incorrect predictions are done. While this report does not investigate this behaviour in depth, the results of Cohen's K provides an indicator of possible model behaviour that could be investigated further.

Finally, the results of the MCC scores show that transformer models do not tend to each class equally, which could support the behaviour of the transformer for feature 1 to default to a majority class. While investigating the exact causes of these MCC scores are out of the scope of this report, MCC scores can provide indications on what the model struggles with, such as evenly predicting every class.

Friedman's Test on Multi-Class Classification

The application of Friedman's test on multi-class classification poses a challenge due to the output, which comes in the form of text. Friedman's test calculates differences between a series of predictions. However,

text data such as drug names need to be preprocessed into numerical variables. While some studies have used Friedman's test on textual data by converting it into a ranking task¹⁷, there is no consensus on how to handle this change in features. Future work could be done to tackle this issue.

Limitations

One limitation of this study was how the dataset was not investigated using certain assumption tests. For example, logistic regression models needs to meet the assumption of independent errors, which was not tested in this study. However, the most vital limitation of this paper is that interpretations are based on possible model behaviours, not actual model behaviours. Further in-depth investigation of proposed model behaviours could provide a deeper understanding into how statistical tests affect model evaluation. For example, an investigation whether the high Cohen's K value between the transformer models in task 2 feature 1 and feature 3 is due to predictions prioritizing the majority class or some other class imbalance would allow for deeper understanding of model behaviour.

Conclusion and Future Directions

In conclusion, this report has shown how statistical tests, combined with traditional machine learning metrics can help to better understand model behaviour. While statistical tests indicate model behaviour, traditional machine learning models are still important when considering practicality. For example, for multi-class classification, while the transformer models for feature 1 and 2 do not perform evenly on all classes, their accuracy could still make a transformer model preferred in performing predictions. In addition, while some studies have suggested how MCC has advantages over F1-scores and accuracy¹², these metrics should still be understood as providing ideas on model behaviour, instead of replacing traditional machine learning metrics.

Future work can investigate more sophisticated workflows to understand model behaviour based on statistical tests. Additional statistical tests could also be integrated, such as testing assumptions of independence, which could guide what statistical tests fit a specific task and to identify possible problems with a provided dataset¹⁸.

References

1. Zolnoori M, Fung K, Patrick T, Fontelo P, Kharrazi H, Faiola A, et al. The PsyTAR Dataset: From Patients Generated Narratives to a Corpus of Adverse Drug Events and Effectiveness of Psychiatric Medications. *Data in Brief*. 2019;24:103838.
2. Zolnoori M, Fung KW, Patrick TB, Fontelo P, Kharrazi H, Faiola A, et al. A systematic approach for developing a corpus of patient reported adverse drug events: A case study for SSRI and SNRI medications. *Journal of Biomedical Informatics*. 2019;90:103091. Available from: <https://www.sciencedirect.com/science/article/pii/S1532046419300012>.
3. Dror R, Baumer G, Shlomov S, Reichart R. The Hitchhiker's Guide to Testing Statistical Significance in Natural Language Processing. In: Gurevych I, Miyao Y, editors. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Association for Computational Linguistics; 2018. p. 1383-92. Available from: <https://aclanthology.org/P18-1128>.
4. Wang X, Hripcsak G, Markatou M, Friedman C. Active Computerized Pharmacovigilance Using Natural Language Processing, Statistics, and Electronic Health Records: A Feasibility Study. *Journal of the American Medical Informatics Association*. 2009;2009(16):328-37. Available from: <https://doi.org/10.1197/jamia.M3028>.
5. Rainio O, Tauho J, Klén R. Evaluation metrics and statistical tests for machine learning. *Sci Rep*. 2024;14(6086).
6. Xu Z, Liu R, Yang S, Chai Z, Yuan C. Learning Imbalanced Data with Vision Transformers; 2023.
7. Díez J, Luaces O, del Coz JJ, Bahamonde A. Optimizing different loss functions in multilabel classifications. *Regular Paper*. 2015;3:107-18. Available from: <https://link.springer.com/article/10.1007/s13748-014-0060-7>.
8. Hui L, Belkin M. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks; 2021.
9. Cohen J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*. 1960;20(1):37-46. Available from: <https://doi.org/10.1177/001316446002000104>.
10. McHugh ML. Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)*. 2012;22. Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/#:~:text=Cohen%20suggested%20the%20Kappa%20result,1.00%20as%20almost%20perfect%20agreement>.
11. Jurman G, Riccadonna S, Furlanello C. A Comparison of MCC and CEN Error Measures in Multi-Class Prediction. *PLOS ONE*. 2012;2012(8):1-8. Available from: <https://doi.org/10.1371/journal.pone.0041882>.
12. Chicco D, Jurman G. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining*. 2023;2023(2):16.
13. Yang S, Berdine G. The receiver operating characteristic (ROC) curve. *The Southwest Respiratory and Critical Care Chronicles*. 2017 May;5(19):34-6. Available from: <https://pulmonarychronicles.com/index.php/pulmonarychronicles/article/view/391>.
14. Friedman M. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*. 1940;11(1):86-92. Available from: <http://www.jstor.org/stable/2235971>.
15. Friedman M. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*. 1937;32(200):675-701. Available from: <https://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522>.

16. Hoffman JIE. Chapter 26 - Analysis of Variance II. More Complex Forms. In: Hoffman JIE, editor. Biostatistics for Medical and Biomedical Practitioners. Academic Press; 2015. p. 421-47. Available from: <https://www.sciencedirect.com/science/article/pii/B9780128023877000263>.
17. Wang J, Kan H, Meng F, Mu Q, Shi G, Xiao X. Fake Review Detection Based on Multiple Feature Fusion and Rolling Collaborative Training. IEEE Access. 2020 01;8:182625-39.
18. Flatt C, Jacobs R. Principle Assumptions of Regression Analysis: Testing, Techniques, and Statistical Reporting of Imperfect Data Sets. Advances in Developing Human Resources. 2019 11;21:484-502.

Appendix

Appendix A - Equations for Statistical Tests

Cohen's K:

$$\kappa = \frac{p_o - p_e}{1 - p_e} \quad (1)$$

where:

p_o = relative observed agreement among raters
 p_e = hypothetical probability of chance agreement

(p_o) is calculated as:

$$p_o = \frac{a + d}{a + b + c + d} \quad (2)$$

(p_e) is calculated as:

$$p_e = \left(\frac{(a + b)}{a + b + c + d} \cdot \frac{(a + c)}{a + b + c + d} \right) + \left(\frac{(c + d)}{a + b + c + d} \cdot \frac{(b + d)}{a + b + c + d} \right) \quad (3)$$

where:

- a = Number of times both annotators agreed the sample was positive
- d = Number of times both annotators agreed the sample was negative
- b = Number of times Annotator A said positive but Annotator B said negative
- c = Number of times Annotator A said negative but Annotator B said positive

Matthews Correlation Coefficient:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4)$$

where:

TP : True Positive
 FP : False Positive
 TN : True Negative
 FN : False Negative

Hamming Loss:

$$\text{HL} = \frac{1}{N \times L} \sum_{l=1}^L \sum_{i=1}^N (Y_{i,l} \neq X_{i,l})$$

where:

N : Number of samples

L : Number of labels

$Y_{i,l}$: True label for i th sample and
 l th label

\hat{y}_{ij} : Predicted label for i th sample and
 l th label

$(Y_{i,l} \neq \hat{y}_{ij})$: 1 if $y_{ij} \neq \hat{y}_{ij}$ and 0 otherwise

Accuracy, Precision, Recall, and F1-Scores:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (6)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (7)$$

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 * TP}{2 * TP + FP + FN} \quad (8)$$

where:

TP : True Positive

FP : False Positive

TN : True Negative

FN : False Negative

Friedman's Test:

$$T = \frac{12 \sum S_{ij}^2}{jk(j+1)} - 3k(j+1) \quad (9)$$

where:

T = test statistic

$\sum S_{ij}^2$ = sum of the squared sums of ranks for each prediction set

j = number of prediction sets

k = number of instances

Appendix B - Transformer Model Performances

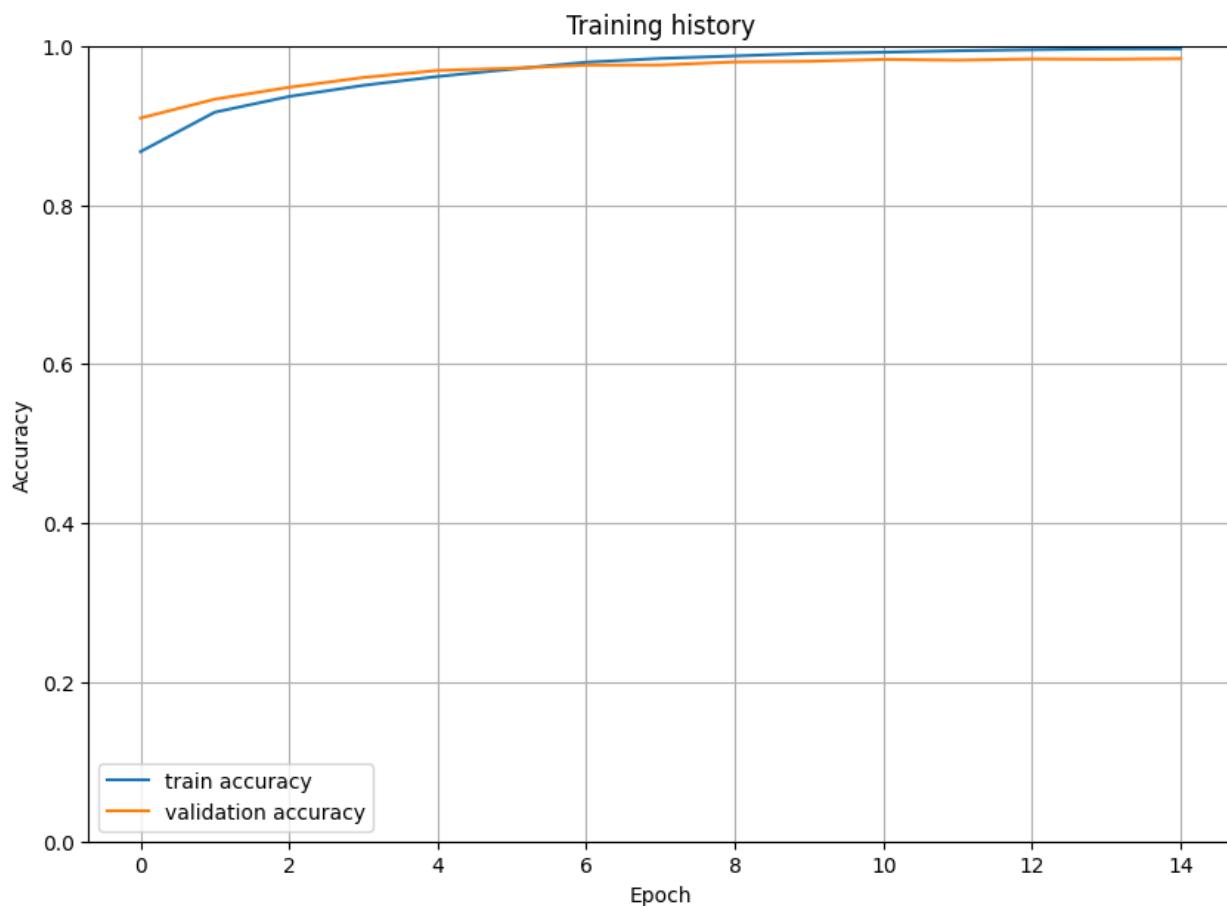


Figure 3. Task 1 Feature 1 Model Training (Total Training Time: 2056.51 Seconds)

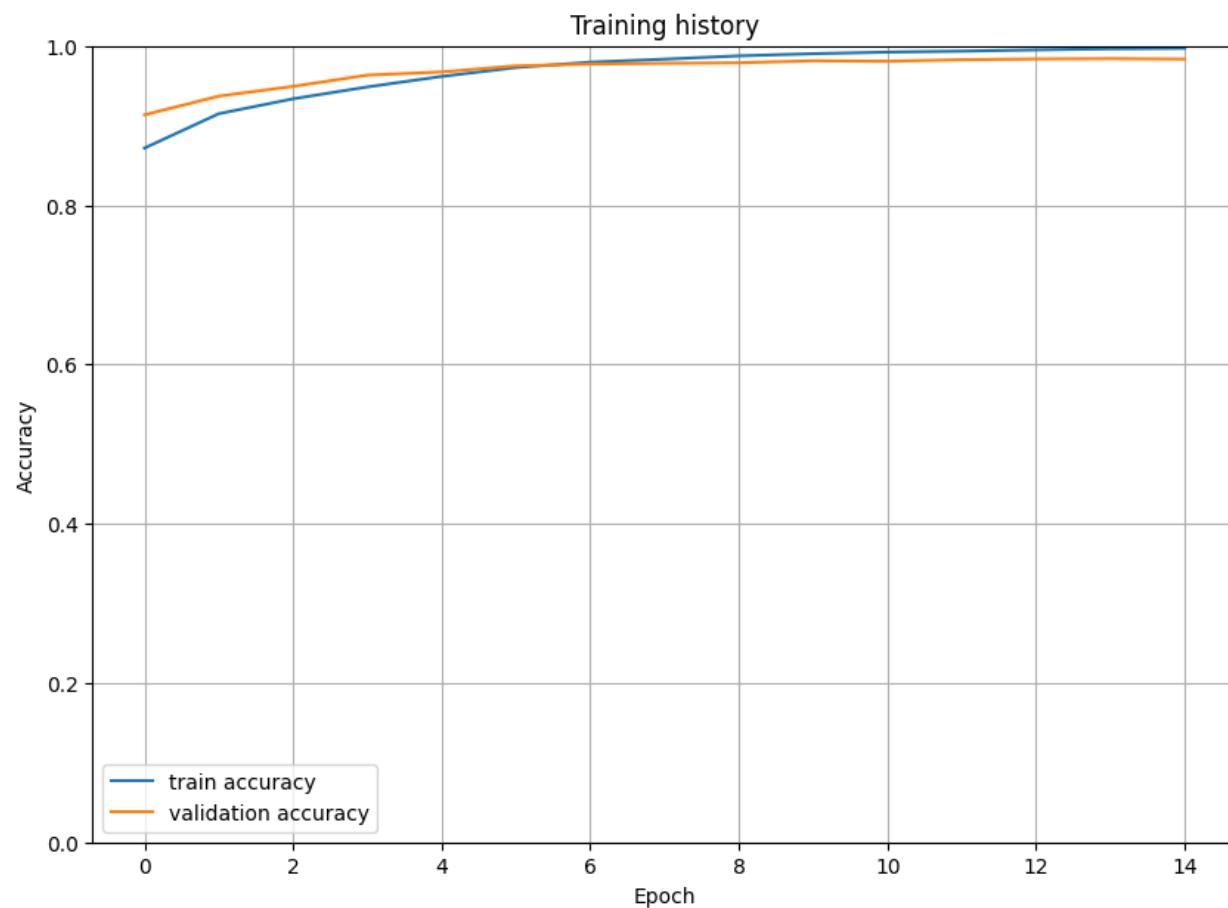


Figure 4. Task 1 Feature 2 Model Training (Total Training Time: 2037.76 Seconds)

