Maryam Abrahams
ABRMAR043

Logistic Regression News Classification

Data Processing:

Our goal with data processing is to load and preprocess the MasakhaNEWS dataset for both English and Xhosa languages, applying text cleaning and label encoding to prepare the data for feature extraction and model training in later sections.

Design Implementation:

1. **load():** Reading in the training, development, and test .tsv files for both English and Xhosa, returning all as dataframes with pandas. This function effectively fetches the data.
2. **process_text()**: Standardizing all the read-in text by converting it to lowercase, removing punctuation and special characters, removing digits, and normalizing extra white spaces. In this way, we parse the text so that it's more uniform and we reduce the risk of future read errors.
3. **clean():** Applying the *process_text* function to the text column of our dataframe, to produce a new *clean_text* column, which we can access.
4. **Encoding**: Fitting a label Encoder on both the English and Xhosa training set categories and converting them to numeric values so that we can convert the dev and test set categories into the same numeric values. This allows our model to access labels as numbers instead of text.

Multinomial Logistic Regression:

For this task, using the code from the Multinomial Logistic Regression Tutorials, we implement a multinomial logistic regression in Pytorch to serve as a baseline classifier for predicting news categories from our extracted features.

Design Implementation:

1. **_init_():** initializing the model
2. **forward():** defining the forward pass and applying the linear transformation to input features
3. **compute_probabilities():** Applying the softmax function to the linear outputs to convert them into class probability distributions.
4. **predict():** returns the most likely class label by selecting the index with the highest probability.

Training:

When training our multinomial logistic regression model, we use the previously found best hyperparameters, with early stopping to prevent any overfitting. The goal is to optimize model weights on the training data while watching the validation performance.

Design Implementation:

1. **demonstrate_training():** demonstration of our model with the best hyperparameter combination shows us the optimal performance of our model to evaluate.
2. **train_model():** handles the full training loop by converting features and labels to Pytorch tensors and batching them to optimize memory and gradient updates.
3. **SGD + CrossEntropyLoss:** iteratively updates models with cross-entropy being the standard and appropriate loss function for our multi-class classification.
4. **Validation tracking:** We make sure to consistently measure validation accuracy and save/return the model that produces the highest validation accuracy, which is not necessarily the model with the highest epoch, but if it is, we can better evaluate the trade-off between time and accuracy and where we're willing to train until.

Demonstration:

```
Epoch   1: Train Loss = 1.6571, Val Acc = 0.4958, Best = 0.4958
Epoch   2: Train Loss = 1.5786, Val Acc = 0.5699, Best = 0.5699
Epoch   3: Train Loss = 1.5065, Val Acc = 0.6801, Best = 0.6801
Epoch   4: Train Loss = 1.4402, Val Acc = 0.7648, Best = 0.7648
Epoch   6: Train Loss = 1.3230, Val Acc = 0.8093, Best = 0.8093
Epoch   7: Train Loss = 1.2726, Val Acc = 0.8178, Best = 0.8178
Epoch   9: Train Loss = 1.1819, Val Acc = 0.8199, Best = 0.8199
Epoch  10: Train Loss = 1.1413, Val Acc = 0.8305, Best = 0.8305
Epoch  11: Train Loss = 1.1045, Val Acc = 0.8326, Best = 0.8326
Epoch  12: Train Loss = 1.0703, Val Acc = 0.8390, Best = 0.8390
Epoch  13: Train Loss = 1.0383, Val Acc = 0.8453, Best = 0.8453
Epoch  16: Train Loss = 0.9558, Val Acc = 0.8496, Best = 0.8496
Epoch  19: Train Loss = 0.8876, Val Acc = 0.8559, Best = 0.8559
Epoch  20: Train Loss = 0.8679, Val Acc = 0.8453, Best = 0.8559
Early stopping at epoch 24. No improvement for 5 epochs.
Training completed. Best validation accuracy: 0.8559 at epoch 19
```

From the graph and the results, it is clear that validation accuracy increases as training loss decreases, which coincides with a higher number of epochs. The numerical point of overfitting is seen to be at epoch **19,** with a validation accuracy of **~86%** that does not improve with additional epochs. From our graphs, however, we can see that this plateau in validation accuracy more likely begins around epoch **10**. Because the training cycle is fairly fast in the circumstances of this assignment, however, we can safely take the better stopping epoch (**19**), but if, for some reason, it were to take much longer, then we can rather consider a trade-off of validation accuracy for speed by using a lower epoch, like epoch **10**.

