Challenge: LLM for Insurance Claim Automation

Name: Nitesh Jindal

1. Introduction

The transformer-based models such as BERT, GPT, T5, BART, and XLNet are trained on extensive text data, giving them a strong understanding of language. By considering the surrounding words, they can capture the context and meaning of words in a claim, leading to accurate claim categorizations. Further, we can adjust weights and parameters which allow these models to adapt to specific claims and categories, enhancing their performance in claim processing tasks. However, there are challenges when using large language models like BERT:

- Training these models requires significant computational resources and time, especially with substantial amounts of data.
- Effectiveness of these models vastly relies on high-quality and domain-specific training data. LLMs can inherit biases present in the data which can reinfornce existing bias and lead to unfair claim categorization.
- Obtaining labeled data for training can be labor-intensive, particularly in the insurance sector.
- Additionally, interpreting the results of these models can be complex due to their neural network architecture.

The approach to implement transformer-based models involves fine-tuning these models on an insurance domain-based dataset, with the option to freeze the weights of the pre-trained models. This allows the models to adapt and specialize to the specific characteristics and nuances of the insurance industry. Another approach involves utilizing the pre-trained transformer models for generating embeddings. These embeddings capture the contextualized representations of the claim texts and can be used as input for downstream tasks, such as claim categorization.

Ensuring compliance with privacy, security, and regulatory guidelines is of utmost importance when deploying models in sensitive domains. We can take steps to protect personally identifiable information (PII) by anonymizing it before training or inferencing with the model that helps safeguard the privacy and confidentiality of individuals involved in the insurance claims process. To provide interpretrability, we can leverage techniques such as Knowledge-Assisted Attention Maps. These maps utilize activation and saliency information to highlight important input features that influence the model's decision-making process. We can visualize the output predictions through guided backpropagation to understand the most influential parts of the input data with respect to the downstream task. These steps can also enhance the transparency and interpretability of our model.

2. Methodology

2.1 Model Architecture

We conducted three different experiments, outlined as follows:

(a) Pretrained BERT for Embeddings + XGBoost Classifier: We process the data in batches, with each batch containing a specified number of input tokens (tokenized encodings). The input tokens are converted into tensors and passed through the BERT model to obtain outputs for each token. We extracted the embedding of the first token in each sequence, representing the overall contextualized representation. These embeddings are collected separately for the training, validation and test data. We then trained an XGBoost classifier using the generated embeddings. We initialize XGBoost classifier with a 100 estimators and default learning rate. The training and validation sets, along with their corresponding labels, served as evaluation sets for the classifier. The training and validation errors are depicted in Section 3.2. Finally, the trained classifier was employed to predict labels for the test data.

(b) Pretrained BERT for Embeddings + Multi-Class Neural Network: Using the approach described in part (a), we generated text embeddings for the train, validation, and test datasets. These embeddings were then utilized as input for a neural network model, consisting of two fully connected (linear) layers with ReLU activation and a dropout layer. The model was optimized using the Adam optimizer with a learning rate of 0.001, and the loss function employed was CrossEntropyLoss. The training loop spanned 5 epochs, with the Training data processed in batches of 16 instances per epoch. For each batch, a forward pass was executed, followed by backward pass computation and parameter updates using the optimizer. During the evaluation stage, the model was switched to evaluation mode to generate predictions on the test data.

(c) Fine-tune BERT base uncased: We initialized the 'bert-base-uncased' BERT model and utilized the NVIDIA GeForce RTX 3070 Laptop GPU for efficient training. The batches of 16 training samples were processed to enhance computational efficiency. We used the AdamW optimizer and the cross-entropy loss function for optimization. Considering the size of the training loader, we conducted training for 2 epochs. During the training loop, the model operated in training mode, iteratively processing each batch. This involved conducting a forward pass, computing the loss, executing backpropagation, and optimizing the model's parameters. The training loss was accumulated per epoch. For validation, we switched the model to evaluation mode, and the validation loader underwent a similar iterative process. The loss was computed and accumulated and thus we calculated average training and validation losses. The training and validation losses enabled us to monitor the model's performance and make informed training decisions.

