Report

1. Task

Task 1: News Article Classification

Group Number: Group 7

Description:

The News Article Classification task involved classifying news articles into categories (likely 8 classes, with labels ranging from 1 to 8, based on prior context). The dataset was provided in separate CSV files: `TrainData.csv` and `TrainLabels.csv` for training, `ValidationData.csv` and `ValidationLabels.csv` for validation, and `TestData.csv` for testing. The goal was to predict the category labels for the test set articles.

Task 2: Political Bias Tweet Classification

Group Number: Group 9

Description:

The Political Bias Tweet Classification task required classifying tweets into 5 political bias categories (labels 1 to 5: strongly left to strongly right). The dataset included `training.csv` and `validation.csv` (both with `"tweet"` and `"label"` columns) for training and validation, and `tweets_test.csv` (with a `"tweet"` column and a header `"text","label"`, but no labels) for testing. The test set was expected to have 150 samples, and predictions needed to be saved in `cleaned_combined_result.csv` with a single `"Label"` column.

Task 3: Phishing Email Classification

Group Number: Group 30

Description:

The Phishing Email Classification task focused on classifying emails as spam (0) or non-spam (1). The dataset included training.csv and validation.csv (each with three columns: email address, email text, and label, no headers) for training and validation, and testing.csv (email address and text, no labels) for testing. The goal was to predict binary labels for the test set and submit them in answer.txt, zipped as submission.zip, for the Codabench platform.

2. Approaches

Task 1: News Article Classification

Preprocessing and Data Preparation:

The dataset was provided in separate files for text and labels (`TrainData.csv`, `TrainLabels.csv`, `ValidationData.csv`, `ValidationLabels.csv`, `TestData.csv`), with no headers. I loaded each file using `pd.read_csv(header=None)`, assigning column names (`"text"` for articles, `"label"` for labels). I combined the text and labels into single DataFrames for training and validation, ensuring proper alignment. The DataFrames were converted to Hugging Face `Dataset` objects for compatibility with the `transformers` library. I tokenized the text using the DistilBERT tokenizer with a `max_length` of 512 to handle longer news articles, applied padding and truncation, and mapped the labels from 1 to 8 to 0 to 7 for model training. For the test set, I tokenized the text similarly but did not include labels, as they were not provided.

Model Selection:

I chose the `distilbert-base-uncased` model for this task because it's well-suited for text classification. From my experience in Homework 4, I learned that DistilBERT is more efficient than BERT (66M parameters vs. 110M) while delivering comparable performance, making it a practical choice for this task. I avoided zero-shot learning, as it produced poor results in Homework 4 for similar text classification tasks, and opted for supervised fine-tuning instead.

Training and Hyperparameter Tuning:

I fine-tuned the `distilbert-base-uncased` model using the Hugging Face `Trainer` API, with accuracy as the evaluation metric on the validation set. I started with default hyperparameters: a learning rate of 2e-5, weight decay of 0.01, and 3 epochs, using a batch size of 16. The initial validation accuracy was around 53%, which was not satisfactory.

To improve performance, I tweaked the hyperparameters:

- Increased the learning rate to 3e-5 to accelerate convergence.
- Increased the weight decay to 0.1 (from 0.01) to add more regularization and mitigate overfitting.
- Increased the number of epochs to 5 (from 3) to allow the model more time to learn.

These adjustments improved the validation accuracy to approximately 70%, a significant 17% increase, indicating that the model benefited from the additional training and regularization. I then experimented by decreasing the learning rate back to 2e-5 to test the effect of momentum, but the accuracy dropped by 3% (to ~67%), confirming that the higher learning rate of 3e-5 was more effective for this task.

Evaluation:

I evaluated the model after each training run using the validation set, with accuracy computed via the `evaluate` library's accuracy metric. The `Trainer` API automatically saved the best model based on validation accuracy, which I used for generating test set predictions.

The model achieved a test accuracy of 64% on the Codabench leaderboard.

Task 2: Political Bias Tweet Classification

Preprocessing and Data Preparation:

The dataset included `training.csv` and `validation.csv` (with `"tweet"` and `"label"` columns) and `tweets_test.csv` (with a `"tweet"` column, a header `"text","label"`, but no labels). I loaded the training and validation files directly with `pd.read_csv()`, as they had headers, and used the `"tweet"` column for text and `"label"` for labels, renaming them to `"text"` and `"labels"` for consistency. For the test set, I initially loaded the file with `header=None`, which caused an issue: the header row (`"text","label"`) was treated as a data row, resulting in 151 samples instead of the expected 150. This led to a mismatch between the predicted labels (151) and the expected test set size (150). I fixed this by reloading the test set with `header=0` and `usecols=["tweet"]`, ensuring exactly 150 samples by excluding the header and the empty `"label"` column.