Hyperparameter Tuning:

The goal of the hyperparameter tuning is to evaluate different learning rates and batch sizes to find an ideal configuration for our training loop. For this reason, I implemented this before task 3, and I judged the effectiveness of the combination by seeing which one maximizes validation accuracy.

Design Implementation:

1. **Limited number of epochs:** By limiting the number of epochs, we avoid excessive run times.
2. **Setup:** By selecting the English TF-IDF features as input, we balanced out the word frequency with importance so that we can determine class input and dimensions to match the dataset.
3. **Grid Search Loop:** By testing different combinations of multiple learning rates and batch sizes, we allow for a fair comparison.
4. **Tensor conversion:** Originally, I was having issues when passing TF-IDF features directly on Pytorch, so after AI-assisted debugging with DeepSeek, converting data to tensors and grouping it inside the loop fixed my issue by ensuring the model always received correctly formatted inputs.

Results Analysis:

```
Tuning for: English with TF-IDF
Input dimension: 5000
Number of classes: 6

Testing lr=0.01, batch_size=16... Val Acc: 0.2119
Testing lr=0.01, batch_size=32... Val Acc: 0.2119
Testing lr=0.01, batch_size=64... Val Acc: 0.2119
Testing lr=0.05, batch_size=16... Val Acc: 0.5699
Testing lr=0.05, batch_size=32... Val Acc: 0.3941
Testing lr=0.05, batch_size=64... Val Acc: 0.2182
Testing lr=0.1, batch_size=16... Val Acc: 0.7648
Testing lr=0.1, batch_size=32... Val Acc: 0.5148
Testing lr=0.1, batch_size=64... Val Acc: 0.4025

==================================================
Best params: {'lr': 0.1, 'batch_size': 16}
Best validation accuracy: 0.7648305296897888
==================================================
```

Very low learning rates (**0.01**) consistently produced only **~21%** validation accuracy. This implies that the model's updates were too small to make any meaningful progress during the limited training time. As we increased the learning rate to a moderate one, validation accuracy also increased. But for better learning rates, validation accuracy consistently decreased with an increase in batch size (**~76%** < **~40%**). This explains our optimal configuration of **lr = 0.01** and **size = 16** associated with the **~76%** accuracy.

Feature Extraction:

In this section, we extract numerical features from the cleaned English and Xhosa text using different vectorization techniques (Bag-of-Words, binary presence/absence, and TF-IDF) to represent documents in a format suitable for machine learning models.

Design Implementation:

1. **Bag-of-Words:** Converts each document into a vector of word counts, with a max of 5000 most frequent words as features.
2. **Binary Features:** Represents each document as a vector of binary indicators like a Bernoulli random variable (1 - present, 0 - absent)
3. **TF-IDF:** Weights each word by its importance, with more important words getting higher weightings

Results Analysis:

Both the English and Xhosa vocabularies were capped at 5000 features, meaning the vectorizers used the 5000 most common tokens. The printed sample TF-IDF rows mostly contained zeros which reflects the sparse nature of the text data, implying most words don't appear in a given document. The sparsity highlights the curse of dimensionality within our text data since each document only uses a small fraction of the total vocabulary.

---

isiXhosa Training Decisions:

By experimenting with several strategies that improve the isiXhosa multinomial logistic regression model, we can address potential overfitting and class imbalances within the training data. By comparing all our following techniques to a baseline accuracy, we support a fair evaluation. Our goal is to maximize accuracy without unfair category representation.

Design Implementation:

1. **L2 Regularization:** penalizes large weights to reduce overfitting.

2. **L1 Regularization**: encourages sparsity in model weights, so we discount irrelevant features.
3. **Upsampling:** replicates samples from underrepresented classes so all classes have equal size; in this way, classes are balanced.
4. **Downsampling:** reduces samples from overrepresented classes so that all classes have qual size; in this way, classes are also balanced but with a loss of data.

```
============================================================
SUMMARY
============================================================
Baseline:           0.6463
Best L2:            0.6599 ( +0.0136)
Best L1:            0.6599 ( +0.0136)
Upsampling:         0.8707 ( +0.2245)
Downsampling:       0.7891 ( +0.1429)

Best technique: Upsampling with 0.8707
============================================================
```

Result Analysis:

- **L2 & L1 (65.9%, ~1% improvement):** negligible improvement -> regularization prevents overfitting but doesn't handle class imbalances well.
- **Upsampling (87.1%, ~22% improvement):** huge improvement -> upsampling balances classes, showing that class imbalances were a root issue.
- **Downsampling (78.9%, ~14% improvement):** moderate improvement -> similarly balances classes but in a way that loses data; the moderate improvement is a sign of too much data lost.