2.2 Dataset Description

We created an InsuranceClaimGenerator class to generate synthetic insurance claim data, which includes three claim types: Auto, Health, and Property. We manually created template files i.e., three incident reports for each claim type. The generate_insurance_claim() method randomly selects an incident class (Auto, Health, or Property) and generates various details, such as policy number, incident date, policy claim number, claim amount, and claim status.Then, the generate_incident_summary() method generates an incident summary by randomly selecting a template for the incident class and replacing placeholders in the template with random values. By using the InsuranceClaimGenerator class, we generated 30,000 insurance claims, which we then saved to a CSV file named "insurance_claims.csv". The class label distribution in our data is as follows:

Target Class	**Count**	**%**
property	10180	33.93
health	9946	33.15
auto	9874	32.91
Total	30000	

2.3 Data Preprocessing

To prepare the data for model consumption, we split the 30000 insurance claims data into training and test sets using an 80:20 ratio. We encoded the incident class labels to numeric values using a LabelEncoder. We then converted the text data to lists and tokenized using the BERT tokenizer with the 'bert-base-uncased' pre-trained model. The tokenization process includes padding, truncation, and conversion to PyTorch tensors. We then used the preprocessed input for downstream task of classifying the insurance claims. Particularly, when developing the fine-tune BERT model, we converted the tokenized inputs and labels into PyTorch datasets (train_ds, val_ds, and test_ds) and created data loaders (train_loader, val_loader, test_loader) to handle batch processing of the datasets.

3. Experiments and Findings

3.1 Model Performance Results for Claim Classification

We assessed the performance of our claim categorization models by analyzing various important metrics, including macro precision, macro recall, macro F1-score, weighted precision, weighted recall, and weighted F1-score. Weighted metrics consider the support values and provide an average measure across all classes. Additionally, we computed the F1-macro score was calculated as the average F1 score across classes, irrespective of class imbalances, with a focus on accurately identifying instances from minority classes. To ensure the robustness and reliability of our findings, we conducted individual experiments using five different bootstrap sample seeds. We present these performance estimates, accompanied by their corresponding confidence intervals, in Table 1.

Table 1: Performance Results of Claim Classification Models on Training, Validation, and Test Data

Evaluation Results on Train Data:

Model	M-Prec	M-Recall	M-F1	W-Prec	W-Recall	W-F1	Runtime
Pretrained_BERTEmbd+NN	0.955±0.004	0.952±0.006	0.952±0.006	0.954±0.003	0.953±0.005	0.952±0.006	29.579±0.326
Pretrained_BERTEmbd+XGB	0.966±0.001	0.966±0.001	**0.966±0.001**	0.966±0.001	0.966±0.001	0.966±0.001	66.885±2.182
BERT_base_uncased	0.957±0.002	0.953±0.004	0.953±0.003	0.957±0.003	0.953±0.003	0.953±0.003	176.357±17.977

Evaluation Results on Validation Data:

Model	M-Prec	M-Recall	M-F1	W-Prec	W-Recall	W-F1	Runtime
Pretrained_BERTEmbd+NN	0.957±0.005	0.954±0.006	0.955±0.006	0.957±0.005	0.954±0.005	0.955±0.006	29.579±0.326
Pretrained_BERTEmbd+XGB	0.953±0.005	0.953±0.005	**0.953±0.005**	0.953±0.005	0.953±0.005	0.953±0.005	66.885±2.182
BERT_base_uncased	0.955±0.006	0.951±0.01	0.95±0.01	0.956±0.006	0.95±0.009	0.95±0.01	176.357±17.977

Evaluation Results on Test Data:

Model	M-Prec	M-Recall	M-F1	W-Prec	W-Recall	W-F1	Runtime
Pretrained_BERTEmbd+NN	0.955±0.003	0.953±0.006	0.953±0.006	0.955±0.003	0.953±0.006	0.952±0.006	29.579±0.326
Pretrained_BERTEmbd+XGB	0.953±0.002	0.952±0.003	**0.953±0.003**	0.953±0.003	0.952±0.002	0.952±0.002	66.885±2.182
BERT_base_uncased	0.957±0.002	0.951±0.004	0.952±0.003	0.957±0.003	0.952±0.003	0.952±0.003	176.357±17.977

To assess the performance of our various model runs, we conducted a statistical analysis comparing them to the best performing model. This analysis aimed to determine whether there exists a statistically significant difference between the two classifiers. We employed a paired t-test on the macro F1 score estimates obtained from the test set. The Null Hypothesis states that there is no difference between the performances of the two models, while the Alternative Hypothesis suggests that a significant difference exists between their performances.