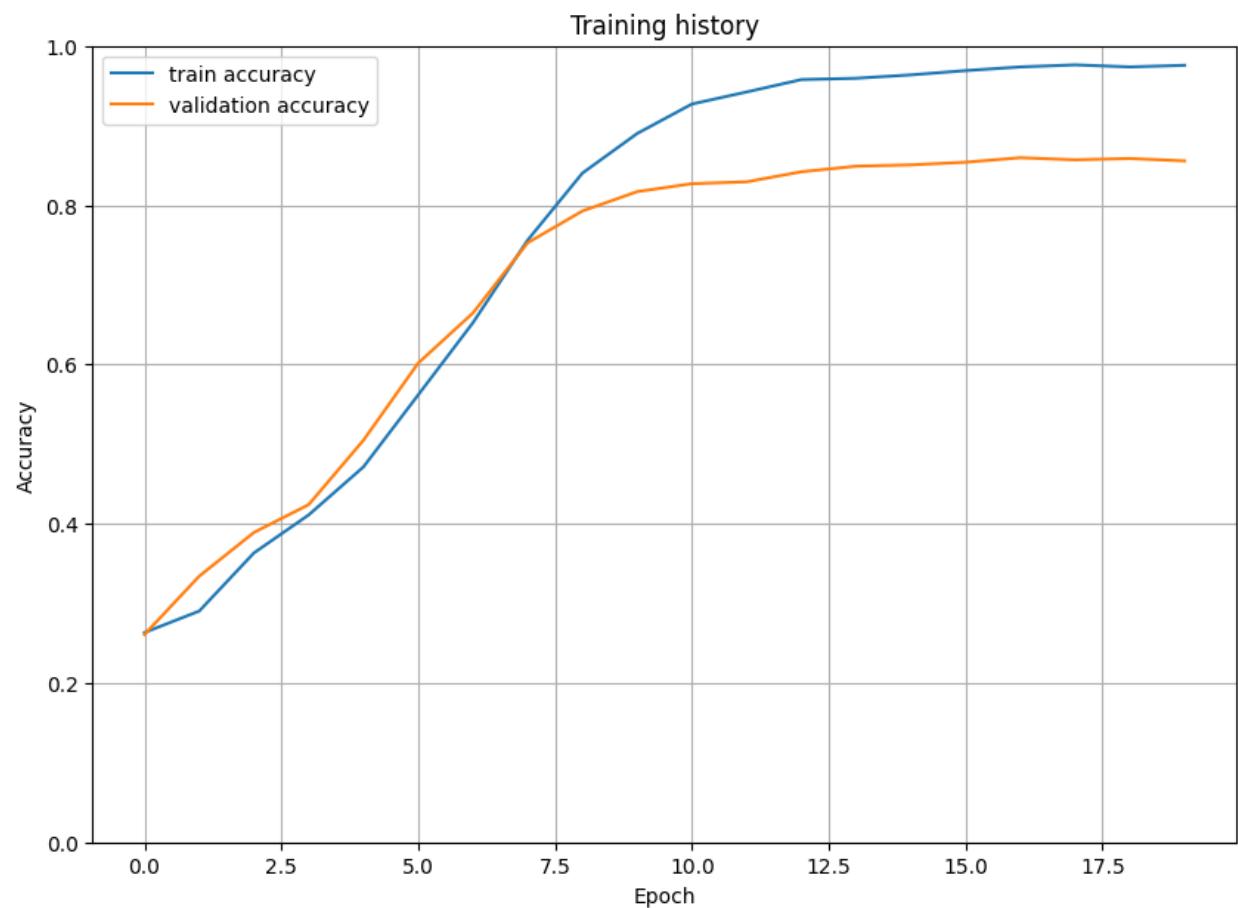


Figure 5. Task 2 Feature 1 Model Training (Total Training Time: 7471.89 Seconds)

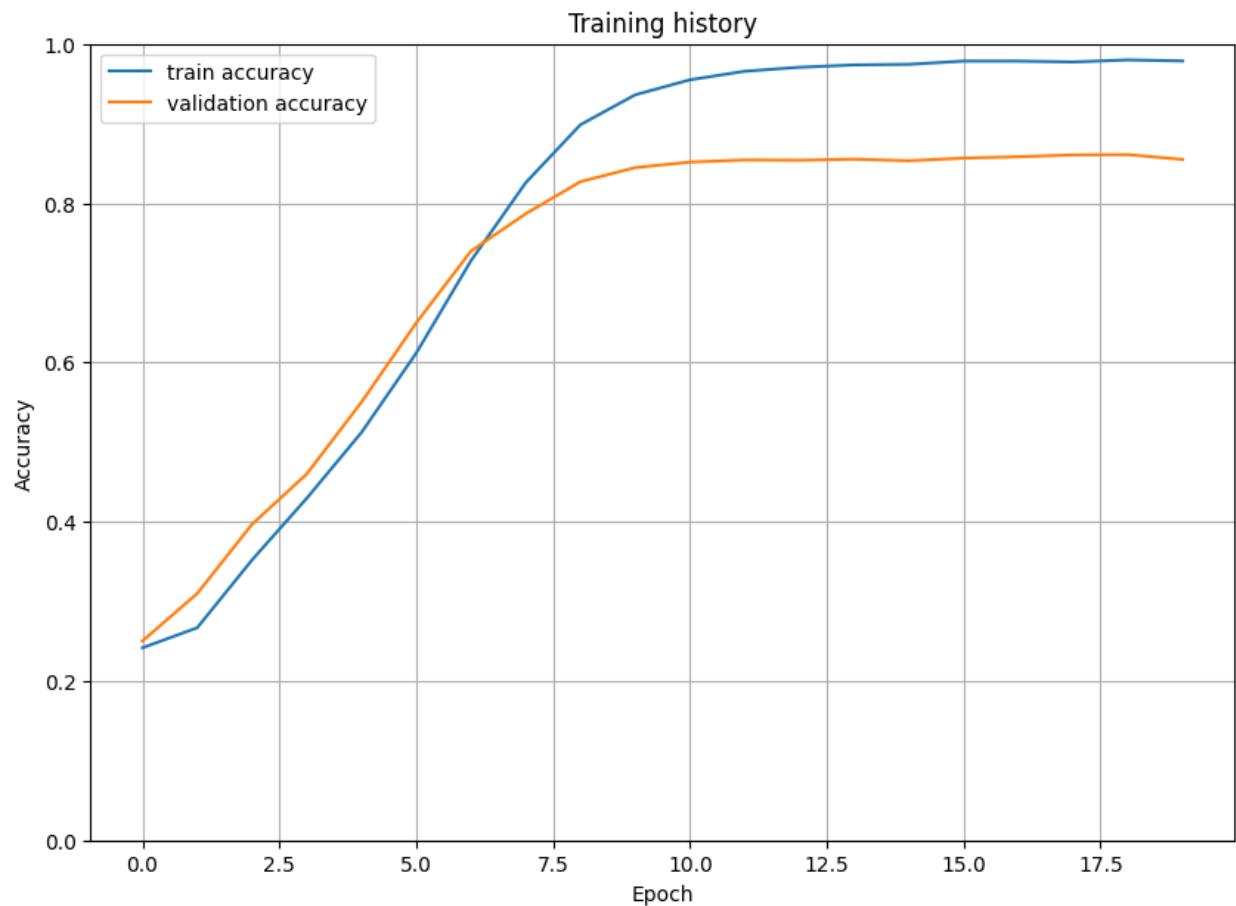


Figure 6. Task 2 Feature 2 Model Training (Total Training Time: 6587.32 Seconds)

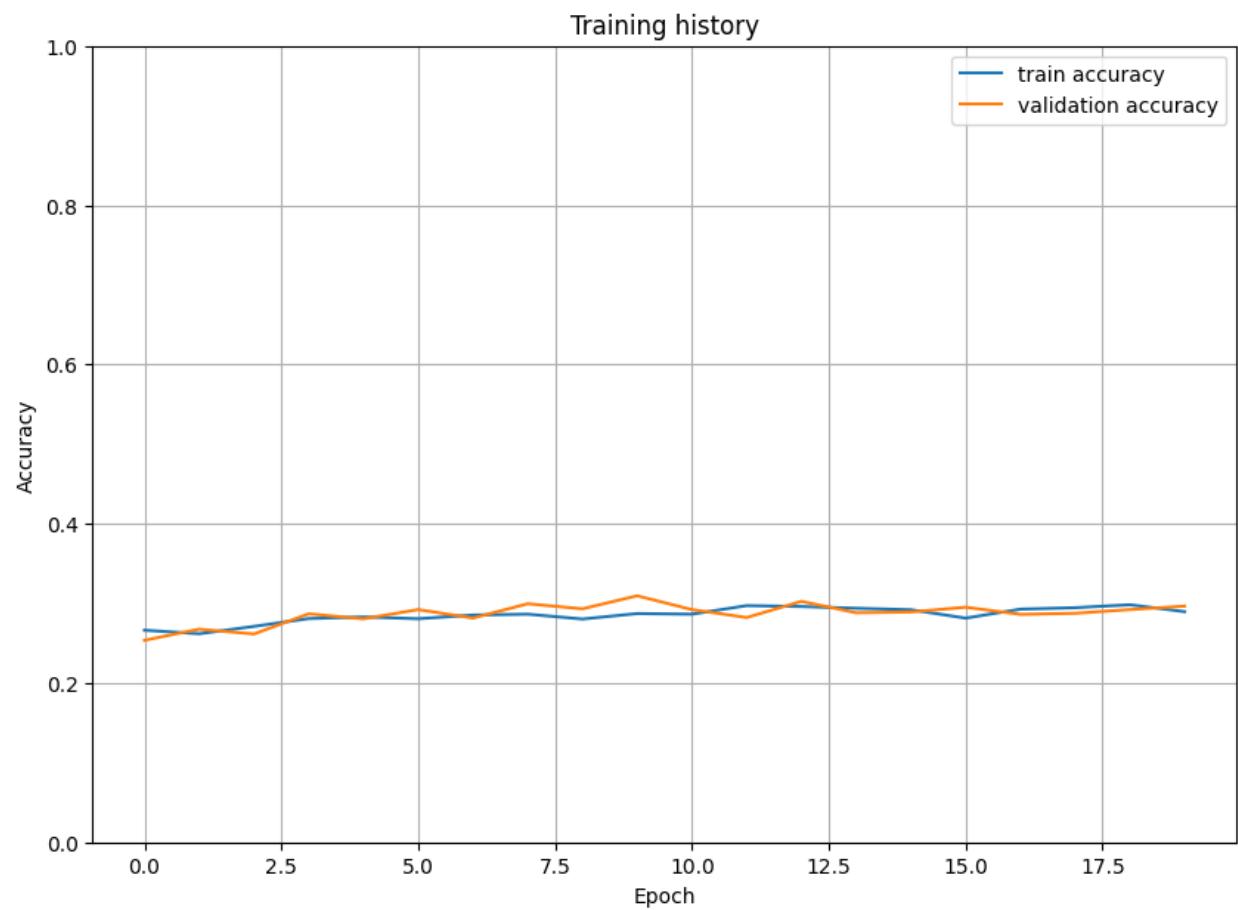


Figure 7. Task 2 Feature 3 Model Training (Total Training Time: 6590.17 Seconds)

Appendix C - Source Code

Preprocessing

```
In [ ]: ## Packages
##Imports
import pandas as pd
import transformers
import openpyxl
import numpy
import torch

# Does augmenting classifiers with structured data help?
# Do it for both the baseline classical ML and transformers then compare
# Try it with just structured data [0s and 1s]

In [ ]: !pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
Looking in indexes: https://download.pytorch.org/whl/cu121
Requirement already satisfied: torch in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (2.3.0+cu121)
Requirement already satisfied: torchvision in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (0.18.0+cu121)
Requirement already satisfied: torchaudio in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (2.3.0+cu121)
Requirement already satisfied: filelock in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (1.12)
Requirement already satisfied: networkx in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (2024.3.1)
Requirement already satisfied: mkl<=2021.4.0,>=2021.1.1 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torch) (2021.4.0)
Requirement already satisfied: numpy in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from torchvision) (10.2.0)
Requirement already satisfied: intel-openmp==2021.* in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from mkl<=2021.4.0,>=2021.1.1->torch) (2021.4.0)
Requirement already satisfied: tbb==2021.* in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from mkl<=2021.4.0,>=2021.1.1->torch) (2021.12.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in c:\users\julev\anaconda3\envs\cudaenv\lib\site-packages (from sympy->torch) (1.3.0)

In [ ]: torch.cuda.is_available()

Out[ ]: True

In [ ]: torch.cuda.get_device_name(0)

Out[ ]: 'NVIDIA GeForce RTX 3060 Ti'
```

```
In [ ]: ## Reading in the PSYtar data set and parsing it

## Use sentence_labelling sheet

fileName=".\\ONLINE_FORA\\PsyTAR_dataset.xlsx"
data=pd.ExcelFile(fileName)
sheets={}
for sheet in data.sheet_names:
    sheets[sheet]=data.parse(sheet)

## Remove the first two sheets (License and read_me)
sheets.pop('License',None)
sheets.pop('read_me',None)

## This will print out the sheet names for the whole excel
for sheet in sheets.keys():
    print(f"Sheet Name: {sheet}")

#To access a sheet, perform sheet['Sheet_Name']; e.g., sheets['Sample']
```

Sheet Name: Sample
Sheet Name: Sentence_Labeling
Sheet Name: ADR_Identified
Sheet Name: ADR_Mapped
Sheet Name: WD_Identified
Sheet Name: WD-Mapped
Sheet Name: SSI_Identified
Sheet Name: SSI_Mapped
Sheet Name: DI_Identified
Sheet Name: DI_Mapped

```
In [ ]: ## Vectorize data into TF-IDF and preprocess data

import nltk
import string
from sklearn.feature_extraction.text import TfidfVectorizer

def preprocess(text):
    text=str(text)
    tokens=nltk.word_tokenize(text.lower())
    tokens_clean=[t for t in tokens if (t not in stop_words) and (t not in punctuations)]
    return ' '.join(tokens_clean)

stop_words=nltk.corpus.stopwords.words('english')
punctuations=string.punctuation

df=data.parse('Sentence_Labeling')
df.drop(df.tail(1).index,inplace=True)
df['drug_id']=df['drug_id'].str.lower()
df['drug_name']=df['drug_id'].str.replace(r'\.\d+',' ',regex=True)
df['sentences']=df['sentences'].apply(preprocess)
df.fillna(0,inplace=True)
unique_drug_count=df['drug_name'].nunique()

tfidf=TfidfVectorizer()
tfidfSentences=tfidf.fit_transform(df['sentences'])

print(tfidfSentences)

print('Number of Drug Types:',unique_drug_count)
print(df)
```

```
## ADR: adverse drug reaction
## WD: withdrawal symptom
## EF: effective
## INF: ineffective
## SSI: Sign/symptom/illness - if report contains explicit SSI that patient experienced that is not explicitly mentioned as being resolved because of drug
## DI: drug indication - shows SSI that explicitly mentioned as being resolved because of drug
```

(0, 2183)	0.3560733570424169
(0, 2858)	0.533521055775061
(0, 2980)	0.33695305203929327
(0, 4715)	0.3577130194209486
(0, 4193)	0.36113778160835414
(0, 2071)	0.2627216248356759
(0, 5156)	0.2390135798252401
(0, 1829)	0.30083291559032393
(1, 2765)	0.4585168729312828
(1, 1351)	0.8886857021677111
(2, 2263)	0.2708094866549267
(2, 4586)	0.4316846093641784
(2, 5071)	0.44358904667953863
(2, 4827)	0.2570546169681927
(2, 3078)	0.23875893121623276
(2, 4151)	0.3207489298964512
(2, 1492)	0.29911785906957045
(2, 1201)	0.35264280744022497
(2, 4287)	0.3220996041881179
(3, 4012)	0.534375375933816
(3, 3326)	0.5729355939201095
(3, 1018)	0.39488574070665183
(3, 1237)	0.3054771887014283
(3, 5)	0.3700495936825698
(4, 4801)	0.3601630582861626
:	:
(6006, 3810)	0.266723700884449
(6006, 278)	0.30278988091992165
(6006, 4033)	0.24160419308016043
(6006, 1558)	0.24160419308016043
(6006, 2821)	0.21617383387506503
(6006, 3470)	0.20987895271102364
(6006, 1390)	0.23609327417139986
(6006, 2098)	0.1721949331238259
(6006, 2792)	0.15461917912599263
(6007, 5221)	0.40690224630890753
(6007, 1388)	0.36244934695018927
(6007, 2358)	0.40690224630890753
(6007, 987)	0.36244934695018927
(6007, 5165)	0.28822766104713454
(6007, 2896)	0.29736053264983103
(6007, 1619)	0.26742602182403663
(6007, 1809)	0.3151406469723878
(6007, 2777)	0.25257333627086354
(6008, 3452)	0.462967378880836
(6008, 4290)	0.4133719291682648
(6008, 1695)	0.41770974043639536
(6008, 3642)	0.462967378880836
(6008, 186)	0.3151272227530411
(6008, 2777)	0.24953491233016006
(6008, 5269)	0.25375538243161966

