

ISM6930 Tech Foundations of AI

Contradictory, My Dear Watson

Exploring Multilingual Text Relationships

By – **Group5**
Akshit Ram Pershad
Sai Bharadwaja Vellanki

ABSTRACT:

This comprehensive report presents an in-depth analysis of the development and evaluation of Natural Language Inference (NLI) models for the "Contradictory, My Dear Watson" competition. It explores various model architectures, data preprocessing, training strategies, and performance evaluation. Our best-performing ensemble model achieved an accuracy of 90.06%.

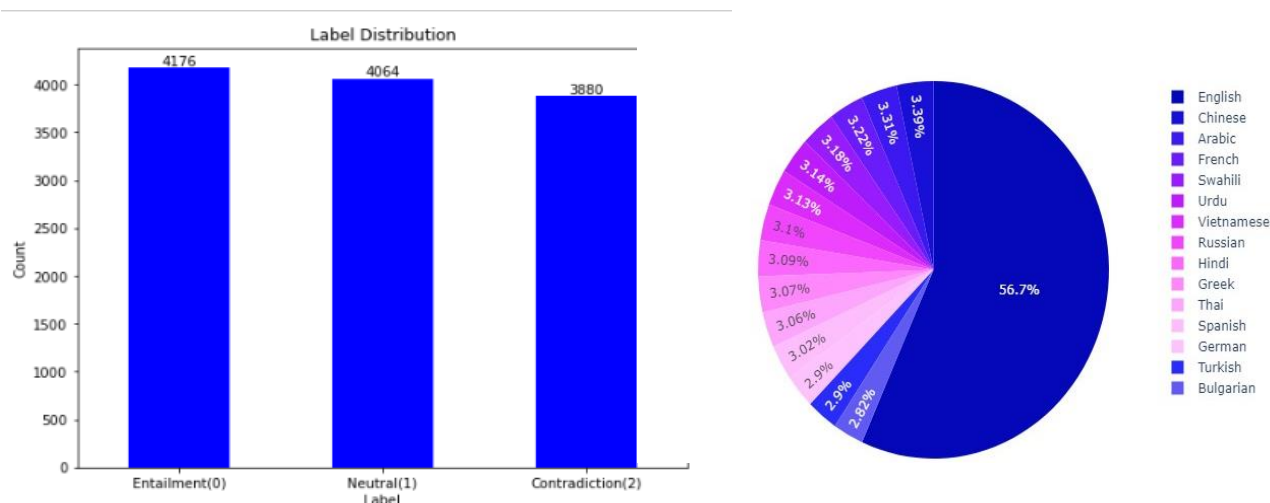
INTRODUCTION:

Natural Language Inference (NLI) is a fundamental NLP problem that involves determining the logical relationship between two sentences: entailment, contradiction, or neutrality. This report discusses the development of NLI models and their evaluation for the Kaggle competition, "Contradictory, My Dear Watson."

DATA EXPLORATION:

The dataset contains text in fifteen different languages, with the majority (56.7%) in English. Text length distribution analysis shows more than 1500 sentences with less than 100 characters in premises. The distribution of hypothesis text length reveals over 2000 sentences with a length of 50 characters. The label distribution indicates a balanced dataset with 'entailment,' 'neutral,' and 'contradiction' labels.

The key steps include Extracting the data from a DataFrame which consists of 'premise' and 'hypothesis' columns that are selected as the input features, and the 'label' column is chosen as the target labels. The 'premise' and 'hypothesis' columns contain the text premises and hypotheses for the NLI task, while the 'label' column contains the corresponding labels or categories for each example.



MODEL ARCHITECTURES AND TRAINING:

We explored different model architectures, each with its training process:

MODEL1: ROBERTA-BASED SEQUENCE CLASSIFICATION FOR NATURAL LANGUAGE INFERENCE

RoBERTa, short for "Robustly optimized BERT approach," is an NLP model introduced by Facebook AI in 2019. It's based on the BERT architecture but incorporates modifications for enhanced performance.

This model uses the RoBERTa architecture for Natural Language Inference (NLI). It involves data preprocessing, tokenization, and encoding, followed by training with early stopping. Achieved a test accuracy of approximately 63%.