I converted the DataFrames to Hugging Face `Dataset` objects, tokenized the tweets using the DistilBERT tokenizer with a `max_length` of 128 (appropriate for shorter tweet text), and applied padding and truncation. I mapped the labels from 1 to 5 to 0 to 4 for training, and formatted the datasets for PyTorch, keeping only the necessary columns (`input_ids`, `attention_mask`, `labels`).

Model Selection and Initial Attempts:

I initially selected the pretrained model `m-newhauser/distilbert-political-tweets` (https://huggingface.co/m-newhauser/distilbert-political-tweets) because it was fine-tuned on US Senator tweets for political bias classification, which seemed relevant to this task. However, I encountered a classifier head mismatch: the pretrained model was fine-tuned for a 2-class task (Republican vs. Democrat), while my task required 5 classes. I resolved this by using `ignore_mismatched_sizes=True`, reinitializing the classifier head for 5 classes while retaining the pretrained weights. I fine-tuned this model with a learning rate of 3e-5, weight decay of 0.1, and 5 epochs, but the validation accuracy was only ~21%. I believe this low accuracy was because the model's original fine-tuning was too specific (Republican vs. Democrat) for my task, which involved a broader political bias spectrum, including Canadian politics and #elxn43 hashtags.

Due to the poor performance, I switched to the base `distilbert-base-uncased` model, the same model used in the News Article Classification task, to fine-tune from scratch for this 5-class classification task. This model doesn't have a pretrained classifier head, so there were no mismatch issues, and fine-tuning started with a randomly initialized classifier head for 5 classes.

Training and Hyperparameter Tuning:

Using `distilbert-base-uncased`, I fine-tuned the model with initial hyperparameters: a learning rate of 3e-5, weight decay of 0.1, and 5 epochs, matching the setup from the News Article Classification task. The validation accuracy was ~27%, a 6% improvement over the pretrained model, likely because the base model wasn't constrained by prior fine-tuning on a mismatched task. I experimented with further tuning:

- Increased the learning rate to 4e-5 to speed up learning.
- Increased the number of epochs to 7 to allow more training time.

However, this decreased the accuracy to ~25%, likely due to overfitting, as the model trained too long on the dataset, memorizing training patterns instead of generalizing to the validation set. I reverted to the original hyperparameters (3e-5 learning rate, 5 epochs) for the final model, as they provided the best performance.

Evaluation:

I evaluated the model on the validation set (`validation.csv`), which had labels, using accuracy as the metric via the `evaluate` library. The `Trainer` API facilitated this process, and I monitored the validation accuracy after each epoch to identify the best model. For the test set, I ensured the predictions aligned with the expected 150 samples by fixing the data loading issue, and saved the predictions in `cleaned_combined_result.csv` with a single `"Label"` column, as required.

The test accuracy reached 21% as evaluated on the Codabench platform.

Task 3: Phishing Email Classification

Preprocessing and Data Preparation:

The dataset (training.csv, validation.csv, testing.csv) had no headers. For training and validation, I loaded the CSVs using `pd.read_csv(header=None, usecols=[1, 2])`, naming columns "text" for email content and "labels" for binary labels (0 for spam, 1 for non-spam), ignoring the email address column. For the test set, I used `usecols=[1]` to load only the text column, as no labels were provided. The emails varied significantly: spam contained promotional content (e.g., “Father's Day Special...$49.95”), while non-spam included technical or personal messages (e.g., “GTV weekly project status update”).

I converted the `DataFrames` to Hugging Face Dataset objects for compatibility with the transformers library. Using the `DistilBERT` tokenizer, I tokenized the raw email text with a `max_length` of 512 to accommodate longer emails, applying padding and truncation to standardize input lengths. The raw text was used without additional cleaning, as it preserved original patterns like URLs and pricing in spam emails. I formatted the datasets for `PyTorch`, retaining only `input_ids`, `attention_mask`, and labels for training and validation, and `input_ids` and `attention_mask` for testing. Labels (0 and 1) were kept as-is, requiring no mapping since they were already suitable for binary classification.

Model Selection:

I chose `distilbert-base-uncased` for its efficiency and effectiveness in text classification tasks. With 66 million parameters, it's lighter than BERT (110 million parameters), making it suitable for fine-tuning on the email dataset, which ranged from short promotional spam to detailed technical emails. My prior experience with DistilBERT in similar tasks (Homework 4, as noted in Task 1) showed it balances performance and computational cost. I opted for supervised fine-tuning over pretrained models fine-tuned for unrelated tasks (e.g., sentiment analysis), as the binary spam/non-spam classification required task-specific learning.