Number of Drug Types: 4

	id	comment_id	drug_id	sentence_index	\
0	1.0	1.0	lexapro.1	1.0	
1	2.0	1.0	lexapro.1	2.0	
2	3.0	1.0	lexapro.1	3.0	
3	4.0	1.0	lexapro.1	4.0	
4	5.0	1.0	lexapro.1	5.0	
...
6004	1545.0	228.0	effexorxr.228	14.0	
6005	1546.0	228.0	effexorxr.228	15.0	
6006	1547.0	228.0	effexorxr.228	16.0	
6007	1548.0	228.0	effexorxr.228	17.0	

```

6008 1549.0      228.0 effexorxr.228          18.0

sentences    ADR    WD    EF    INF  \
0   extreme weight gain short-term memory loss hai...  1.0  0.0  0.0  0.0
1                               detoxing lexapro  0.0  0.0  0.0  0.0
2   slowly cut dosage several months took vitamin ...  0.0  0.0  0.0  0.0
3                           10 days completely omg rough  0.0  0.0  0.0  0.0
4   flu-like symptoms dizziness major mood swings ...  0.0  1.0  0.0  0.0
...
6004                               increase dosage yet  0.0  0.0  0.0  0.0
6005      'm hoping able stay 75 mgs long possible  0.0  0.0  0.0  0.0
6006 reading withdrawals little scarey like said pe...  0.0  0.0  0.0  0.0
6007 effexor made huge difference life come experie...  0.0  0.0  1.0  0.0
6008           would small price pay able enjoy life  0.0  0.0  1.0  0.0

SSI DI Findings others rating category drug_name
0  0.0  0     0.0     0     1.0    ssri    lexapro
1  0.0  0     0.0     0     1.0    ssri    lexapro
2  0.0  0     0.0     1     1.0    ssri    lexapro
3  0.0  0     0.0     1     1.0    ssri    lexapro
4  0.0  0     0.0     0     1.0    ssri    lexapro
...
6004 0.0  0     0.0     1     5.0    snri    effexorxr
6005 0.0  0     0.0     1     5.0    snri    effexorxr
6006 0.0  0     0.0     1     5.0    snri    effexorxr
6007 0.0  0     0.0     0     5.0    snri    effexorxr
6008 0.0  0     0.0     0     5.0    snri    effexorxr

```

[6009 rows x 16 columns]

```
In [ ]: ## Investigate K-means clustering to see whether certain medicines have different side effects
from sklearn.cluster import KMeans

vocabulary=tfidf.get_feature_names_out()
print(vocabulary)

num_clusters=4
km = KMeans(n_clusters=num_clusters,random_state=42)
km.fit(tfidfSentences)
top_words=km.cluster_centers_.argsort()
print(vocabulary[top_words])
df['cluster']=km.labels_

['00' '04' '06' ... 'zoning' 'zopiclone' 'zyprexa']
[['00' 'paramedics' 'paralyzing' ... 'though' 'feel' 'even']
 ['easliy' 'snowball' 'smptoms' ... 'depression' 'effects' 'side']
 ['00' 'pantry' 'pant' ... 'side' 'effects' 'drug']
 ['kidney' 'nose' 'notable' ... 'day' 'first' 'taking']]
```

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency

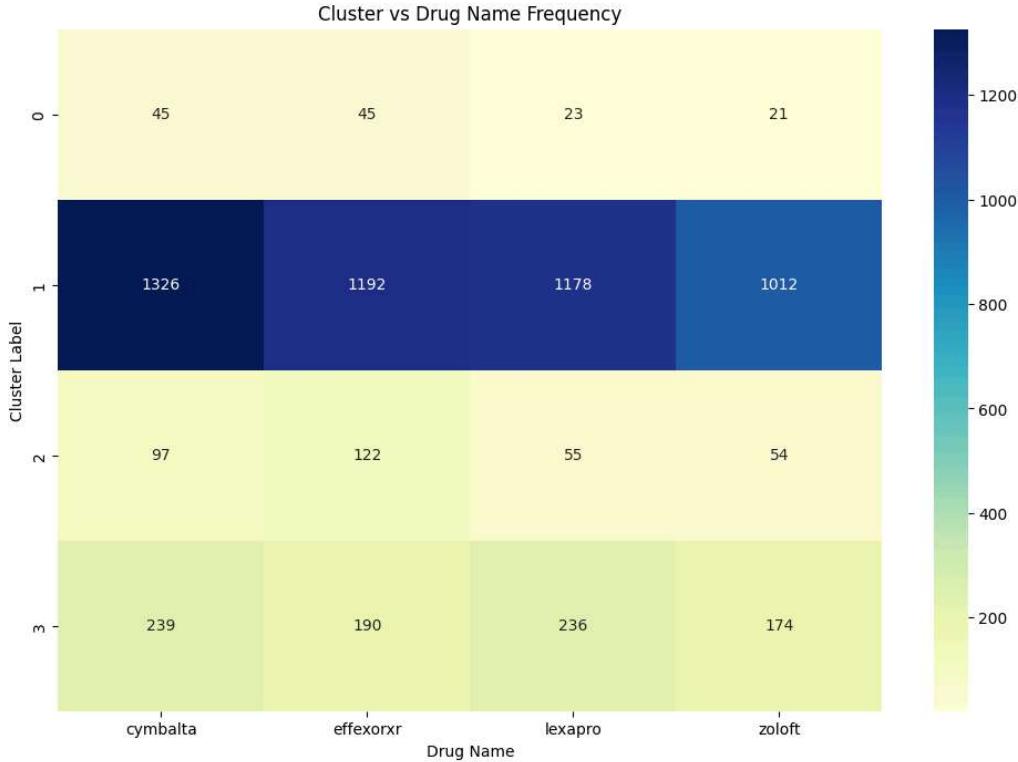
print(df['drug_name'].value_counts())

contingency_table = pd.crosstab(df['cluster'], df['drug_name'])
print(contingency_table)

plt.figure(figsize=(12, 8))
sns.heatmap(contingency_table, annot=True, fmt="d", cmap="YlGnBu")
plt.title('Cluster vs Drug Name Frequency')
plt.ylabel('Cluster Label')
plt.xlabel('Drug Name')
plt.show()
```

```
chi2, p, dof, expected = chi2_contingency(contingency_table)
print(f"Chi2: {chi2}, p-value: {p}")
```

```
drug_name
cymbalta    1707
effexorxr   1549
lexapro     1492
zoloft       1261
Name: count, dtype: int64
drug_name  cymbalta  effexorxr  lexapro  zoloft
cluster
0           45        45       23      21
1          1326      1192     1178    1012
2           97        122       55      54
3          239       190      236     174
```



```
In [ ]: from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans

pca = PCA(n_components=3)
X_dim = pca.fit_transform(tfidfSentences.toarray())

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(X_dim[:, 0], X_dim[:, 1], X_dim[:, 2], c=km.labels_, cmap='viridis', s=1
ax.set_xlabel('PCA 1')
```

```

ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')

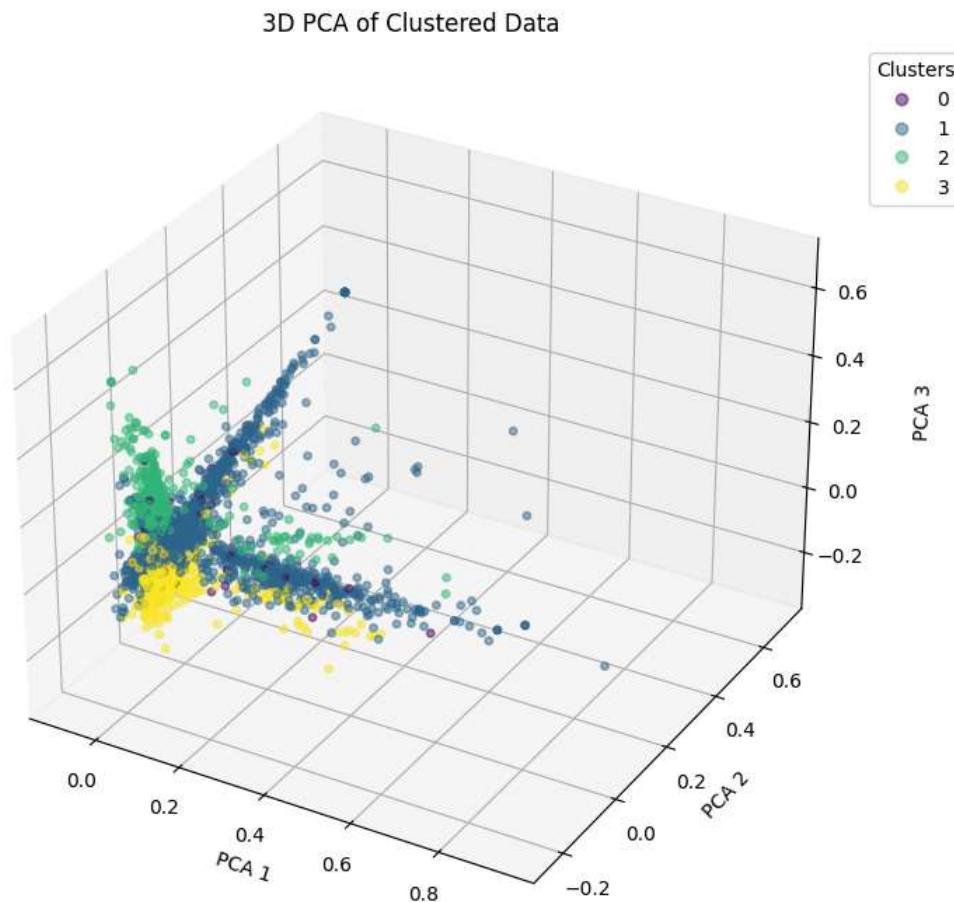
ax.xaxis.labelpad = 15

plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

legend = ax.legend(*scatter.legend_elements(), loc='upper left', bbox_to_anchor=(1, 1), title='Clusters')

plt.title('3D PCA of Clustered Data')
plt.show()

```



```

In [ ]: from sklearn.preprocessing import LabelEncoder
from matplotlib.patches import Patch
import numpy as np

label_encoder = LabelEncoder()
drug_labels = label_encoder.fit_transform(df['drug_name'])

pca = PCA(n_components=3)
X_dim = pca.fit_transform(tfidfSentences.toarray())

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

```

```

scatter = ax.scatter(X_dim[:, 0], X_dim[:, 1], X_dim[:, 2], c=drug_labels, cmap='viridis', s=100)

ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')

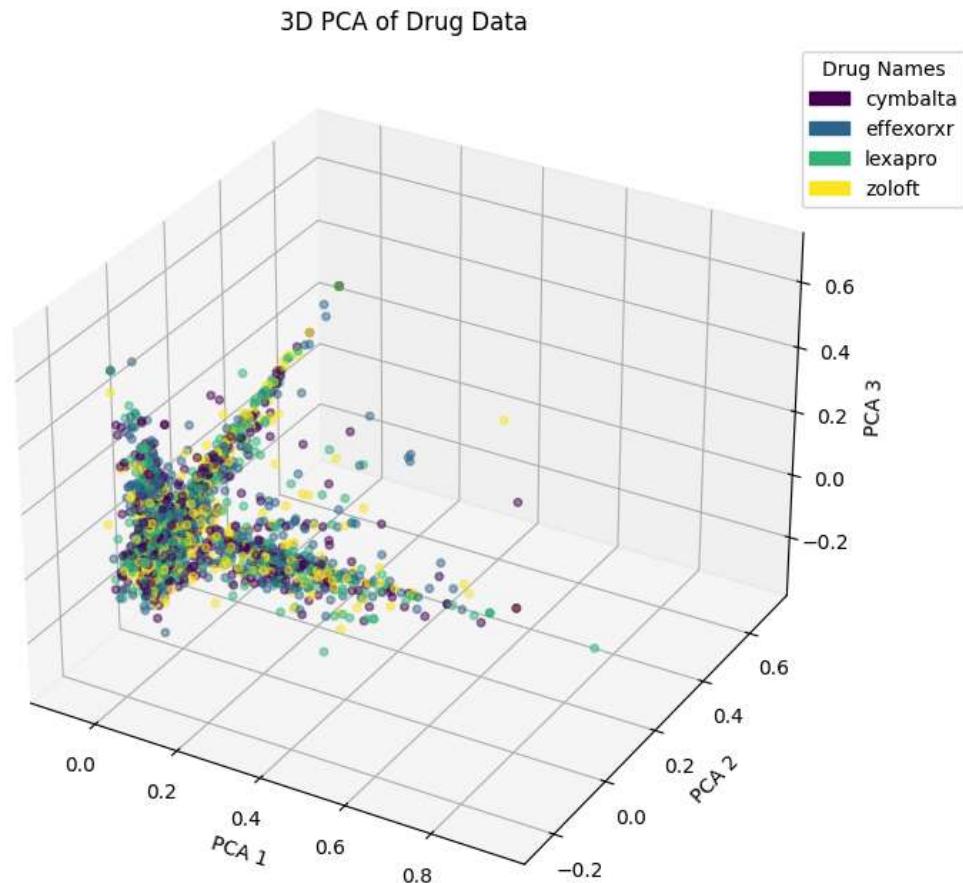
cmap = plt.cm.viridis

legend_handles = [Patch(color=cmap(norm), label=label) for label, norm in zip(label_encoder.classes_, np.linspace(0, 1, len(label_encoder.classes_)))]

plt.legend(handles=legend_handles, title="Drug Names", loc='upper right', bbox_to_anchor=(1.15, 0.9))

plt.title('3D PCA of Drug Data')
plt.show()

```



Multi-Label

Logistic Regression

```

In [ ]: ## This is just the plain sentences -- Task 1 Feature 1
from sklearn.model_selection import train_test_split

df['DI'] = pd.to_numeric(df['DI'], errors='coerce')

```

```

df_train, df_temp = train_test_split(df, random_state=42, test_size=0.55, shuffle=True)

df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True

x_train = df_train['sentences'].values
y_train = df_train[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_train = np.nan_to_num(y_train, nan=0).astype(int)

x_val = df_valid['sentences'].values
y_val = df_valid[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_val = np.nan_to_num(y_val, nan=0).astype(int)

x_test = df_test['sentences'].values
y_test = df_test[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_test = np.nan_to_num(y_test, nan=0).astype(int)

```

In []: # This one uses the drug_name with the sentences -- Task 1 Feature 2