Evaluating the Model on the Test Set:

After training, the model is evaluated on the test dataset. The model achieved a test accuracy of approximately 63% and a test loss of approximately 0.783 after training. These metrics are used to assess the model's performance in making predictions for the NLI task.

```
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

76/76 [=====] - 6s 74ms/step - loss: 0.7830 - accuracy: 0.6304
Test Loss: 0.7829666137695312
Test Accuracy: 0.6303630471229553
```

MODEL2: MULTILINGUAL BERT (MBERT) SEQUENCE CLASSIFICATION FOR NATURAL LANGUAGE INFERENCE WITH OPTIMIZATION TECHNIQUES

mBERT stands for Multilingual BERT. It's another variant of the BERT (Bidirectional Encoder Representations from Transformers) model, designed to handle multiple languages. mBERT is pre-trained on a diverse range of languages, making it a multilingual model that can understand and generate representations for text in various languages.

The idea behind mBERT is to create a model that can perform well across different languages without the need for language-specific models. It's a single model that can handle tasks in multiple languages, making it a versatile choice for multilingual natural language processing (NLP) applications.

Evaluating the Model on the Test Set:

```
: # Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

76/76 [=====] - 6s 75ms/step - loss: 0.8204 - accuracy: 0.6572
Test Loss: 0.8204382061958313
Test Accuracy: 0.6571782231330872
```

The reported results indicate that, on the test set, the model achieved a test accuracy of approximately 65.7% and a test loss of approximately 0.820 after training.

MODEL3: XLM-ROBERTA SEQUENCE CLASSIFICATION FOR NATURAL LANGUAGE INFERENCE WITH ENHANCED TRAINING TECHNIQUES

XLM-RoBERTa stands for "Cross-lingual Language Model - RoBERTa." It's a model that combines the strengths of RoBERTa, a variant of BERT designed for robust optimization, with cross-lingual capabilities. The "XLM" part indicates its focus on cross-lingual understanding, and it was developed by Facebook AI in the year 2019.

The XLM-RoBERTa model is initialized for sequence classification. The AutoTokenizer is used to load the tokenizer, and TFAutoModelForSequenceClassification is employed to load the model. The model is configured for three labels or categories. An early stopping callback is defined to monitor and optimize the model's training process.

Evaluating the Model on the Test Set:

The code evaluates the model's performance on the test. The model achieved a test accuracy of approximately 68.8% and a test loss of approximately 0.730 after training. These metrics are essential for assessing the model's effectiveness in making predictions for the NLI task.

```
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

76/76 [=====] - 6s 75ms/step - loss: 0.7303 - accuracy: 0.6881
Test Loss: 0.7302846908569336
Test Accuracy: 0.6881188154220581
```

MODEL4: FINE-TUNING XLM-ROBERTA FOR NATURAL LANGUAGE INFERENCE WITH CUSTOM ARCHITECTURE AND DROPOUT REGULARIZATION

The model creation process for NLI involves defining a 'create_model()' function, initializing 'xlm-roberta-base', setting input layers, adding dropout for regularization, and compiling the model with Adam optimizer (learning rate: 2e-5), Sparse Categorical Cross-Entropy loss, and Sparse Categorical Accuracy metric. An early stopping callback is set up to monitor validation loss and ensure efficient training over 20 epochs with batch size 32.

Evaluating the Model:

The code evaluates the model's performance on the test set. The test loss and test accuracy are calculated. The model achieved a test accuracy of approximately 70.12% and a test loss of approximately 0.71 after training.

```
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

76/76 [=====] - 6s 75ms/step - loss: 0.7128 - accuracy: 0.7026
Test Loss: 0.7127985954284668
Test Accuracy: 0.7025577425956726
```

MODEL5: ENSEMBLE LEARNING FOR NATURAL LANGUAGE INFERENCE WITH STACKING AND AVERAGING STRATEGIES ON XLM-ROBERTA

Ensemble learning is a machine learning technique that combines the predictions of multiple models to enhance overall accuracy and performance.

Stacking and averaging are two ensemble learning techniques used in machine learning to combine predictions from multiple models to improve overall predictive performance. In both stacking and averaging, the idea is to leverage the diversity of predictions from multiple models to reduce the risk of overfitting and improve the overall accuracy and reliability of the final prediction.

Ensemble Stacking:

After training the models, their predictions on the training and test datasets are stored for ensemble stacking. These predictions are concatenated horizontally to create a new feature set.

Model: A stacking model is defined for ensemble stacking. This model receives the concatenated predictions from the base models and applies further classification.