Training and Hyperparameter Tuning:

I fine-tuned the model using the Hugging Face Trainer API, with accuracy as the primary evaluation metric on the validation set. Initial hyperparameters were:

- Learning rate: 3e-5
- Weight decay: 0.01
- Epochs: 3
- Batch size: 16

This setup achieved a validation accuracy of approximately 62%, which was suboptimal, as the model struggled with distinguishing some spam emails (e.g., those with subtle promotional language) from non-spam.

To improve performance, I tuned the hyperparameters:

- **Increased Epochs**: Extended training to 5 epochs from 3, allowing the model more time to learn complex patterns, such as promotional cues in spam versus technical terms in non-spam.
- **Increased Weight Decay**: Raised weight decay to 0.1 from 0.01 to add stronger regularization, helping prevent overfitting on noisy email data (e.g., URLs, special characters).

These changes improved validation accuracy to 68%, a 6% increase, indicating better generalization. I experimented with a lower learning rate (2e-5) to test stability, but accuracy dropped to ~65%, suggesting 3e-5 was more effective for convergence. The batch size remained at 16, balancing memory usage and training speed, and I retained the default linear learning rate scheduler to maintain simplicity.

Evaluation:

I evaluated the model on validation.csv after each epoch using the evaluate library's accuracy metric. The Trainer API saved the best model based on validation accuracy, which I used for test predictions. The final validation accuracy of 68% reflected the model's ability to classify most emails correctly, though some errors persisted (e.g., non-spam with links misclassified as spam). For the test set, I generated binary predictions (0 or 1), ensuring the output matched the number of rows in testing.csv. Predictions were saved in answer.txt as a vertical list of labels, one per line, and zipped into submission.zip as required by Codabench.

The model achieved a test accuracy of 68.8% on Codabench.

3. Best Results

Task 1: News Article Classification

The best approach was using `distilbert-base-uncased` with a learning rate of 3e-5, weight decay of 0.1, and 5 epochs, achieving a validation accuracy of ~70% and test accuracy of ~64%. This approach performed well because:

- DistilBERT's efficiency (66M parameters) allowed effective fine-tuning on the news article dataset, capturing semantic patterns in longer texts with a `max_length` of 512.
- The increased learning rate (3e-5) and epochs (5) enabled better convergence, while the higher weight decay (0.1) prevented overfitting, which was critical for an 8-class classification task.
- Fine-tuning from scratch (for classification) allowed the model to learn task-specific patterns without biases from prior fine-tuning, and the balanced hyperparameters ensured the model generalized well to the validation set.

Task 2: Political Bias Tweet Classification

The best approach was using `distilbert-base-uncased` with a learning rate of 3e-5, weight decay of 0.1, and 5 epochs, achieving a validation accuracy of ~27% and test accuracy of ~21%. Although this accuracy is low, it outperformed the pretrained `m-newhauser/distilbert-political-tweets` model (which achieved ~21% validation accuracy) and several of my peers' test accuracy on the codabench platform. The reasons for this performance include:

- The base `distilbert-base-uncased` model was more adaptable to the 5-class political bias spectrum of this task, as it wasn't constrained by prior fine-tuning on a binary task (Republican vs. Democrat), unlike the pretrained model.
- The dataset's complexity (nuanced political bias categories, including Canadian politics) and limited size likely contributed to the low accuracy. Fine-tuning with 5 epochs balanced learning and generalization, as increasing to 7 epochs caused overfitting.
- Proper preprocessing, such as setting a `max_length` of 128 for tweets and fixing the test set loading to ensure 150 predictions, ensured the model processed the data correctly.

Task 3: Phishing Email Classification

The best approach used distilbert-base-uncased with a learning rate of 3e-5, weight decay of 0.1, 5 epochs, and a batch size of 16, achieving a validation accuracy of 68% and test accuracy of ~68.8%. This approach performed well because:

- **DistilBERT's Efficiency**: With 66 million parameters, DistilBERT effectively captured contextual differences between spam (e.g., promotional offers like "$49.95") and non-spam (e.g., technical updates), making it suitable for the varied email dataset.
- **Extended Training**: Increasing epochs to 5 allowed the model to learn nuanced patterns, such as distinguishing promotional language from legitimate communication, improving accuracy by 6% from the initial 62%.
- **Stronger Regularization**: The higher weight decay (0.1) mitigated overfitting on noisy email text, including URLs and special characters, ensuring better generalization to the validation set.
- **Appropriate Learning Rate**: The 3e-5 learning rate balanced convergence speed and stability, outperforming a lower rate (2e-5), which reduced accuracy to ~65%.
- **Raw Text Usage**: Retaining raw text preserved critical spam indicators (e.g., pricing, links), enabling the model to learn directly from the dataset's natural structure.

The model produced answer.txt with binary predictions (0 for spam, 1 for non-spam) for testing.csv, zipped as submission.zip, meeting Codabench submission requirements.