```

from sklearn.model_selection import train_test_split

df['DI'] = pd.to_numeric(df['DI'], errors='coerce')
df['drug_sentences'] = df['drug_name'] + ' ' + df['sentences']

df_train, df_temp = train_test_split(df, random_state=42, test_size=0.55, shuffle=True)

df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True

x_train = df_train['drug_sentences'].values
y_train = df_train[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_train = np.nan_to_num(y_train, nan=0).astype(int)

x_val = df_valid['drug_sentences'].values
y_val = df_valid[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_val = np.nan_to_num(y_val, nan=0).astype(int)

x_test = df_test['drug_sentences'].values
y_test = df_test[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_test = np.nan_to_num(y_test, nan=0).astype(int)

```

In []: print('total length=',len(df))
print('%len of training set=',len(x_train)/len(df)*100)
print('%len of validation set=',len(x_val)/len(df)*100)
print('%len of test set=',len(x_test)/len(df)*100)

```

total length= 6000
%len of training set= 44.99916791479448
%len of validation set= 49.99167914794475
%len of test set= 5.009152937260775

```

In []: train_label_counts_0 = [sum(y_train[:, i] == 0) for i in range(y_train.shape[1])]
train_label_counts_1 = [sum(y_train[:, i] == 1) for i in range(y_train.shape[1])]
val_label_counts_0 = [sum(y_val[:, i] == 0) for i in range(y_val.shape[1])]
val_label_counts_1 = [sum(y_val[:, i] == 1) for i in range(y_val.shape[1])]
test_label_counts_0 = [sum(y_test[:, i] == 0) for i in range(y_test.shape[1])]
test_label_counts_1 = [sum(y_test[:, i] == 1) for i in range(y_test.shape[1])]

distribution_df = pd.DataFrame({
 'Label': ['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI'],
 'Train Count (0)': train_label_counts_0,
 'Train Count (1)': train_label_counts_1,
 'Validation Count (0)': val_label_counts_0,
 'Validation Count (1)': val_label_counts_1,
 'Test Count (0)': test_label_counts_0,
 'Test Count (1)': test_label_counts_1
})

```

        'Validation Count (1)': val_label_counts_1,
        'Test Count (0)': test_label_counts_0,
        'Test Count (1)': test_label_counts_1
    })

# Display the DataFrame
print(distribution_df)

  Label Train Count (0) Train Count (1) Validation Count (0) \
0   ADR          1712           992            1936
1    WD          2507           197            2781
2    EF          2188           516            2488
3   INF          2548           156            2836
4   SSI          2358           346            2595
5    DI          2476           228            2743

   Validation Count (1) Test Count (0) Test Count (1)
0                  1068       193         108
1                   223       283         18
2                   516       246         55
3                   168       288         13
4                   409       267         34
5                   261       274         27

```

Come here to change prediction variables

```

In [ ]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
multi_label_y = y_test
tfidf_vectorizer = TfidfVectorizer()
x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)
x_val_tfidf = tfidf_vectorizer.transform(x_val)
x_test_tfidf = tfidf_vectorizer.transform(x_test)

# param_grid = {
#     'estimator__C': [0.01, 0.1, 1, 10, 100, 1000],
#     'estimator__penalty': ['l1', 'l2'],
#     'estimator__solver': ['liblinear', 'saga', 'lbfgs'],
# }

param_grid = {
    'estimator__C': [10],
    'estimator__penalty': ['l2'],
    'estimator__solver': ['saga'],
}

log_reg = LogisticRegression()
multi_target_lr = MultiOutputClassifier(log_reg, n_jobs=-1)
grid_search = GridSearchCV(multi_target_lr, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train_tfidf, y_train)

print("best param ", grid_search.best_params_)
print("cross val accuracy ", grid_search.best_score_)

# feature2: best param {'estimator__C': 10, 'estimator__penalty': 'l2', 'estimator__solver':
# feature1: best param {'estimator__C': 10, 'estimator__penalty': 'l2', 'estimator__solver': 'saga'},

best_model = grid_search.best_estimator_

```

```

y_test_pred = best_model.predict(x_test_tfidf)

### adjust and comment out the predictions depending on what feature you are using
#####
# task_1_logistic_f1 = y_test_pred

# task_1_logistic_f2 = y_test_pred
#####
test_accuracy = accuracy_score(y_test, y_test_pred)
test_f1_micro = f1_score(y_test, y_test_pred, average='micro')
test_f1_macro = f1_score(y_test, y_test_pred, average='macro')
test_precision_micro = precision_score(y_test, y_test_pred, average='micro')
test_precision_macro = precision_score(y_test, y_test_pred, average='macro')
test_recall_micro = recall_score(y_test, y_test_pred, average='micro')
test_recall_macro = recall_score(y_test, y_test_pred, average='macro')

print(f'Test Accuracy: {test_accuracy:.4f}')
print(f'Test F1 Score (Micro): {test_f1_micro:.4f}')
print(f'Test F1 Score (Macro): {test_f1_macro:.4f}')
print(f'Test Precision (Micro): {test_precision_micro:.4f}')
print(f'Test Precision (Macro): {test_precision_macro:.4f}')
print(f'Test Recall (Micro): {test_recall_micro:.4f}')
print(f'Test Recall (Macro): {test_recall_macro:.4f}')

class_rep = classification_report(y_test,y_test_pred)
print(class_rep)
for i, label in enumerate(['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']):
    print(f'{label}:')
    #[tn, fp,
    # fn, tp]
    cm = confusion_matrix(y_test[:, i], y_test_pred[:, i])
    print(f'{cm}')

    report = classification_report(y_test[:, i], y_test_pred[:, i], zero_division=1, target_n
    print(f'{report}')

```

```

best param {'estimator__C': 10, 'estimator__penalty': 'l2', 'estimator__solver': 'saga'}
cross val accuracy 0.5773040323132745
Test Accuracy: 0.5249
Test F1 Score (Micro): 0.5308
Test F1 Score (Macro): 0.4719
Test Precision (Micro): 0.6707
Test Precision (Macro): 0.6460
Test Recall (Micro): 0.4392
Test Recall (Macro): 0.3774
      precision    recall   f1-score  support
      0       0.63     0.52     0.57      108
      1       0.44     0.22     0.30      18
      2       0.74     0.36     0.49      55
      3       0.43     0.23     0.30      13
      4       0.86     0.56     0.68      34
      5       0.77     0.37     0.50      27
  micro avg       0.67     0.44     0.53      255
  macro avg       0.65     0.38     0.47      255
weighted avg       0.68     0.44     0.53      255
samples avg       0.30     0.29     0.29      255

ADR:
[[160 33]
 [ 52 56]]
      precision    recall   f1-score  support
      0       0.75     0.83     0.79      193
      1       0.63     0.52     0.57      108
accuracy                   0.72      301
macro avg       0.69     0.67     0.68      301
weighted avg       0.71     0.72     0.71      301

WD:
[[278 5]
 [ 14 4]]
      precision    recall   f1-score  support
      0       0.95     0.98     0.97      283
      1       0.44     0.22     0.30      18
accuracy                   0.94      301
macro avg       0.70     0.60     0.63      301
weighted avg       0.92     0.94     0.93      301

EF:
[[239 7]
 [ 35 20]]
      precision    recall   f1-score  support
      0       0.87     0.97     0.92      246
      1       0.74     0.36     0.49      55
accuracy                   0.86      301
macro avg       0.81     0.67     0.70      301
weighted avg       0.85     0.86     0.84      301

INF:
[[284 4]
 [ 10 3]]
      precision    recall   f1-score  support

```

	0	0.97	0.99	0.98	288
	1	0.43	0.23	0.30	13
accuracy				0.95	301
macro avg		0.70	0.61	0.64	301
weighted avg		0.94	0.95	0.95	301
SSI:					
	[[264 3]				
	[15 19]]				
	precision	recall	f1-score	support	
	0	0.95	0.99	0.97	267
	1	0.86	0.56	0.68	34
accuracy				0.94	301
macro avg		0.90	0.77	0.82	301
weighted avg		0.94	0.94	0.93	301
DI:					
	[[271 3]				
	[17 10]]				
	precision	recall	f1-score	support	
	0	0.94	0.99	0.96	274
	1	0.77	0.37	0.50	27
accuracy				0.93	301
macro avg		0.86	0.68	0.73	301
weighted avg		0.93	0.93	0.92	301

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in samples with no pr
edicted labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in samples with no true
labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in samples with no true
nor predicted labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Transformers

```
In [ ]: # ## This is just the plain sentences -- Task 1 Feature 1
# from sklearn.model_selection import train_test_split

# df['DI'] = pd.to_numeric(df['DI'], errors='coerce')

# newDf=df
# newDf['DI'].fillna(0, inplace=True)
# newDf.drop(columns=['id', "comment_id", "drug_id", "sentence_index", "Findings", "others",

# # x = df['sentences'].values
# # y = df[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
# # y = np.nan_to_num(y, nan=0).astype(int)

# # Split data
```

```
# df_train, df_temp = train_test_split(newDf, random_state=42, test_size=0.55, shuffle=True)
# df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)
```

```
In [ ]: print('total length=',len(df))
print('%len of training set=',len(df_train)/len(df)*100)
print('%len of validation set=',len(df_valid)/len(df)*100)
print('%len of test set=',len(df_test)/len(df)*100)

total length= 6009
%len of training set= 44.99916791479448
%len of validation set= 49.99167914794475
%len of test set= 5.009152937260775
```

```
In [ ]: # This one uses the drug_name with the sentences -- Task 1 Feature 2

from sklearn.model_selection import train_test_split

df['DI'] = pd.to_numeric(df['DI'], errors='coerce')
newDf=df
newDf['DI'].fillna(0, inplace=True)
newDf['drug_sentences'] = df['drug_name'] + ' ' + df['sentences']
newDf.drop(columns=["id", "comment_id", "drug_id", "sentence_index", "Findings", "others", "r
# x = df['drug_sentences'].values
# y = df[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
# y = np.nan_to_num(y, nan=0).astype(int)

# Split data
df_train, df_temp = train_test_split(newDf, random_state=42, test_size=0.55, shuffle=True)
df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)
```

C:\Users\julev\AppData\Local\Temp\ipykernel_16412\347856079.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
newDf['DI'].fillna(0, inplace=True)
```

```
In [ ]: print(newDf)
```

```

          ADR  WD  EF  INF  SSI  DI  cluster \
0    1.0  0.0  0.0  0.0  0.0  0.0      1
1    0.0  0.0  0.0  0.0  0.0  0.0      1
2    0.0  0.0  0.0  0.0  0.0  0.0      1
3    0.0  0.0  0.0  0.0  0.0  0.0      1
4    0.0  1.0  0.0  0.0  0.0  0.0      1
...
6004  0.0  0.0  0.0  0.0  0.0  0.0      1
6005  0.0  0.0  0.0  0.0  0.0  0.0      1
6006  0.0  0.0  0.0  0.0  0.0  0.0      1
6007  0.0  0.0  1.0  0.0  0.0  0.0      1
6008  0.0  0.0  1.0  0.0  0.0  0.0      1

                           drug_sentences
0    lexapro extreme weight gain short-term memory ...
1    lexapro detoxing lexapro
2    lexapro slowly cut dosage several months took ...
3    lexapro 10 days completely omg rough
4    lexapro flu-like symptoms dizziness major mood...
...
6004           effexorxr increase dosage yet
6005 effexorxr 'm hoping able stay 75 mgs long poss...
6006 effexorxr reading withdrawals little scarey li...
6007 effexorxr effexor made huge difference life co...
6008 effexorxr would small price pay able enjoy life

[6009 rows x 8 columns]

```

```
In [ ]: print(f"Train: {df_train.shape}, Test: {df_test.shape}, Valid: {df_valid.shape}")
```

```
Train: (2704, 8), Test: (301, 8), Valid: (3004, 8)
```

```
In [ ]: # Hyperparameters
MAX_LEN = 256
TRAIN_BATCH_SIZE = 8
VALID_BATCH_SIZE = 8
TEST_BATCH_SIZE = 8
EPOCHS = 15
LEARNING_RATE = 1e-05
THRESHOLD = 0.5 # threshold for the sigmoid
```

```
In [ ]: from transformers import BertTokenizer, BertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\huggingface_hub\file_download.py:1132:
FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads
always resume when possible. If you want to force a new download, use `force_download=True`.
warnings.warn(
```

```
In [ ]: #df_train['sentences']
```

```
In [ ]: class CustomDatasetTask1(torch.utils.data.Dataset):
    def __init__(self, df, tokenizer, max_len, target_list):
        self.tokenizer = tokenizer
        self.df = df
        self.title = list(df['drug_sentences'])
        self.targets = self.df[target_list].values
        self.max_len = max_len

    def __len__(self):
        return len(self.title)

    def __getitem__(self, index):
```

```

        title = str(self.title[index])
        title = " ".join(title.split())
        inputs = self.tokenizer.encode_plus(
            title,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True,
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )
        return {
            'input_ids': inputs['input_ids'].flatten(),
            'attention_mask': inputs['attention_mask'].flatten(),
            'token_type_ids': inputs["token_type_ids"].flatten(),
            'targets': torch.FloatTensor(self.targets[index]),
            'title': title
        }

    class CustomDatasetTask2(torch.utils.data.Dataset):
        def __init__(self, df, tokenizer, max_len, target_list):
            self.tokenizer = tokenizer
            self.df = df
            self.title = list(df['drug_sentences'])
            self.targets = self.df[target_list].values
            self.max_len = max_len

        def __len__(self):
            return len(self.title)

        def __getitem__(self, index):
            title = str(self.title[index])
            title = " ".join(title.split())
            inputs = self.tokenizer.encode_plus(
                title,
                None,
                add_special_tokens=True,
                max_length=self.max_len,
                padding='max_length',
                return_token_type_ids=True,
                truncation=True,
                return_attention_mask=True,
                return_tensors='pt'
            )
            return {
                'input_ids': inputs['input_ids'].flatten(),
                'attention_mask': inputs['attention_mask'].flatten(),
                'token_type_ids': inputs["token_type_ids"].flatten(),
                'targets': torch.FloatTensor(self.targets[index]),
                'title': title
            }

```

In []: target_list = list(['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI'])
target_list

Out[]: ['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']

In []: #change this to CustomDatasetTask_ for feature _
train_dataset = CustomDatasetTask2(df_train, tokenizer, MAX_LEN, target_list)

```
valid_dataset = CustomDatasetTask2(df_valid, tokenizer, MAX_LEN, target_list)
test_dataset = CustomDatasetTask2(df_test, tokenizer, MAX_LEN, target_list)
```

```
In [ ]: next(iter(train_dataset))
```

```
In [ ]: train_data_loader = torch.utils.data.DataLoader(train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    shuffle=True,
    num_workers=0
)
```

```
val_data_loader = torch.utils.data.DataLoader(valid_dataset,
    batch_size=VALID_BATCH_SIZE,
    shuffle=False,
    num_workers=0
)

test_data_loader = torch.utils.data.DataLoader(test_dataset,
    batch_size=TEST_BATCH_SIZE,
    shuffle=False,
    num_workers=0
)
```

```
In [ ]: device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
class BERTClass(torch.nn.Module):
    def __init__(self):
        super(BERTClass, self).__init__()
        self.bert_model = BertModel.from_pretrained('bert-base-uncased', return_dict=True)
        self.dropout = torch.nn.Dropout(0.3)
        self.linear = torch.nn.Linear(768, 6)

    def forward(self, input_ids, attn_mask, token_type_ids):
        output = self.bert_model(
            input_ids,
            attention_mask=attn_mask,
            token_type_ids=token_type_ids
        )
        output_dropout = self.dropout(output.pooler_output)
        output = self.linear(output_dropout)
        return output

model = BERTClass()

# # Freezing BERT Layers: (tested, weaker convergence)
# for param in model.bert_model.parameters():
#     param.requires_grad = False

model.to(device)
```

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\huggingface_hub\file_download.py:1132:
FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads
always resume when possible. If you want to force a new download, use `force_download=True`.
warnings.warn(
```