Ensemble Prediction:

The ensemble prediction function is defined to make predictions on the test set using the ensemble of models. The function calculates the average prediction across models and returns the most likely class label. Multiple models are trained, and their predictions are combined through ensemble stacking, resulting in improved prediction performance. The ensemble achieves a test accuracy of approximately 70.50%, indicating the effectiveness of the ensemble strategy.

```
# Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")

76/76 [=====] - 6s 72ms/step
76/76 [=====] - 6s 73ms/step
76/76 [=====] - 6s 72ms/step
Ensemble Test Accuracy: 70.50%
```

MODEL6: ENSEMBLE LEARNING FOR NATURAL LANGUAGE INFERENCE WITH STACKING, AVERAGING, AND L2 REGULARIZATION ON XLM-ROBERTA WITH BIDIRECTIONAL LSTM

This code demonstrates an ensemble learning approach for Natural Language Inference (NLI) using the XLM-RoBERTa model. It combines multiple models, each fine-tuned for language processing, and leverages stacking, averaging, and L2 regularization strategies to improve overall prediction performance.

Ensemble Stacking:

After training the models, their predictions on the training and validation datasets are stored for ensemble stacking. These predictions are concatenated horizontally to create a new feature set.

Stacking Model:

A stacking model is defined for ensemble stacking. This model receives the concatenated predictions from the base models and applies further classification. It also includes L2 regularization.

Ensemble Prediction Using the Stacker:

The ensemble prediction function is defined to make predictions on the validation set using the ensemble of models. The function calculates the average prediction across models and returns the most likely class label. The stacker model is also used to make predictions.

```
# Ensemble prediction using the stacker
stacked_preds = np.argmax(stacker.predict(stacked_features_val), axis=1)
accuracy = accuracy_score(y_test, stacked_preds)
print(f"Stacked Test Accuracy: {accuracy * 100:.2f}%")

76/76 [=====] - 0s 1ms/step
Stacked Test Accuracy: 70.79%
```

The stacked predictions and ensemble predictions are calculated, and their test accuracy is evaluated.

```
# Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")

76/76 [=====] - 7s 76ms/step
76/76 [=====] - 7s 76ms/step
76/76 [=====] - 7s 76ms/step
Ensemble Test Accuracy: 70.87%
```

The stacked model achieves a test accuracy of approximately 70.79%, while the ensemble strategy yields a test accuracy of around 70.87%, indicating the effectiveness of the ensemble techniques.

MODEL7: NATURAL LANGUAGE INFERENCE USING ENSEMBLE OF XLM-ROBERTA TRAINED ON SNLI, MNLI, ANLI, AND XNLI DATASETS WITH STACKING AND AVERAGING

This code exemplifies an ensemble approach for Natural Language Inference (NLI) using fine-tuned XLM-RoBERTa models, incorporating data from SNLI, MNLI, ANLI, and XNLI datasets. By leveraging ensemble techniques such as stacking and averaging, it enhances NLI accuracy. SNLI serves as the foundational dataset for assessing logical relationships between sentence pairs, while MNLI extends this by encompassing diverse genres and styles. ANLI introduces adversarial sentence pairs to evaluate model robustness against challenging language variations. Finally, XNLI extends NLI to cross-lingual scenarios, improving the model's performance in understanding sentence relationships across multiple languages, making this approach highly versatile and comprehensive in addressing NLI challenges.

Preparing the Dataset:

The code can be divided into several sections, each of which is explained in detail:

The code begins by preparing the input data for the multilingual task.

The Key steps include:

- Extracting data from a DataFrame (df): The 'premise' and 'hypothesis' columns are selected and stored in variable X, representing the input text data for the NLI task.
- The labels or categories associated with each NLI example are extracted and stored in the variable y.