Table 2: Paired T-test

Model	t_stats	HYPOTHESIS_CHECK
BERT_base_uncased	2.557	Accept Null
Pretrained_BERTEmbd+NN	4.417	Reject Null
Pretrained_BERTEmbd+XGB	Nan	Nan

3.2 Training and Validation Error Plots

We visualized the evolution of classification error across 100 epochs for various samples as illustrated in Fig. 1. Our observations reveal that running the models for 100 epochs often results in overfitting, as indicated by the gradual increase in validation loss after approximately 25 to 30 epochs across different sample runs.

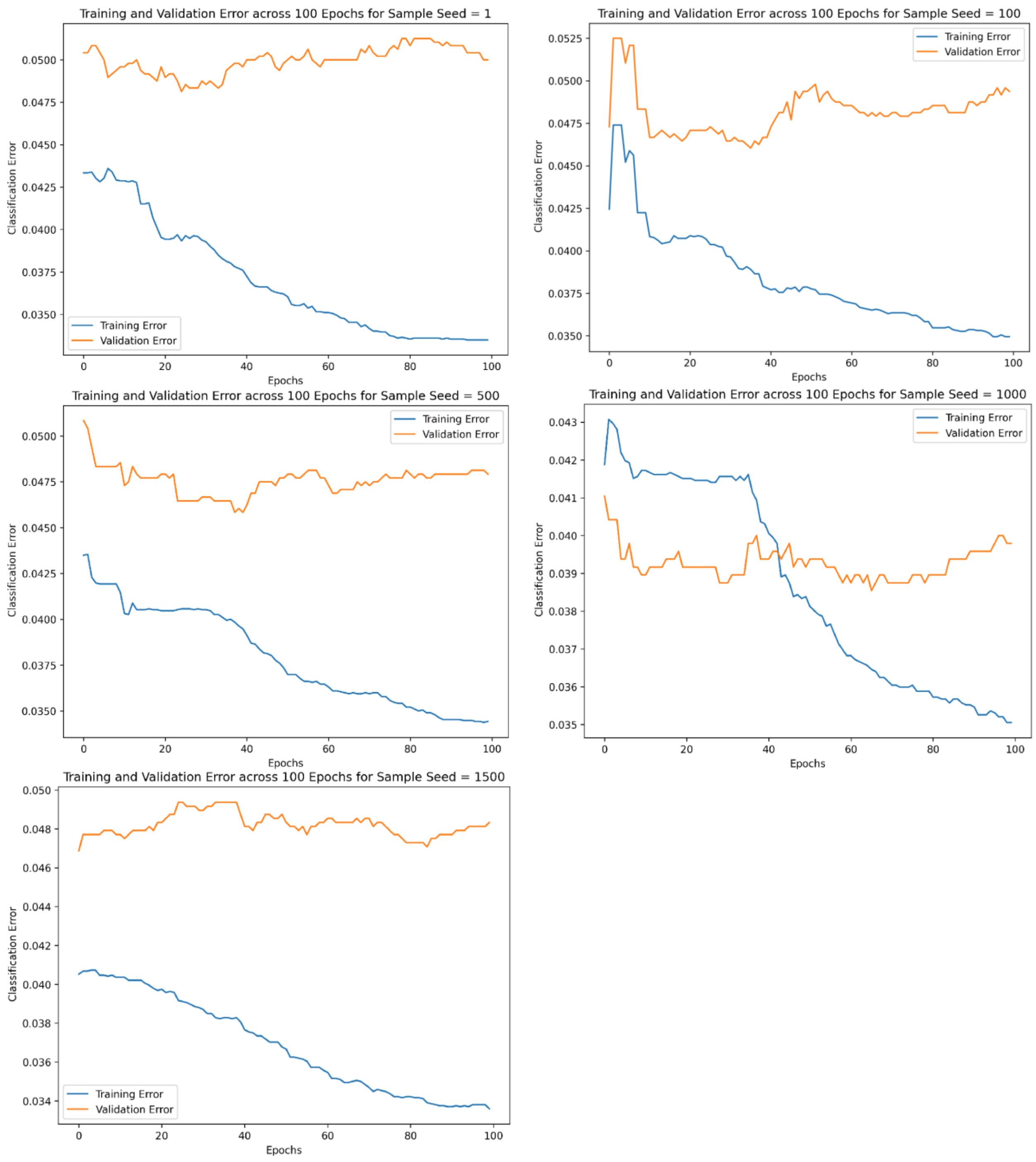

Fig. 1 Training and Validation Error Plots

3.3 Classification Report on Test Data

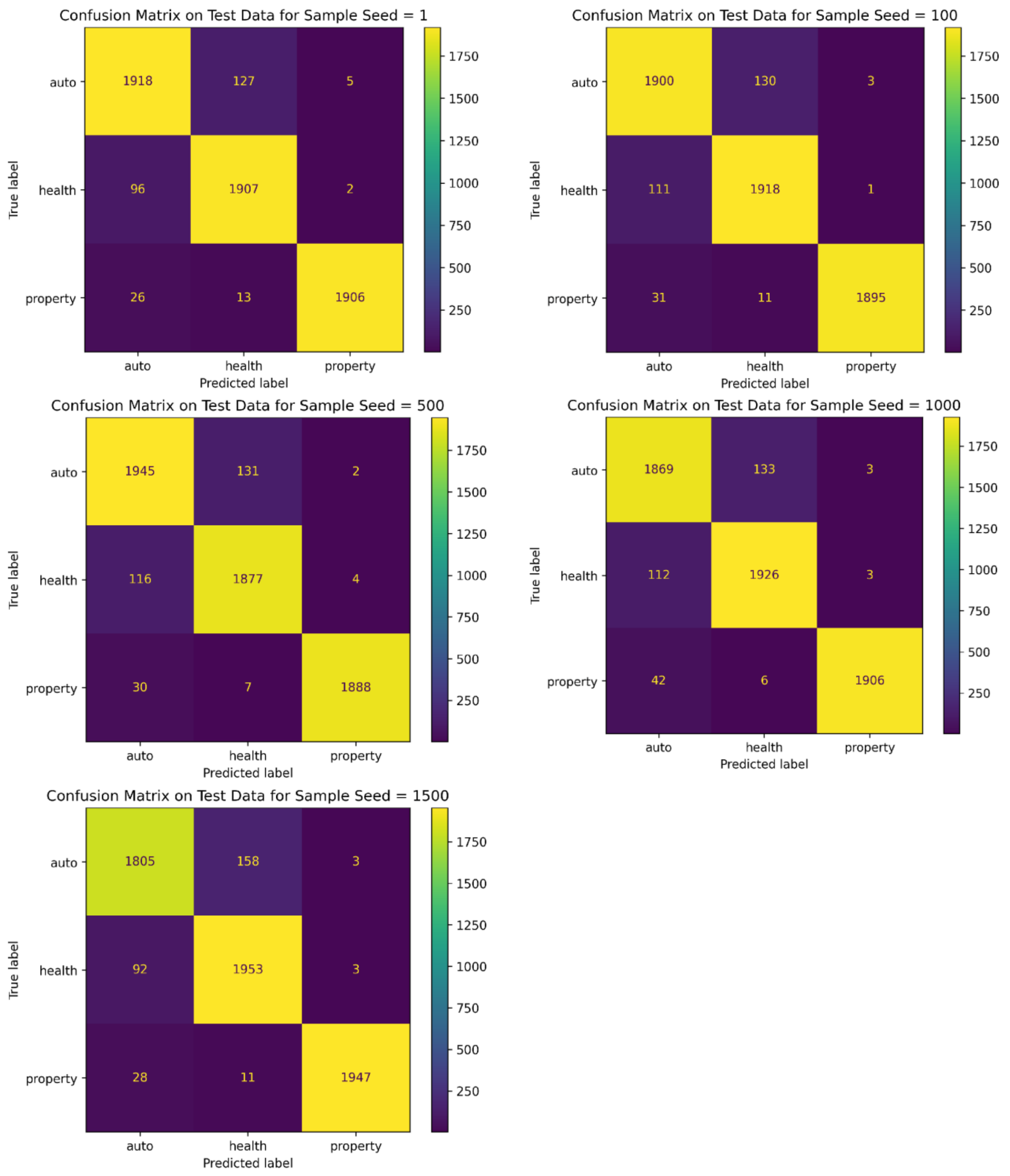

Fig. 2 Classification Results on Test Data