```
Out[ ]: BERTClass(  
    (bert_model): BertModel(  
        (embeddings): BertEmbeddings(  
            (word_embeddings): Embedding(30522, 768, padding_idx=0)  
            (position_embeddings): Embedding(512, 768)  
            (token_type_embeddings): Embedding(2, 768)  
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)  
            (dropout): Dropout(p=0.1, inplace=False)  
        )  
        (encoder): BertEncoder(  
            (layer): ModuleList(  
                (0-11): 12 x BertLayer(  
                    (attention): BertAttention(  
                        (self): BertSelfAttention(  
                            (query): Linear(in_features=768, out_features=768, bias=True)  
                            (key): Linear(in_features=768, out_features=768, bias=True)  
                            (value): Linear(in_features=768, out_features=768, bias=True)  
                            (dropout): Dropout(p=0.1, inplace=False)  
                        )  
                        (output): BertSelfOutput(  
                            (dense): Linear(in_features=768, out_features=768, bias=True)  
                            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)  
                            (dropout): Dropout(p=0.1, inplace=False)  
                        )  
                    )  
                    (intermediate): BertIntermediate(  
                        (dense): Linear(in_features=768, out_features=3072, bias=True)  
                        (intermediate_act_fn): GELUActivation()  
                    )  
                    (output): BertOutput(  
                        (dense): Linear(in_features=3072, out_features=768, bias=True)  
                        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)  
                        (dropout): Dropout(p=0.1, inplace=False)  
                    )  
                )  
            )  
            (pooler): BertPooler(  
                (dense): Linear(in_features=768, out_features=768, bias=True)  
                (activation): Tanh()  
            )  
            (dropout): Dropout(p=0.3, inplace=False)  
            (linear): Linear(in_features=768, out_features=6, bias=True)  
        )  
    )
```

```
In [ ]: def loss_fn(outputs, targets):  
    return torch.nn.BCEWithLogitsLoss()(outputs, targets)
```

```
In [ ]: from transformers import AdamW  
  
# define the optimizer  
optimizer = AdamW(model.parameters(), lr = 1e-5)
```

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\transformers\optimization.py:521: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version.  
Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True`  
to disable this warning  
warnings.warn(
```

```
In [ ]: def train_model(training_loader, model, optimizer):  
    losses = []
```

```

correct_predictions = 0
num_samples = 0
# set model to training mode (activate dropout, batch norm)
model.train()
# initialize the progress bar
loop = tq.tqdm(enumerate(training_loader), total=len(training_loader),
                leave=True, colour='steelblue')
for batch_idx, data in loop:
    ids = data['input_ids'].to(device, dtype = torch.long)
    mask = data['attention_mask'].to(device, dtype = torch.long)
    token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
    targets = data['targets'].to(device, dtype = torch.float)

    # forward
    outputs = model(ids, mask, token_type_ids) # (batch,predict)=(32,8)
    loss = loss_fn(outputs, targets)
    losses.append(loss.item())
    # training accuracy, apply sigmoid, round (apply thresh 0.5)
    outputs = torch.sigmoid(outputs).cpu().detach().numpy().round()
    targets = targets.cpu().detach().numpy()
    correct_predictions += np.sum(outputs==targets)
    num_samples += targets.size # total number of elements in the 2D array

    # backward
    optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    # grad descent step
    optimizer.step()

    # Update progress bar
    #Loop.set_description(f"""
    #Loop.set_postfix(batch_loss={loss})
    # returning: trained model, model accuracy, mean loss
    return model, float(correct_predictions)/num_samples, np.mean(losses)

```

```
In [ ]: def eval_model(validation_loader, model, optimizer):
    losses = []
    correct_predictions = 0
    num_samples = 0
    # set model to eval mode (turn off dropout, fix batch norm)
    model.eval()

    with torch.no_grad():
        for batch_idx, data in enumerate(validation_loader, 0):
            ids = data['input_ids'].to(device, dtype = torch.long)
            mask = data['attention_mask'].to(device, dtype = torch.long)
            token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
            targets = data['targets'].to(device, dtype = torch.float)
            outputs = model(ids, mask, token_type_ids)

            loss = loss_fn(outputs, targets)
            losses.append(loss.item())

            # validation accuracy
            # add sigmoid, for the training sigmoid is in BCEWithLogitsLoss
            outputs = torch.sigmoid(outputs).cpu().detach().numpy().round()
            targets = targets.cpu().detach().numpy()
            correct_predictions += np.sum(outputs==targets)
            num_samples += targets.size # total number of elements in the 2D array

    return float(correct_predictions)/num_samples, np.mean(losses)
```

```
In [ ]: from collections import defaultdict
import time

torch.cuda.empty_cache()
history = defaultdict(list)
best_accuracy = 0

# model_save_path = './model2_task1_feature1_model(unfrozen)'
start_time = time.time()

# for epoch in range(1, EPOCHS+1):
#     print(f'Epoch {epoch}/{EPOCHS}')
#     model, train_acc, train_loss = train_model(train_data_loader, model, optimizer)
#     val_acc, val_loss = eval_model(val_data_loader, model, optimizer)

#     print(f'train_loss={train_loss:.4f}, val_loss={val_loss:.4f} train_acc={train_acc:.4f},'
# 
#     history['train_acc'].append(train_acc)
#     history['train_loss'].append(train_loss)
#     history['val_acc'].append(val_acc)
#     history['val_loss'].append(val_loss)

#     if val_acc > best_accuracy:
#         if not os.path.exists(model_save_path):
#             os.makedirs(model_save_path)
#         torch.save(model.state_dict(), os.path.join(model_save_path, 'model_state.bin'))
#         print(f"Model saved to {model_save_path}")
#         best_accuracy = val_acc

# end_time = time.time()
# total_train_time = end_time - start_time

# print(f"Total training time: {total_train_time:.2f} seconds")
# torch.cuda.empty_cache()
```

```
In [ ]: import matplotlib.pyplot as plt

# plt.rcParams["figure.figsize"] = (10,7)
# plt.plot(history['train_acc'], label='train accuracy')
# plt.plot(history['val_acc'], label='validation accuracy')
# plt.title('Training history')
# plt.ylabel('Accuracy')
# plt.xlabel('Epoch')
# plt.legend()
# plt.ylim([0, 1])
# plt.grid()
```

```
In [ ]: torch.cuda.empty_cache()
```

Load Models

```
In [ ]: import os

# # Load feature 1
# model_load_path = './model2_task1_feature1_model(unfrozen)'
# model = BERTClass()
# model.load_state_dict(torch.load(os.path.join(model_load_path, 'model_state.bin')))
# model = model.to(device)

# Load feature 2
model_load_path = './model2_task1_feature2_model(unfrozen)'
model = BERTClass()
model.load_state_dict(torch.load(os.path.join(model_load_path, 'model_state.bin')))
model = model.to(device)
```

```
In [ ]: test_acc, test_loss = eval_model(test_data_loader, model, optimizer)
```

```
In [ ]: test_acc
```

```
Out[ ]: 0.9839424141749723
```

```
In [ ]: from sklearn.metrics import confusion_matrix, classification_report

def get_predictions(model, data_loader):
    """
    Outputs:
        predictions -
    """
    model = model.eval()

    titles = []
    predictions = []
    prediction_probs = []
    target_values = []

    with torch.no_grad():
        for data in data_loader:
            title = data["title"]
            ids = data["input_ids"].to(device, dtype = torch.long)
            mask = data["attention_mask"].to(device, dtype = torch.long)
            token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
            targets = data["targets"].to(device, dtype = torch.float)

            outputs = model(ids, mask, token_type_ids)
            # add sigmoid, for the training sigmoid is in BCEWithLogitsLoss
            outputs = torch.sigmoid(outputs).detach().cpu()
```

```

# thresholding at 0.5
preds = outputs.round()
targets = targets.detach().cpu()

titles.extend(title)
predictions.extend(preds)
prediction_probs.extend(outputs)
target_values.extend(targets)

predictions = torch.stack(predictions)
prediction_probs = torch.stack(prediction_probs)
target_values = torch.stack(target_values)

return titles, predictions, prediction_probs, target_values

```

```
In [ ]: titles, predictions, prediction_probs, target_values = get_predictions(model, test_data_loaders)
```

```
In [ ]: predictions.numpy()
```

```
Out[ ]: array([[0., 0., 0., 0., 0., 0.],
   [0., 0., 0., 0., 0., 0.],
   [1., 0., 0., 0., 0., 0.],
   ...,
   [0., 0., 1., 0., 0., 0.],
   [0., 0., 0., 0., 0., 0.],
   [1., 0., 0., 0., 0., 0.]], dtype=float32)
```

Come here to change prediction variables

```
In [ ]: ### adjust and comment out the predictions depending on what feature you are using
#####
# task_1_transformer_f1 = predictions
#
# task_1_transformer_f2 = predictions
#####

```

```
In [ ]: print(f"titles:{len(titles)} \npredictions:{predictions.shape} \nprediction_probs:{prediction_probs.shape}\n")
titles:301
predictions:torch.Size([301, 6])
prediction_probs:torch.Size([301, 6])
target_values:torch.Size([301, 6])
```

```
In [ ]: print(classification_report(target_values, predictions, target_names=target_list))
```

	precision	recall	f1-score	support
ADR	0.96	0.97	0.97	108
WD	0.89	0.89	0.89	18
EF	0.96	0.91	0.93	55
INF	0.80	0.92	0.86	13
SSI	0.92	0.97	0.94	34
DI	1.00	0.89	0.94	27
micro avg	0.94	0.94	0.94	255
macro avg	0.92	0.93	0.92	255
weighted avg	0.95	0.94	0.94	255
samples avg	0.61	0.61	0.61	255

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in samples with no pr
edicted labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in samples with no true
labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in samples with no true
nor predicted labels. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In []: `print(df_test)`

					sentences	ADR	WD	EF	INF	\
2018	allow think one subject time previous	9,846	th...	0.0	0.0	1.0	0.0			
5762	since power chair almost nothing			0.0	0.0	0.0	0.0			
5715	dizziness standing bending			1.0	0.0	0.0	0.0			
1870	started taking 100mg went 50mg due	headaches	a...	1.0	0.0	0.0	0.0			
5142	3 weeks stopped working			0.0	0.0	0.0	1.0			
...
1894	sweats			1.0	0.0	0.0	0.0			
2240	zoloft made difference life	'm longer depressed		0.0	0.0	1.0	0.0			
2677	first week two bliss			0.0	0.0	1.0	0.0			
4411	currently take 40 mg cymbalta day			0.0	0.0	0.0	0.0			
191	'm serious pain right			1.0	0.0	0.0	0.0			
	SSI DI cluster					drug_sentences				
2018	1.0 1.0	1	zoloft allow think one subject time previous	9...						
5762	0.0 0.0	1	effexorxr since power chair almost nothing							
5715	0.0 0.0	1	effexorxr dizziness standing bending							
1870	0.0 0.0	3	zoloft started taking 100mg went 50mg due head...							
5142	0.0 0.0	3	effexorxr 3 weeks stopped working							
...
1894	0.0 0.0	1	zoloft	sweats						
2240	1.0 1.0	1	zoloft zoloft made difference life	'm longer d...						
2677	0.0 0.0	3	zoloft first week two bliss							
4411	0.0 0.0	3	cymbalta currently take 40 mg cymbalta day							
191	0.0 0.0	1	lexapro 'm serious pain right							

[301 rows x 9 columns]

In []: `from sklearn.metrics import multilabel_confusion_matrix
true_test_labels = df_test[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].to_numpy(dtype=np.float32)
print(true_test_labels)`

```
[[0. 0. 1. 0. 1. 1.  
[0. 0. 0. 0. 0.  
[1. 0. 0. 0. 0. 0.  
...  
[0. 0. 1. 0. 0. 0.  
[0. 0. 0. 0. 0. 0.  
[1. 0. 0. 0. 0. 0.]]]
```

In []: `confusion_matrixes=multilabel_confusion_matrix(true_test_labels,predictions.numpy())
class_names = ['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']

for idx, class_name in enumerate(class_names):
 cm = confusion_matrixes[idx]
 print(f"Confusion matrix for class '{class_name}':\n{cm}\n")
 report = classification_report(true_test_labels[:, idx], predictions[:, idx], target_name
 print(report)`

Confusion matrix for class 'ADR':

```
[[184  9]
 [ 1 107]]
```

	precision	recall	f1-score	support
ADR_0	0.99	0.95	0.97	193
ADR_1	0.92	0.99	0.96	108
accuracy			0.97	301
macro avg	0.96	0.97	0.96	301
weighted avg	0.97	0.97	0.97	301

Confusion matrix for class 'WD':

```
[[281  2]
 [ 3 15]]
```

	precision	recall	f1-score	support
WD_0	0.99	0.99	0.99	283
WD_1	0.88	0.83	0.86	18
accuracy			0.98	301
macro avg	0.94	0.91	0.92	301
weighted avg	0.98	0.98	0.98	301

Confusion matrix for class 'EF':

```
[[242  4]
 [ 5 50]]
```

	precision	recall	f1-score	support
EF_0	0.98	0.98	0.98	246
EF_1	0.93	0.91	0.92	55
accuracy			0.97	301
macro avg	0.95	0.95	0.95	301
weighted avg	0.97	0.97	0.97	301

Confusion matrix for class 'INF':

```
[[284  4]
 [ 1 12]]
```

	precision	recall	f1-score	support
INF_0	1.00	0.99	0.99	288
INF_1	0.75	0.92	0.83	13
accuracy			0.98	301
macro avg	0.87	0.95	0.91	301
weighted avg	0.99	0.98	0.98	301

Confusion matrix for class 'SSI':

```
[[265  2]
 [ 1 33]]
```

	precision	recall	f1-score	support
SSI_0	1.00	0.99	0.99	267
SSI_1	0.94	0.97	0.96	34
accuracy			0.99	301
macro avg	0.97	0.98	0.98	301

```

weighted avg      0.99      0.99      0.99      301

Confusion matrix for class 'DI':
[[274   0]
 [ 3  24]]

          precision    recall   f1-score   support
DI_0       0.99     1.00     0.99      274
DI_1       1.00     0.89     0.94      27

accuracy           0.99      301
macro avg       0.99     0.94     0.97      301
weighted avg     0.99     0.99     0.99      301

```

Statistical Tests

```

In [ ]: ## 4 Total ones
# task_1_transformer_f1, task_1_transformer_f2
# task_1_logistic_f1, task_1_logistic_f2

print(task_1_logistic_f1)
print(task_1_logistic_f2)
print(task_1_transformer_f1)
print(task_1_transformer_f2)

[[1 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 ...
 [1 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
[[1 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 ...
 [1 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 ...
 [1 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
tensor([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.]]))

tensor([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.]])
```



```

In [ ]: logistic_float_1 = task_1_logistic_f1.astype(float)
transformer_float_1 = task_1_transformer_f1.numpy().astype(float)
logistic_float_2 = task_1_logistic_f2.astype(float)
transformer_float_2 = task_1_transformer_f2.numpy().astype(float)
```

```
## multi_label_y is the ground truth
multi_label_y_float = multi_label_y.astype(float)
```

```
In [ ]: ## Hamming Loss
from sklearn.metrics import hamming_loss

def calculate_hamming_losses(logistic_f1, transformer_f1, logistic_f2, transformer_f2, y_float):
    combinations = [
        ('logistic_1 vs logistic_2', logistic_f1, logistic_f2),
        ('logistic_1 vs transformer_1', logistic_f1, transformer_f1),
        ('logistic_1 vs transformer_2', logistic_f1, transformer_f2),
        ('transformer_1 vs logistic_2', transformer_f1, logistic_f2),
        ('transformer_1 vs transformer_2', transformer_f1, transformer_f2),
        ('logistic_2 vs transformer_2', logistic_f2, transformer_f2),
        ('logistic_1 vs ground truth', logistic_f1, y_float),
        ('transformer_1 vs ground truth', transformer_f1, y_float),
        ('logistic_2 vs ground truth', logistic_f2, y_float),
        ('transformer_2 vs ground truth', transformer_f2, y_float)
    ]

    for name, array1, array2 in combinations:
        loss = hamming_loss(array1, array2)
        print(f"Hamming Loss for {name}: {loss}")

# Calculate and print Hamming Losses for all combinations
calculate_hamming_losses(logistic_float_1, transformer_float_1, logistic_float_2, transformer_float_2)
```