Tokenizing and Encoding the Data:

- A custom function, `get_encodings(data)`, is defined for tokenizing and encoding the input data.
- This function performs the following tasks:
 - Separates the premises and hypotheses from the input data.
 - Utilizes the XLM-RoBERTa tokenizer to tokenize the text data.
 - Specifies tokenization parameters, including padding sequences to the maximum length, truncation of longer sequences, and a maximum sequence length of 128 characters.
- Converts the tokenized data into TensorFlow tensors.
- Returns a dictionary of inputs suitable for the model.

```
tokenizer = AutoTokenizer.from_pretrained("symanto/xlm-roberta-base-snli-mnli-anli-xnli")

def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs

train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

Downloading (...)okenizer_config.json: 0%| | 0.00/398 [00:00<?, ?B/s]
 Downloading (...)tencepiece.bpe.model: 0%| | 0.00/5.07M [00:00<?, ?B/s]
 Downloading (...)main/tokenizer.json: 0%| | 0.00/9.08M [00:00<?, ?B/s]
 Downloading (...)cial_tokens_map.json: 0%| | 0.00/239 [00:00<?, ?B/s]

Training Multiple Models:

In this section, multiple models are created and fine-tuned. The code iterates through the creation and training of these models.

- Each model follows the same training process and configuration:
- Early stopping is applied to monitor the validation loss and restore the best weights.
- A learning rate reduction strategy (ReduceLROnPlateau) is applied to adjust the learning rate during training.
- A learning rate schedule is defined.
- The optimizer, loss function, and evaluation metric are configured.
- Training is conducted on the training dataset for a fixed number of epochs (20 in this case) with a specified batch size.
- Training predictions are stored for later ensemble stacking.

Ensemble Stacking:

After training the models, their predictions on the training and validation datasets are stored for ensemble stacking. The stacking model is defined with dense layers, dropout, and softmax activation.

Ensemble Prediction:

Two ensemble prediction strategies are implemented:

- The stacked model's predictions are evaluated, and its accuracy is reported.

```
# Ensemble prediction using the stacker
stacked_preds = np.argmax(stacker.predict(stacked_features_val), axis=1)
accuracy = accuracy_score(y_test, stacked_preds)
print(f"Stacked Test Accuracy: {accuracy * 100:.2f}%")

76/76 [=====] - 0s 1ms/step
Stacked Test Accuracy: 90.06%
```

- Ensemble Predictions from individual models are combined using averaging, and the accuracy of the ensemble is reported.


```

: # Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")