Hamming Loss for logistic_1 vs logistic_2: 0.014396456256921373
Hamming Loss for logistic_1 vs transformer_1: 0.10908084163898117
Hamming Loss for logistic_1 vs transformer_2: 0.10575858250276855
Hamming Loss for transformer_1 vs logistic_2: 0.10243632336655592
Hamming Loss for transformer_1 vs transformer_2: 0.014396456256921373
Hamming Loss for logistic_2 vs transformer_2: 0.10243632336655592
Hamming Loss for logistic_1 vs ground truth: 0.11517165005537099
Hamming Loss for transformer_1 vs ground truth: 0.016057585825027684
Hamming Loss for logistic_2 vs ground truth: 0.10963455149501661
Hamming Loss for transformer_2 vs ground truth: 0.01937984496124031

```
In [ ]: from scipy.stats import friedmanchisquare
from sklearn.preprocessing import LabelEncoder

def calculate_friedman_multi_label(logistic_f1, transformer_f1, logistic_f2, transformer_f2, n_labels = true_labels.shape[1] # Number of Labels
                                    ):
    results = []

    for i in range(n_labels):
        label_logistic_f1 = logistic_f1[:, i]
        label_transformer_f1 = transformer_f1[:, i]
        label_logistic_f2 = logistic_f2[:, i]
        label_transformer_f2 = transformer_f2[:, i]
        label_true = true_labels[:, i]

        stat, p = friedmanchisquare(label_logistic_f1, label_transformer_f1, label_logistic_f2, label_transformer_f2)
        results.append((i, stat, p))
        print(f"Friedman test for label {i}: statistic={stat}, p-value={p}")

    return results

calculate_friedman_multi_label(
    logistic_f1=logistic_float_1,
    transformer_f1=transformer_float_1,
    logistic_f2=logistic_float_2,
```

```

        transformer_f2=transformer_float_2,
        true_labels=multi_label_y_float
    )

Friedman test for label 0: statistic=16.372759856632985, p-value=0.0025576715535565652
Friedman test for label 1: statistic=10.666666666664245, p-value=0.030577016627630297
Friedman test for label 2: statistic=64.68148148148094, p-value=3.0031014251411415e-13
Friedman test for label 3: statistic=12.653061224503691, p-value=0.013101500388329173
Friedman test for label 4: statistic=34.711864406790355, p-value=5.323421579493195e-07
Friedman test for label 5: statistic=27.854545454554025, p-value=1.334881684275001e-05

Out[ ]: [(0, 16.372759856632985, 0.0025576715535565652),
           (1, 10.666666666664245, 0.030577016627630297),
           (2, 64.68148148148094, 3.0031014251411415e-13),
           (3, 12.653061224503691, 0.013101500388329173),
           (4, 34.711864406790355, 5.323421579493195e-07),
           (5, 27.854545454554025, 1.334881684275001e-05)]

```

Multi-Class

Logistic Regression

```

In [ ]: fileName=".\\ONLINE_FORA\\PsyTAR_dataset.xlsx"
data=pd.ExcelFile(fileName)
sheets={}
for sheet in data.sheet_names:
    sheets[sheet]=data.parse(sheet)

## Remove the first two sheets (License and read_me)
sheets.pop('License',None)
sheets.pop('read_me',None)

## This will print out the sheet names for the whole excel
for sheet in sheets.keys():
    print(f"Sheet Name: {sheet}")

df=data.parse('Sentence_Labeling')
df.drop(df.tail(1).index,inplace=True)
df['drug_id']=df['drug_id'].str.lower()
df['drug_name']=df['drug_id'].str.replace(r'\.\d+', '', regex=True)
df['sentences']=df['sentences'].apply(preprocess)
df.fillna(0,inplace=True)
unique_drug_count=df['drug_name'].nunique()

```

Sheet Name: Sample
Sheet Name: Sentence_Labeling
Sheet Name: ADR_Identified
Sheet Name: ADR_Mapped
Sheet Name: WD_Identified
Sheet Name: WD-Mapped
Sheet Name: SSI_Identified
Sheet Name: SSI_Mapped
Sheet Name: DI_Identified
Sheet Name: DI_Mapped

```

In [ ]: # # feature 1
# from sklearn.model_selection import train_test_split

# # this one has no 0s and 1s as extra features
# df['DI'] = pd.to_numeric(df['DI'], errors='coerce')

```

```

# df_train, df_temp = train_test_split(df, random_state=42, test_size=0.55, shuffle=True)

# df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)

# x_train = df_train['sentences'].values
# y_train = df_train['drug_name'].values

# x_val = df_valid['sentences'].values
# y_val = df_valid['drug_name'].values

# x_test = df_test['sentences'].values
# y_test = df_test['drug_name'].values

# tfidf_vectorizer = TfidfVectorizer()
# x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)
# x_val_tfidf = tfidf_vectorizer.transform(x_val)
# x_test_tfidf = tfidf_vectorizer.transform(x_test)

```

```

In [ ]: # # feature 2
# # this one has extra 0s and 1s as extra features
# sentences = df['sentences'].values
# df['DI'] = pd.to_numeric(df['DI'], errors='coerce')

# annotation_values = df[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values

# df_train, df_temp = train_test_split(df, random_state=42, test_size=0.55, shuffle=True)

# df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)

# sentences_train = df_train['sentences'].values
# annotation_values_train = df_train[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
# x_train = [
#     sentence + " " + ".join(map(str, annotation_value))
#     for sentence, annotation_value in zip(sentences_train, annotation_values_train)
# ]
# y_train = df_train['drug_name'].values

# sentences_val = df_valid['sentences'].values
# annotation_values_val = df_valid[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
# x_val = [
#     sentence + " " + ".join(map(str, annotation_value))
#     for sentence, annotation_value in zip(sentences_val, annotation_values_val)
# ]
# y_val = df_valid['drug_name'].values

# sentences_test = df_test['sentences'].values
# annotation_values_test = df_test[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
# x_test = [
#     sentence + " " + ".join(map(str, annotation_value))
#     for sentence, annotation_value in zip(sentences_test, annotation_values_test)
# ]
# y_test = df_test['drug_name'].values

# tfidf_vectorizer = TfidfVectorizer()
# x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)
# x_val_tfidf = tfidf_vectorizer.transform(x_val)
# x_test_tfidf = tfidf_vectorizer.transform(x_test)

```

```

In [ ]: # feature 3
# this one uses ONLY the annotated values
df['DI'] = pd.to_numeric(df['DI'], errors='coerce')

df_train, df_temp = train_test_split(df, random_state=42, test_size=0.55, shuffle=True)

```

```

df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True

x_train = df_train[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_train = df_train['drug_name'].values

x_val = df_valid[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_val = df_valid['drug_name'].values

x_test = df_test[['ADR', 'WD', 'EF', 'INF', 'SSI', 'DI']].values
y_test = df_test['drug_name'].values

x_train = np.nan_to_num(x_train, nan=0.0)
x_val = np.nan_to_num(x_val, nan=0.0)
x_test = np.nan_to_num(x_test, nan=0.0)

```

```

In [ ]: # param_grid = {
#         'C': [0.01, 0.1, 1, 10, 100, 1000],
#         'penalty': ['L1', 'L2'],
#         'solver': ['liblinear', 'saga', 'lbfgs'],
#     }

#feature1: Best parameters found: {'C': 0.1, 'penalty': 'L1', 'solver': 'saga'}
#feature2: Best parameters found: {'C': 0.1, 'penalty': 'L1', 'solver': 'saga'}
#feature3: Best parameters found: {'C': 0.1, 'penalty': 'L1', 'solver': 'saga'}

logistic_regression = LogisticRegression(C=0.1, penalty='L1', solver='saga') # Increased max_iter
logistic_regression.fit(x_train, y_train) ##### CHANGE HERE FOR DIFFERENT FEATURES (x_train)
# grid_search = GridSearchCV(estimator=logistic_regression_number, param_grid=param_grid, cv=5)

# grid_search.fit(x_train_number, y_train_number)

best_logistic_regression = logistic_regression

y_test_pred = best_logistic_regression.predict(x_test) ##### CHANGE HERE FOR DIFFERENT FEATURES (x_test)
test_accuracy_number = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_accuracy_number:.4f}')
print('Test Classification Report:')
print(classification_report(y_test, y_test_pred))

```

Test Accuracy: 0.3422
Test Classification Report:

	precision	recall	f1-score	support
cymbalta	0.32	0.87	0.47	84
effexorxr	0.45	0.29	0.36	82
lexapro	0.32	0.08	0.13	72
zoloft	0.00	0.00	0.00	63
accuracy			0.34	301
macro avg	0.27	0.31	0.24	301
weighted avg	0.29	0.34	0.26	301

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
In [ ]: y_train_series = pd.Series(y_train)
y_val_series = pd.Series(y_val)
y_test_series = pd.Series(y_test)

print("Distribution in training set:")
print(y_train_series.value_counts())
print("Distribution in validation set:")
print(y_val_series.value_counts())
print("Distribution in test set:")
print(y_test_series.value_counts())

Distribution in training set:
cymbalta    791
effexorxr   704
lexapro     644
zoloft      565
Name: count, dtype: int64
Distribution in validation set:
cymbalta    832
lexapro     776
effexorxr   763
zoloft      633
Name: count, dtype: int64
Distribution in test set:
cymbalta    84
effexorxr   82
lexapro     72
zoloft      63
Name: count, dtype: int64
```

Come here to change prediction variables

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

# tfidf_vectorizer = TfidfVectorizer()
# x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)
# x_val_tfidf = tfidf_vectorizer.transform(x_val)
# x_test_tfidf = tfidf_vectorizer.transform(x_test)

param_grid = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l2'],
    'solver': ['liblinear']
}

logistic_regression_model = LogisticRegression()
grid_search = GridSearchCV(logistic_regression_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
```

```

print("best param ", grid_search.best_params_)
print("cross_val accuracy ", grid_search.best_score_)

best_model = grid_search.best_estimator_
y_test_pred = best_model.predict(x_test)

### adjust and comment out the predictions depending on what feature you are using
#####
# task_2_logistic_f1 = y_test_pred

# task_2_logistic_f2 = y_test_pred

# task_2_logistic_f3 = y_test_pred
#####

print("test set")
print(classification_report(y_test, y_test_pred))
conf = confusion_matrix(y_test, y_test_pred)
print(conf)

plt.figure(figsize=(8, 6))
sns.heatmap(conf, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test), yticklabe
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

```

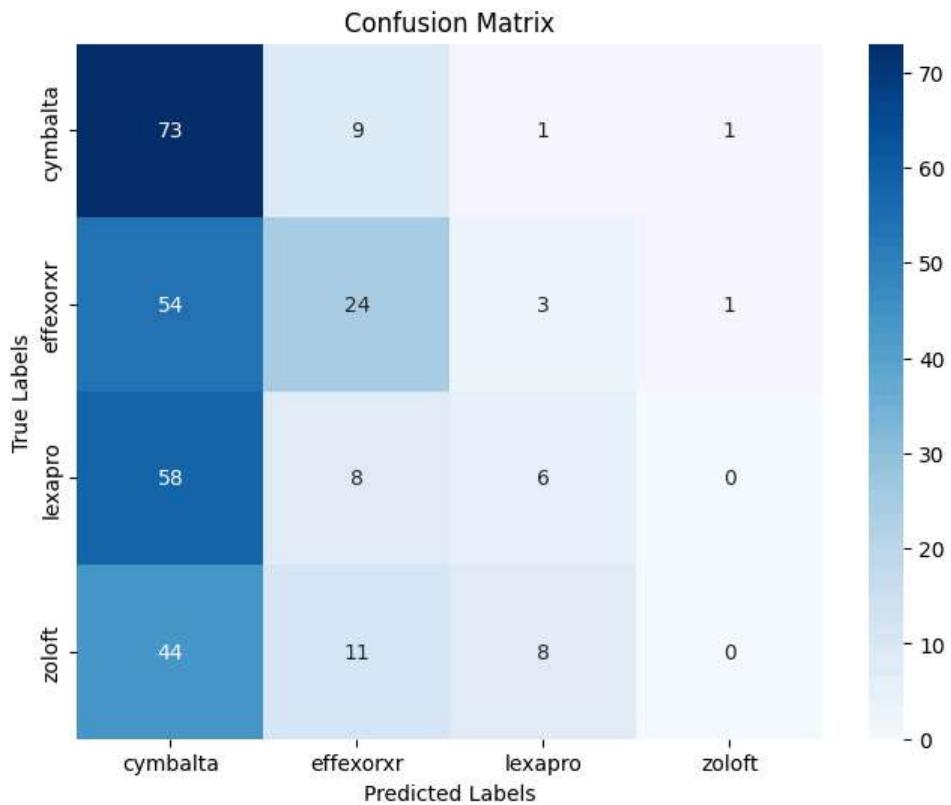
best param {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
cross_val accuracy 0.31286780310809886

	precision	recall	f1-score	support
cymbalta	0.32	0.87	0.47	84
effexorxr	0.46	0.29	0.36	82
lexapro	0.33	0.08	0.13	72
zoloft	0.00	0.00	0.00	63
accuracy			0.34	301
macro avg	0.28	0.31	0.24	301
weighted avg	0.29	0.34	0.26	301

```

[[73  9  1  1]
 [54 24  3  1]
 [58  8  6  0]
 [44 11  8  0]]

```



Transformers

```
In [ ]: def transform_labels(row):
    labels = [
        '[POS] adverse drug reaction' if row['ADR'] == 1 else '[NEG] adverse drug reaction',
        '[POS] withdrawal symptom' if row['WD'] == 1 else '[NEG] withdrawal symptom',
        '[POS] effective' if row['EF'] == 1 else '[NEG] effective',
        '[POS] ineffective' if row['INF'] == 1 else '[NEG] ineffective',
        '[POS] sign/symptom/illness' if row['SSI'] == 1 else '[NEG] sign/symptom/illness',
        '[POS] drug indication' if row['DI'] == 1 else '[NEG] drug indication'
    ]
    return ' '.join(labels)

# df['drug_name'] = df['drug_name'].astype('category').cat.codes
```



```
In [ ]: # #this one is for the combined sentences -- Task 2 Feature 1

# df_word_Label = df

# df_word_Label['transformed_sentences'] = df.apply(lambda row: f'{row["sentences"]}' {transform}
# #print(df_word_Label)

# ## ADR: adverse drug reaction
# ## WD: withdrawal symptom
# ## EF: effective
# ## INF: ineffective
# ## SSI: Sign/symptom/illness - if report contains explicit SSI that patient experienced that
# ## DI: drug indication - shows SSI that explicitly mentioned as being resolved because of di
# df_word_Label.drop(columns=['id', "comment_id", "drug_id", "sentence_index", "Findings", "oi
```

```
# df_word_Label = df_word_Label[['transformed_sentences', 'drug_name']]  
  
## Split data  
# df_train, df_temp = train_test_split(df_word_Label, random_state=42, test_size=0.55, shuffle=True)  
  
# df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)
```

```
In [ ]: ## This one is just for the text without transformed sentences -- Task 2 Feature 2  
# df_word_Label=df  
  
# x = df_word_Label['sentences'].values  
# y = df_word_Label['drug_name'].astype('category').cat.codes.values  
# df_word_Label.drop(columns=["id", "comment_id", "drug_id", "sentence_index", "Findings", "other"], axis=1)  
# df_word_Label = df_word_Label[['sentences', 'drug_name']]  
  
## Split data  
# df_train, df_temp = train_test_split(df_word_Label, random_state=42, test_size=0.55, shuffle=True)  
  
# df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)
```

```
In [ ]: ## This one is just for the annotated dataset -- Task 2 Feature 3  
  
df_word_Label = df  
  
df_word_Label['transformed_labels'] = df_word_Label.apply(transform_labels, axis=1)  
print(df_word_Label['transformed_labels'])  
  
df_word_Label.drop(columns=["id", "comment_id", "drug_id", "sentence_index", "Findings", "other"], axis=1)  
df_word_Label = df_word_Label[['transformed_labels', 'drug_name']]  
  
# x = df_word_Label['transformed_labels'].values  
# y = df_word_Label['drug_name'].astype('category').cat.codes.values  
  
#Split data  
df_train, df_temp = train_test_split(df_word_Label, random_state=42, test_size=0.55, shuffle=True)  
  
df_valid, df_test = train_test_split(df_temp, random_state=42, test_size=(1/11), shuffle=True)
```

```
0      [POS] adverse drug reaction [NEG] withdrawal s...  
1      [NEG] adverse drug reaction [NEG] withdrawal s...  
2      [NEG] adverse drug reaction [NEG] withdrawal s...  
3      [NEG] adverse drug reaction [NEG] withdrawal s...  
4      [NEG] adverse drug reaction [POS] withdrawal s...  
     ...  
6004      [NEG] adverse drug reaction [NEG] withdrawal s...  
6005      [NEG] adverse drug reaction [NEG] withdrawal s...  
6006      [NEG] adverse drug reaction [NEG] withdrawal s...  
6007      [NEG] adverse drug reaction [NEG] withdrawal s...  
6008      [NEG] adverse drug reaction [NEG] withdrawal s...  
Name: transformed_labels, Length: 6009, dtype: object
```

```
In [ ]: print(df_word_Label)
```

```

          transformed_labels  drug_name
0      [POS] adverse drug reaction [NEG] withdrawal s...    lexapro
1      [NEG] adverse drug reaction [NEG] withdrawal s...    lexapro
2      [NEG] adverse drug reaction [NEG] withdrawal s...    lexapro
3      [NEG] adverse drug reaction [NEG] withdrawal s...    lexapro
4      [NEG] adverse drug reaction [POS] withdrawal s...    lexapro
...
6004  [NEG] adverse drug reaction [NEG] withdrawal s... effexorxr
6005  [NEG] adverse drug reaction [NEG] withdrawal s... effexorxr
6006  [NEG] adverse drug reaction [NEG] withdrawal s... effexorxr
6007  [NEG] adverse drug reaction [NEG] withdrawal s... effexorxr
6008  [NEG] adverse drug reaction [NEG] withdrawal s... effexorxr

```

[6009 rows x 2 columns]

```
In [ ]: print(f"Train: {df_train.shape}, Test: {df_test.shape}, Valid: {df_valid.shape}")

Train: (2704, 2), Test: (301, 2), Valid: (3004, 2)
```

```
In [ ]: print(df['drug_name'].unique().tolist())

['lexapro', 'zoloft', 'cymbalta', 'effexorxr']
```

come here to change the label df[""]

```

In [ ]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
labels = {'lexapro':0,
          'zoloft':1,
          'cymbalta':2,
          'effexorxr':3
        }

class Dataset(torch.utils.data.Dataset):

    def __init__(self, df):
        self.labels = [labels[label] for label in df['drug_name']]
        self.texts = [tokenizer(text,
                               padding='max_length', max_length = 512, truncation=True,
                               return_tensors="pt") for text in df['transformed_labels']]
        ## change the df[''] depending on the feature
    def classes(self):
        return self.labels

    def __len__(self):
        return len(self.labels)

    def get_batch_labels(self, idx):
        return np.array(self.labels[idx])

    def get_batch_texts(self, idx):
        return self.texts[idx]

    def __getitem__(self, idx):

        batch_texts = self.get_batch_texts(idx)
        batch_y = self.get_batch_labels(idx)

        return batch_texts, batch_y

```

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\huggingface_hub\file_download.py:1132:  
FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads  
always resume when possible. If you want to force a new download, use `force_download=True`.  
warnings.warn(
```

```
In [ ]: from torch import nn  
  
class BertClassifier(nn.Module):  
  
    def __init__(self, dropout=0.5):  
  
        super(BertClassifier, self).__init__()  
  
        self.bert = BertModel.from_pretrained('bert-base-cased')  
        self.dropout = nn.Dropout(dropout)  
        self.linear = nn.Linear(768, 5)  
        self.relu = nn.ReLU()  
  
    def forward(self, input_id, mask):  
  
        _, pooled_output = self.bert(input_ids=input_id, attention_mask=mask, return_dict=False)  
        dropout_output = self.dropout(pooled_output)  
        linear_output = self.linear(dropout_output)  
        final_layer = self.relu(linear_output)  
  
        return final_layer
```

```
In [ ]: from torch.optim import Adam  
from tqdm import tqdm
```

```
In [ ]: history = defaultdict(list)  
torch.cuda.empty_cache()  
# def train(model, train_data, val_data, learning_rate, epochs, model_save_path):  
  
#     train, val = Dataset(train_data), Dataset(val_data)  
  
#     train_dataloader = torch.utils.data.DataLoader(train, batch_size=2, shuffle=True)  
#     val_dataloader = torch.utils.data.DataLoader(val, batch_size=2, shuffle=True)  
  
#     use_cuda = torch.cuda.is_available()  
#     device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')  
  
#     criterion = nn.CrossEntropyLoss()  
#     optimizer = Adam(model.parameters(), lr=learning_rate)  
  
#     if use_cuda:  
#         model = model.cuda()  
#         criterion = criterion.cuda()  
#     best_accuracy = 0  
  
#     start_time = time.time()  
  
#     for epoch_num in range(epochs):  
  
#         total_acc_train = 0  
#         total_loss_train = 0  
  
#         for train_input, train_label in tqdm(train_dataloader):  
  
#             train_label = train_label.to(device)  
#             mask = train_input['attention_mask'].to(device)  
#             input_id = train_input['input_ids'].squeeze(1).to(device)
```

```

#             output = model(input_id, mask)

#             batch_loss = criterion(output, train_label.Long())
#             total_loss_train += batch_loss.item()

#             acc = (output.argmax(dim=1) == train_label).sum().item()
#             total_acc_train += acc

#             model.zero_grad()
#             batch_loss.backward()
#             optimizer.step()

#             total_acc_val = 0
#             total_loss_val = 0

#             with torch.no_grad():

#                 for val_input, val_label in val_dataloader:

#                     val_label = val_label.to(device)
#                     mask = val_input['attention_mask'].to(device)
#                     input_id = val_input['input_ids'].squeeze(1).to(device)

#                     output = model(input_id, mask)

#                     batch_loss = criterion(output, val_label.Long())
#                     total_loss_val += batch_loss.item()

#                     acc = (output.argmax(dim=1) == val_label).sum().item()
#                     total_acc_val += acc

#                     train_acc = total_acc_train / len(train_data)
#                     train_loss = total_loss_train / len(train_data)
#                     val_acc = total_acc_val / len(val_data)
#                     val_loss = total_loss_val / len(val_data)

#                     history['train_acc'].append(train_acc)
#                     history['train_loss'].append(train_loss)
#                     history['val_acc'].append(val_acc)
#                     history['val_loss'].append(val_loss)

#                     if val_acc > best_accuracy:
#                         if not os.path.exists(model_save_path):
#                             os.makedirs(model_save_path)
#                         torch.save(model.state_dict(), os.path.join(model_save_path, 'model_state.bin'))
#                         print(f"Model saved to {model_save_path}")
#                         best_accuracy = val_acc

#                     print(f'Epochs: {epoch_num + 1} | Train Loss: {train_loss:.3f} | Train Accuracy: {train_acc:.3f} | Val Loss: {val_loss:.3f} | Val Accuracy: {val_acc:.3f}')

#                     end_time = time.time()
#                     total_train_time = end_time - start_time

#                     print(f"Total training time: {total_train_time:.2f} seconds")
#                     torch.cuda.empty_cache()

#             return history

EPOCHS = 15
model = BertClassifier()
LR = 1e-6
# model_save_path = './model2_task2_feature1_model(unfrozen)'

```

```
# train(model, df_train, df_valid, LR, EPOCHS, model_save_path)
# torch.cuda.empty_cache()
```

```
In [ ]: import matplotlib.pyplot as plt

# plt.rcParams["figure.figsize"] = (10,7)
# plt.plot(history['train_acc'], label='train accuracy')
# plt.plot(history['val_acc'], label='validation accuracy')
# plt.title('Training history')
# plt.ylabel('Accuracy')
# plt.xlabel('Epoch')
# plt.legend()
# plt.ylim([0, 1])
# plt.grid()
```

Load models

```
In [ ]: # feature 1 model
# model_load_path = './model2_task2_feature1_model(unfrozen)'
# model_save_path = model_load_path
# model = BertClassifier()
# model.load_state_dict(torch.load(os.path.join(model_load_path, 'model_state.bin')))
# model = model.to(device)

# # # feature 2 model
# model_load_path = './model2_task2_feature2_model(unfrozen)'
# model_save_path = model_load_path
# model = BertClassifier()
# model.load_state_dict(torch.load(os.path.join(model_load_path, 'model_state.bin')))
# model = model.to(device)

# feature 3 model
model_load_path = './model2_task2_feature3_model(unfrozen)'
model_save_path = model_load_path
model = BertClassifier()
model.load_state_dict(torch.load(os.path.join(model_load_path, 'model_state.bin')))
model = model.to(device)
```

```
In [ ]: test_data_loader = torch.utils.data.DataLoader(df_test, batch_size=8, shuffle=False)
```

```
In [ ]: torch.cuda.empty_cache()
def evaluate(model, test_data):
    test = Dataset(test_data)
    test_dataloader = torch.utils.data.DataLoader(test, batch_size=2)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    if use_cuda:
        model = model.cuda()

    criterion = nn.CrossEntropyLoss()
    total_acc_test = 0
    total_loss_test = 0
    model.eval()

    with torch.no_grad():
        for test_input, test_label in tqdm(test_dataloader, desc="Evaluating"):
            test_label = test_label.to(device, dtype=torch.long)
            mask = test_input['attention_mask'].to(device)
            input_id = test_input['input_ids'].squeeze(1).to(device)
```

```

        output = model(input_id, mask)

        loss = criterion(output, test_label)
        total_loss_test += loss.item()

        acc = (output.argmax(dim=1) == test_label).sum().item()
        total_acc_test += acc

        avg_loss_test = total_loss_test / len(test_dataloader)
        avg_acc_test = total_acc_test / len(test_data)

        print(f'Test Loss: {avg_loss_test:.3f} | Test Accuracy: {avg_acc_test:.3f}')
evaluate(model, df_test)

# Labels = {'Lexapro':0,
#           'zoloft':1,
#           'cymbalta':2,
#           'effexorxr':3
#           }

```

Evaluating: 100%|██████████| 151/151 [00:04<00:00, 30.95it/s]
Test Loss: 1.377 | Test Accuracy: 0.322

```

In [ ]: from torch.utils.data import DataLoader

def get_prediction(model, dataloader):
    use_cuda = torch.cuda.is_available()
    device = torch.device('cuda') if use_cuda else torch.device('cpu')

    if use_cuda:
        model = model.cuda()

    model.eval()

    predictions = []
    true_labels = []

    with torch.no_grad():
        for data_input, data_label in tqdm(dataloader):
            data_label = data_label.to(device)
            mask = data_input['attention_mask'].squeeze(1).to(device)
            input_id = data_input['input_ids'].squeeze(1).to(device)

            output = model(input_id, mask)
            preds = output.argmax(dim=1)

            predictions.extend(preds.cpu().numpy())
            true_labels.extend(data_label.cpu().numpy())

    # Convert Lists to numpy arrays
    predictions = np.array(predictions)
    true_labels = np.array(true_labels)

    return predictions, true_labels

test_dataset = Dataset(df_test)
test_dataloader = DataLoader(test_dataset, batch_size=2, shuffle=False)

model.load_state_dict(torch.load(os.path.join(model_save_path, 'model_state.bin')))

predictions, true_labels = get_prediction(model, test_dataloader)

from sklearn.metrics import classification_report

```

```
print(classification_report(true_labels, predictions))
```

```
100%|██████████| 151/151 [00:04<00:00, 32.88it/s]
      precision    recall   f1-score   support
      0          0.27     0.64     0.38      72
      1          0.00     0.00     0.00      63
      2          0.33     0.45     0.38      84
      3          0.72     0.16     0.26      82
      accuracy           0.32      301
macro avg       0.33     0.31     0.26      301
weighted avg    0.35     0.32     0.27      301
```

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Come here to change prediction variables

```
In [ ]: # labels = {'Lexapro':0,
#                 'zoloft':1,
#                 'cymbalta':2,
#                 'effexorxr':3
#                 }

reverse_labels = {v: k for k, v in labels.items()}

vectorized_reverse_labels = np.vectorize(reverse_labels.get)
label_predictions = np.array(vectorized_reverse_labels(predictions))

print(label_predictions)

### adjust and comment out the predictions depending on what feature you are using
#####
# task_2_transformer_f1 = label_predictions

# task_2_transformer_f2 = label_predictions

# task_2_transformer_f3 = label_predictions
#####
```

```
In [ ]: true_numpy_labels=np.array(df_test[ 'drug_name' ].to_numpy())
print(true_numpy_labels)
```

```

['zoloft' 'effexorxr' 'effexorxr' 'zoloft' 'effexorxr' 'zoloft'
 'effexorxr' 'effexorxr' 'effexorxr' 'lexapro' 'lexapro' 'lexapro'
 'lexapro' 'effexorxr' 'cymbalta' 'effexorxr' 'cymbalta' 'cymbalta'
 'cymbalta' 'zoloft' 'cymbalta' 'zoloft' 'cymbalta' 'cymbalta' 'lexapro'
 'effexorxr' 'zoloft' 'cymbalta' 'cymbalta' 'cymbalta' 'cymbalta'
 'cymbalta' 'zoloft' 'cymbalta' 'cymbalta' 'cymbalta' 'cymbalta']

```

```

In [ ]: from sklearn.metrics import multilabel_confusion_matrix
from sklearn.preprocessing import MultiLabelBinarizer
text_labels = ["zoloft", "effexorxr", "lexapro", "cymbalta"]

mlb = MultiLabelBinarizer(classes=text_labels)
true_binary_labels = mlb.fit_transform([[label] for label in true_numpy_labels])
pred_binary_labels = mlb.transform([[label] for label in label_predictions])

# Compute multilabel confusion matrix for all classes
confusion_matrices = multilabel_confusion_matrix(true_binary_labels, pred_binary_labels)
# print(confusion_matrices)

# Iterate over each class
for idx, class_name in enumerate(labels):
    # Get the confusion matrix for the current class

```

```

cm = confusion_matrices[idx]
print(f"Confusion matrix for class '{class_name}':\n{cm}\n")