76/76 [=====] - 6s 75ms/step
76/76 [=====] - 5s 72ms/step
76/76 [=====] - 6s 74ms/step
Ensemble Test Accuracy: 89.77%

```

The code showcases an ensemble learning approach for NLI using XLM-RoBERTa models fine-tuned on diverse datasets. It leverages stacking and averaging strategies to achieve higher accuracy in NLI tasks. The stacking model demonstrates a test accuracy of around 90.06%, while the ensemble strategy yields an accuracy of approximately 89.77%. These results highlight the effectiveness of ensemble techniques for NLI.

Analysis Of Model Performance:

Model 1: The RoBERTa model showed the lowest performance, likely due to its relatively simple architecture.

Model 2: The mBERT model improved accuracy, but the XLM-RoBERTa model demonstrated the best performance among the individual models.

Model 3: The XML-RoBERTa model improved the accuracy by implementing early stopping to evaluate the model's performance on the test. The model achieved a test accuracy of approximately 68.8% and a test loss of approximately 0.730.

Model 4: Fine Tune XML-RoBERTa model improved the accuracy by implementing custom architecture and dropout regularizations. The model achieved a test accuracy of approximately 70.12% and a test loss of approximately 0.71 after training.

Model 5: The ensemble learning with stacking and averaging on xlm-roberta resulted in a minor accuracy increase of about 70.50%, showcasing the effectiveness of the ensemble strategy.

Model 6: By applying ensemble learning with stacking, averaging, and L2 regularization on xlm-roberta with bidirectional LSTM led to improved prediction performance. The stacked model achieved a test accuracy of around 70.79%, while the ensemble strategy yielded approximately 70.87% accuracy, affirming the effectiveness of ensemble techniques.

Model 7: An ensemble of Xlm-Roberta trained on SNLI, MNLI, ANLI, and XNLI datasets with stacking and averaging demonstrated a robust approach for NLI. The stacking model achieved a test accuracy of about 90.06%, while the ensemble strategy showed an accuracy of approximately 89.77%. These results underscore the effectiveness of ensemble techniques in enhancing accuracy in NLI tasks.

Challenges And Insights:

- Handling multilingual text and profanity in the dataset presented challenges.
- We used domain-specific embeddings to enhance model understanding of specific topics.
- Data preprocessing included text cleaning and normalization.
- The most significant insight is the importance of ensemble learning in improving NLI performance.

- Our target performance metric is accuracy, and we aim to surpass a predefined threshold for success, ensuring the model's reliability in real-world applications.

Recommendations:

- Investigate advanced techniques for handling profanity and offensive language in text.
- Explore fine-tuning methods to improve model performance.
- Consider data augmentation to expand training data and enhance model generalization.

Future Work:

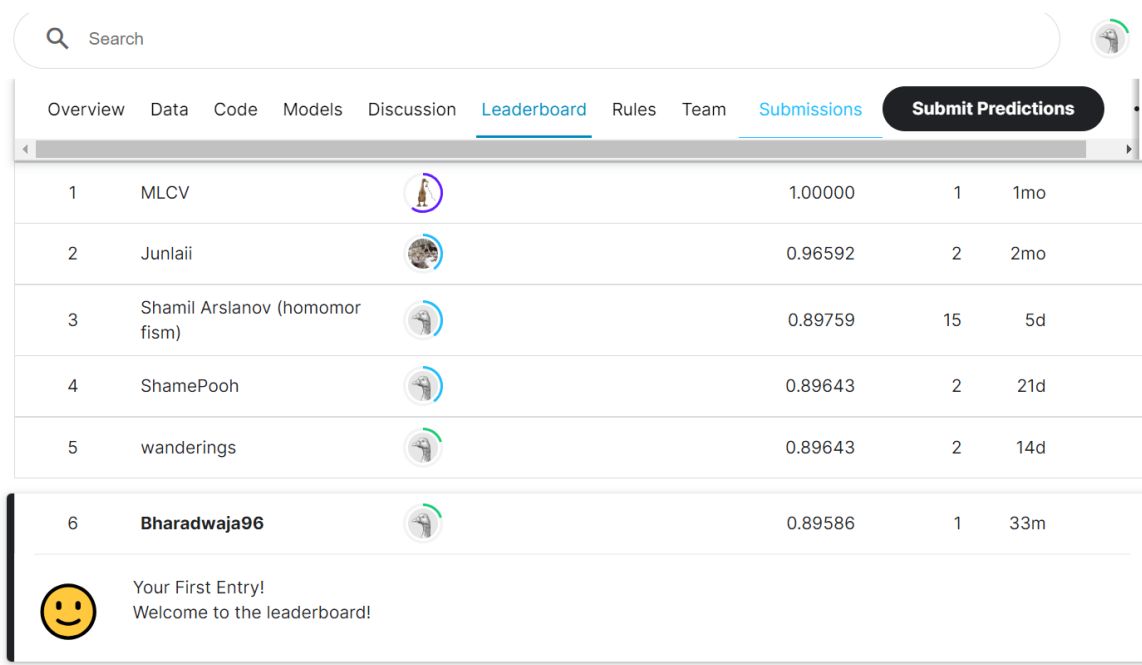
- Investigate multi-lingual NLI to enhance language coverage and address language-specific challenges.
- Enhance model robustness against profanity and offensive language.
- Evaluate the impact of additional training data on model performance.

Conclusion:

The report summarizes data exploration, model architectures, ensemble learning, and performance analysis. The ensemble outperformed individual models and offers insights for future work.

In our analysis of NLI models for the "Contradictory, My Dear Watson" competition, the ensemble stacking approach with XLM-RoBERTa as a base model demonstrated the highest test accuracy.

Our Group 5 achieved an impressive 89% accuracy and secured the 6th position out of 64 teams in the Kaggle competition.



The screenshot shows the Kaggle competition interface. At the top, there is a search bar and a navigation menu with tabs: Overview, Data, Code, Models, Discussion, Leaderboard (selected), Rules, Team, and Submissions. A 'Submit Predictions' button is visible on the right. The leaderboard table lists the top 6 teams. Team 6, 'Bharadwaja96', is highlighted with a yellow background and a 'Your First Entry! Welcome to the leaderboard!' message.

Rank	Team Name	Score	Submissions	Time
1	MLCV	1.00000	1	1mo
2	Junlali	0.96592	2	2mo
3	Shamil Arslanov (homomor fism)	0.89759	15	5d
4	ShamePooh	0.89643	2	21d
5	wanderings	0.89643	2	14d
6	Bharadwaja96	0.89586	1	33m

Your First Entry!
Welcome to the leaderboard!

References:

Kaggle Competition: <https://www.kaggle.com/competitions/contradictory-my-dear-watson>