# Generate classification report for the current class
report = classification_report(true_binary_labels[:, idx], pred_binary_labels[:, idx],
                                 target_names=[f'{class_name}_0', f'{class_name}_1'])
print(report)

```

Confusion matrix for class 'lexapro':

[[238 0]
[63 0]]

	precision	recall	f1-score	support
lexapro_0	0.79	1.00	0.88	238
lexapro_1	0.00	0.00	0.00	63
accuracy			0.79	301
macro avg	0.40	0.50	0.44	301
weighted avg	0.63	0.79	0.70	301

Confusion matrix for class 'zoloft':

[[214 5]
[69 13]]

	precision	recall	f1-score	support
zoloft_0	0.76	0.98	0.85	219
zoloft_1	0.72	0.16	0.26	82
accuracy			0.75	301
macro avg	0.74	0.57	0.56	301
weighted avg	0.75	0.75	0.69	301

Confusion matrix for class 'cymbalta':

[[107 122]
[26 46]]

	precision	recall	f1-score	support
cymbalta_0	0.80	0.47	0.59	229
cymbalta_1	0.27	0.64	0.38	72
accuracy			0.51	301
macro avg	0.54	0.55	0.49	301
weighted avg	0.68	0.51	0.54	301

Confusion matrix for class 'effexorxr':

[[140 77]
[46 38]]

	precision	recall	f1-score	support
effexorxr_0	0.75	0.65	0.69	217
effexorxr_1	0.33	0.45	0.38	84
accuracy			0.59	301
macro avg	0.54	0.55	0.54	301
weighted avg	0.63	0.59	0.61	301

```
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\julev\anaconda3\envs\cudaEnv\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

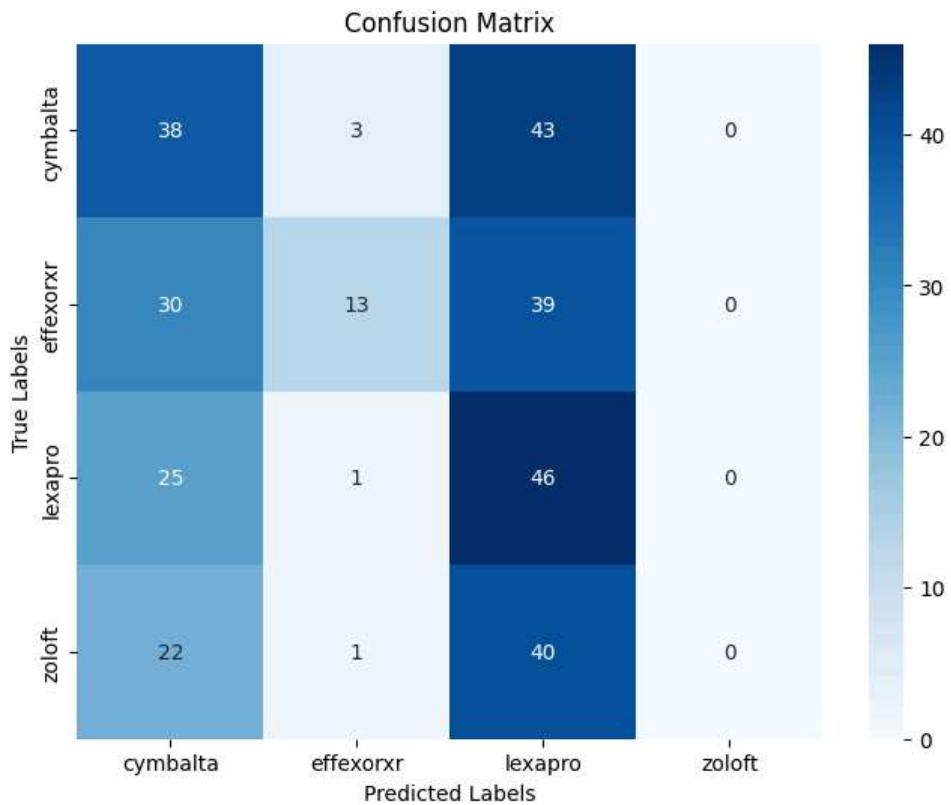
```
In [ ]: from sklearn.metrics import confusion_matrix
all_confusion_matrix=confusion_matrix(np.array(true_numpy_labels), np.array(label_predictions))
```

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

print(all_confusion_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(all_confusion_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(true_numpy_labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
[[38  3 43  0]
 [30 13 39  0]
 [25  1 46  0]
 [22  1 40  0]]
```



Statistical Tests

```
In [ ]: ## 6 Total ones
# task_2_transformer_f1, task_2_transformer_f2, task_2_transformer_f3
# task_2_logistic_f1, task_2_logistic_f2, task_2_logistic_f3

print(true_numpy_labels)

# print(task_2_logistic_f1)
# print(task_2_transformer_f1)
# print(task_2_logistic_f2)
print(task_2_transformer_f2)
# print(task_2_logistic_f3)
# print(task_2_transformer_f3)
```



```
In [ ]: from sklearn.metrics import cohen_kappa_score

def calculate_cohen_kappa_scores(logistic_f1, transformer_f1, logistic_f2, transformer_f2, lo:
    combinations = [
        ('logistic_1 vs logistic_2', logistic_f1, logistic_f2),
        ('logistic_1 vs logistic_3', logistic_f1, logistic_f3),
        ('logistic_2 vs logistic_3', logistic_f2, logistic_f3),
        ('transformer_1 vs transformer_2', transformer_f1, transformer_f2),
        ('transformer_1 vs transformer_3', transformer_f1, transformer_f3),
        ('transformer_2 vs transformer_3', transformer_f2, transformer_f3),
        ('logistic_1 vs transformer_1', logistic_f1, transformer_f1),
        ('logistic_1 vs transformer_2', logistic_f1, transformer_f2),
        ('logistic_1 vs transformer_3', logistic_f1, transformer_f3),
        ('logistic_2 vs transformer_1', logistic_f2, transformer_f1),
        ('logistic_2 vs transformer_2', logistic_f2, transformer_f2),
        ('logistic_2 vs transformer_3', logistic_f2, transformer_f3),
        ('logistic_3 vs transformer_1', logistic_f3, transformer_f1),
        ('logistic_3 vs transformer_2', logistic_f3, transformer_f2),
        ('logistic_3 vs transformer_3', logistic_f3, transformer_f3),
        ('logistic_1 vs ground truth', logistic_f1, true_labels),
        ('logistic_2 vs ground truth', logistic_f2, true_labels),
        ('logistic_3 vs ground truth', logistic_f3, true_labels),
        ('transformer_1 vs ground truth', transformer_f1, true_labels),
        ('transformer_2 vs ground truth', transformer_f2, true_labels),
        ('transformer_3 vs ground truth', transformer_f3, true_labels)
    ]
    for name, array1, array2 in combinations:
        score = cohen_kappa_score(array1, array2)
        print(f"Cohen's Kappa Score for {name}: {score}")

calculate_cohen_kappa_scores(
    logistic_f1=task_2_logistic_f1,
    transformer_f1=task_2_transformer_f1,
```

```

        logistic_f2=task_2_logistic_f2,
        transformer_f2=task_2_transformer_f2,
        logistic_f3=task_2_logistic_f3,
        transformer_f3=task_2_transformer_f3,
        true_labels=true_numpy_labels
    )

Cohen's Kappa Score for logistic_1 vs logistic_2: 0.9907771973097605
Cohen's Kappa Score for logistic_1 vs logistic_3: 0.0921565053113036
Cohen's Kappa Score for logistic_2 vs logistic_3: 0.08796989977691205
Cohen's Kappa Score for transformer_1 vs transformer_2: -0.011594689968072558
Cohen's Kappa Score for transformer_1 vs transformer_3: 0.3870595919621932
Cohen's Kappa Score for transformer_2 vs transformer_3: -0.04598585262001231
Cohen's Kappa Score for logistic_1 vs transformer_1: 0.018608349900596455
Cohen's Kappa Score for logistic_1 vs transformer_2: 0.0029711978618749413
Cohen's Kappa Score for logistic_1 vs transformer_3: 0.06310512532734758
Cohen's Kappa Score for logistic_2 vs transformer_1: 0.01525698744035457
Cohen's Kappa Score for logistic_2 vs transformer_2: 0.005995237825273558
Cohen's Kappa Score for logistic_2 vs transformer_3: 0.0612367300425839
Cohen's Kappa Score for logistic_3 vs transformer_1: 0.33626538722496857
Cohen's Kappa Score for logistic_3 vs transformer_2: -0.005325452894368254
Cohen's Kappa Score for logistic_3 vs transformer_3: 0.25131827678837926
Cohen's Kappa Score for logistic_1 vs ground truth: 0.1927168698036178
Cohen's Kappa Score for logistic_2 vs ground truth: 0.19223544145272387
Cohen's Kappa Score for logistic_3 vs ground truth: 0.09258667153884803
Cohen's Kappa Score for transformer_1 vs ground truth: 0.05221845574387951
Cohen's Kappa Score for transformer_2 vs ground truth: -0.03699950783127304
Cohen's Kappa Score for transformer_3 vs ground truth: 0.08854220784040145

```

```
In [ ]: from sklearn.metrics import matthews_corrcoef
matthews_corrcoef(true_numpy_labels,task_2_transformer_f1)
```

```
Out[ ]: 0.06650903780616059
```

```
In [ ]: from sklearn.preprocessing import LabelEncoder
all_words = task_2_logistic_f1 + task_2_transformer_f1 + task_2_logistic_f2 + task_2_transformer_f2
model = LabelEncoder()
model.fit(all_words)

def words_to_numeric(words, encoder):
    return encoder.transform(words)

logistic_f1_vectors = words_to_numeric(task_2_logistic_f1, label_encoder)
transformer_f1_vectors = words_to_numeric(task_2_transformer_f1, label_encoder)
logistic_f2_vectors = words_to_numeric(task_2_logistic_f2, label_encoder)
transformer_f2_vectors = words_to_numeric(task_2_transformer_f2, label_encoder)
logistic_f3_vectors = words_to_numeric(task_2_logistic_f3, label_encoder)
transformer_f3_vectors = words_to_numeric(task_2_transformer_f3, label_encoder)
true_label_vectors = words_to_numeric(true_numpy_labels, label_encoder)
```

```
In [ ]: def calculate_mcc_scores(logistic_f1, transformer_f1, logistic_f2, transformer_f2, logistic_f3):
combinations = [
    ('logistic_1 vs logistic_2', logistic_f1, logistic_f2),
    ('logistic_1 vs logistic_3', logistic_f1, logistic_f3),
    ('logistic_2 vs logistic_3', logistic_f2, logistic_f3),
    ('transformer_1 vs transformer_2', transformer_f1, transformer_f2),
    ('transformer_1 vs transformer_3', transformer_f1, transformer_f3),
```

```

        ('transformer_2 vs transformer_3', transformer_f2, transformer_f3),
        ('logistic_1 vs transformer_1', logistic_f1, transformer_f1),
        ('logistic_1 vs transformer_2', logistic_f1, transformer_f2),
        ('logistic_1 vs transformer_3', logistic_f1, transformer_f3),
        ('logistic_2 vs transformer_1', logistic_f2, transformer_f1),
        ('logistic_2 vs transformer_2', logistic_f2, transformer_f2),
        ('logistic_2 vs transformer_3', logistic_f2, transformer_f3),
        ('logistic_3 vs transformer_1', logistic_f3, transformer_f1),
        ('logistic_3 vs transformer_2', logistic_f3, transformer_f2),
        ('logistic_3 vs transformer_3', logistic_f3, transformer_f3),
        ('logistic_1 vs ground truth', logistic_f1, true_labels),
        ('logistic_2 vs ground truth', logistic_f2, true_labels),
        ('logistic_3 vs ground truth', logistic_f3, true_labels),
        ('transformer_1 vs ground truth', transformer_f1, true_labels),
        ('transformer_2 vs ground truth', transformer_f2, true_labels),
        ('transformer_3 vs ground truth', transformer_f3, true_labels)
    ]

    for name, array1, array2 in combinations:
        stat= matthews_corrcoef(array1, array2)
        print(f"Matthews Correlation Coefficient for {name}: {stat}")

    calculate_mcc_scores(
        logistic_f1=logistic_f1_vectors,
        transformer_f1=transformer_f1_vectors,
        logistic_f2=logistic_f2_vectors,
        transformer_f2=transformer_f2_vectors,
        logistic_f3=logistic_f3_vectors,
        transformer_f3=transformer_f3_vectors,
        true_labels=true_label_vectors
    )

```

Matthews Correlation Coefficient for logistic_1 vs logistic_2: 0.9908243011149059
 Matthews Correlation Coefficient for logistic_1 vs logistic_3: 0.11603360836444948
 Matthews Correlation Coefficient for logistic_2 vs logistic_3: 0.11044995801961668
 Matthews Correlation Coefficient for transformer_1 vs transformer_2: -0.01661564829025215
 Matthews Correlation Coefficient for transformer_1 vs transformer_3: 0.4551360657179037
 Matthews Correlation Coefficient for transformer_2 vs transformer_3: -0.06644812326256089
 Matthews Correlation Coefficient for logistic_1 vs transformer_1: 0.02283953478105437
 Matthews Correlation Coefficient for logistic_1 vs transformer_2: 0.0035845582243071495
 Matthews Correlation Coefficient for logistic_1 vs transformer_3: 0.0746322866284426
 Matthews Correlation Coefficient for logistic_2 vs transformer_1: 0.018695243554601594
 Matthews Correlation Coefficient for logistic_2 vs transformer_2: 0.007267776505058765
 Matthews Correlation Coefficient for logistic_2 vs transformer_3: 0.07240702830017491
 Matthews Correlation Coefficient for logistic_3 vs transformer_1: 0.3695835673178944
 Matthews Correlation Coefficient for logistic_3 vs transformer_2: -0.008096198075814411
 Matthews Correlation Coefficient for logistic_3 vs transformer_3: 0.3659514030761309
 Matthews Correlation Coefficient for logistic_1 vs ground truth: 0.19444590491905536
 Matthews Correlation Coefficient for logistic_2 vs ground truth: 0.19418937045783918
 Matthews Correlation Coefficient for logistic_3 vs ground truth: 0.12472589271449688
 Matthews Correlation Coefficient for transformer_1 vs ground truth: 0.06650903780616059
 Matthews Correlation Coefficient for transformer_2 vs ground truth: -0.043092423288888446
 Matthews Correlation Coefficient for transformer_3 vs ground truth: 0.10377298511197906

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

## Cohen's Kappa Graph

data = {
    'Task 2 Feature 1 Transformer': [None, 0.01860835, -0.01159469, 0.015256699, 0.387059592,
    'Task 2 Feature 1 Logistic Regression': [0.01860835, None, 0.002971198, 0.990777197, 0.06,
    'Task 2 Feature 2 Transformer': [-0.01159469, 0.002971198, None, 0.005995238, -0.04598585,
    'Task 2 Feature 2 Logistic Regression': [0.015256699, 0.990777197, 0.005995238, None, 0.0,
    'Task 2 Feature 3 Transformer': [0.387059592, 0.063105125, -0.045985853, 0.06123673, None,
    'Task 2 Feature 3 Logistic Regression': [0.015256699, 0.092156505, 0.005995238, 0.0879699,
    'True Labels': [0.052218456, 0.19271687, -0.036999508, 0.192235441, 0.088542208, 0.192235
}

df = pd.DataFrame(data, index=[
    'Task 2 Feature 1 Transformer',
    'Task 2 Feature 1 Logistic Regression',
    'Task 2 Feature 2 Transformer',
    'Task 2 Feature 2 Logistic Regression',
    'Task 2 Feature 3 Transformer',
    'Task 2 Feature 3 Logistic Regression',
    'True Labels'
])

plt.figure(figsize=(10, 8))
sns.heatmap(df, annot=True, fmt=".4f", cmap="coolwarm", cbar=True, linewidths=.5)

plt.title("Cohen's Kappa Heatmap")
plt.show()
```