We can conclude that upsampling was the best technique for boosting accuracy, and significantly so (22% improvement from the baseline). This is likely because there were severe class imbalances that needed correcting, and in doing so, the model better generalizes across all categories.

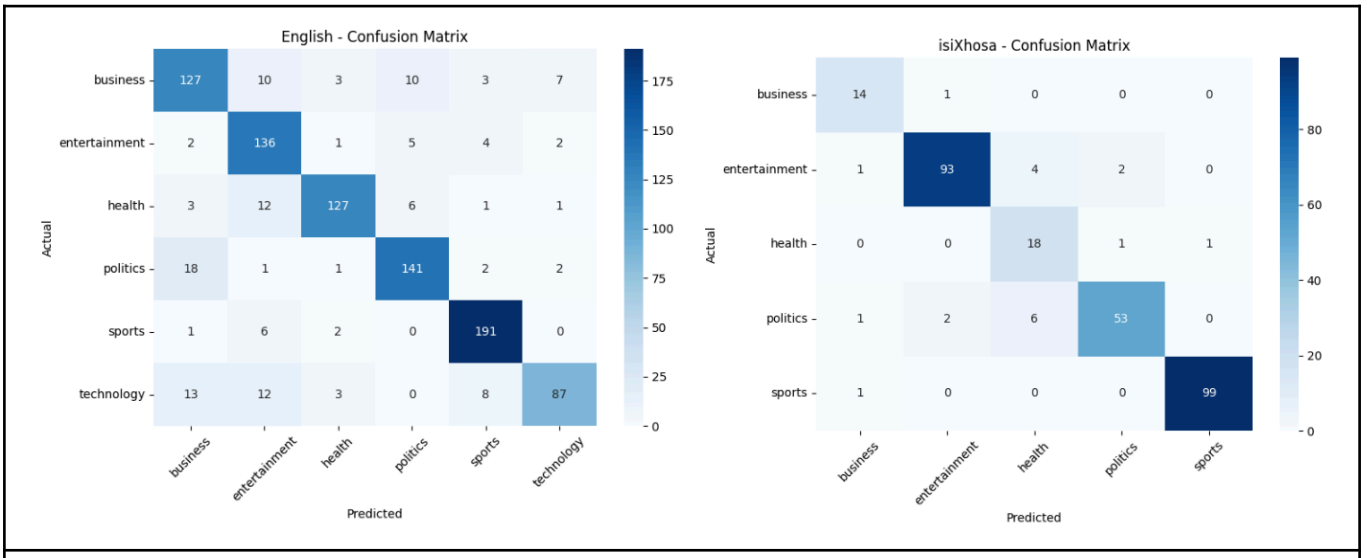
Evaluation:

The final valuation of both the Xhosa and English test sets allows us to quantify the performance of our model across multiple evaluation metrics. We run our test set on a final English and final Xhosa model, which involves all optimizations we found previously throughout our training and validation evaluation. Using confusion matrices, we can better understand the strengths, weaknesses, and class balance of our predictions and model.

Design Implementation:

1. **evaluation():** the evaluation converts test data to tensors. It also generates predictions, computes evaluation metrics: accuracy, recall, precision, and F1 scores.

Then it analyzes the differences between the micro and macro F1 to get better class balance insights and plots a confusion matrix. With all this provided information, we get a comprehensive evaluation of our model, its strengths and weaknesses.




```
==================================================
TASK 7: MODEL EVALUATION
==================================================

English Evaluation:
Accuracy:  0.8534
Precision: Micro=0.8534, Macro=0.8555
Recall:    Micro=0.8534, Macro=0.8440
F1-Score:  Micro=0.8534, Macro=0.8468
→ Micro and Macro F1 are similar: classes are relatively balanced
Most reliable: sports (191 correct)
Least reliable: technology (87 correct)

isiXhosa Evaluation:
Accuracy:  0.9327
Precision: Micro=0.9327, Macro=0.8743
Recall:    Micro=0.9327, Macro=0.9216
F1-Score:  Micro=0.9327, Macro=0.8925
→ Micro and Macro F1 are similar: classes are relatively balanced
Most reliable: sports (99 correct)
Least reliable: business (14 correct)
```

Results Analysis:

- **English**: accuracy (85%), precision (86%), recall(84%), F1-micro(85%), F1-macro(~84%)
- **isiXhosa**: accuracy (93%), precision (87%), recall(92%), F1-micro(93%), F1-macro(~89%)

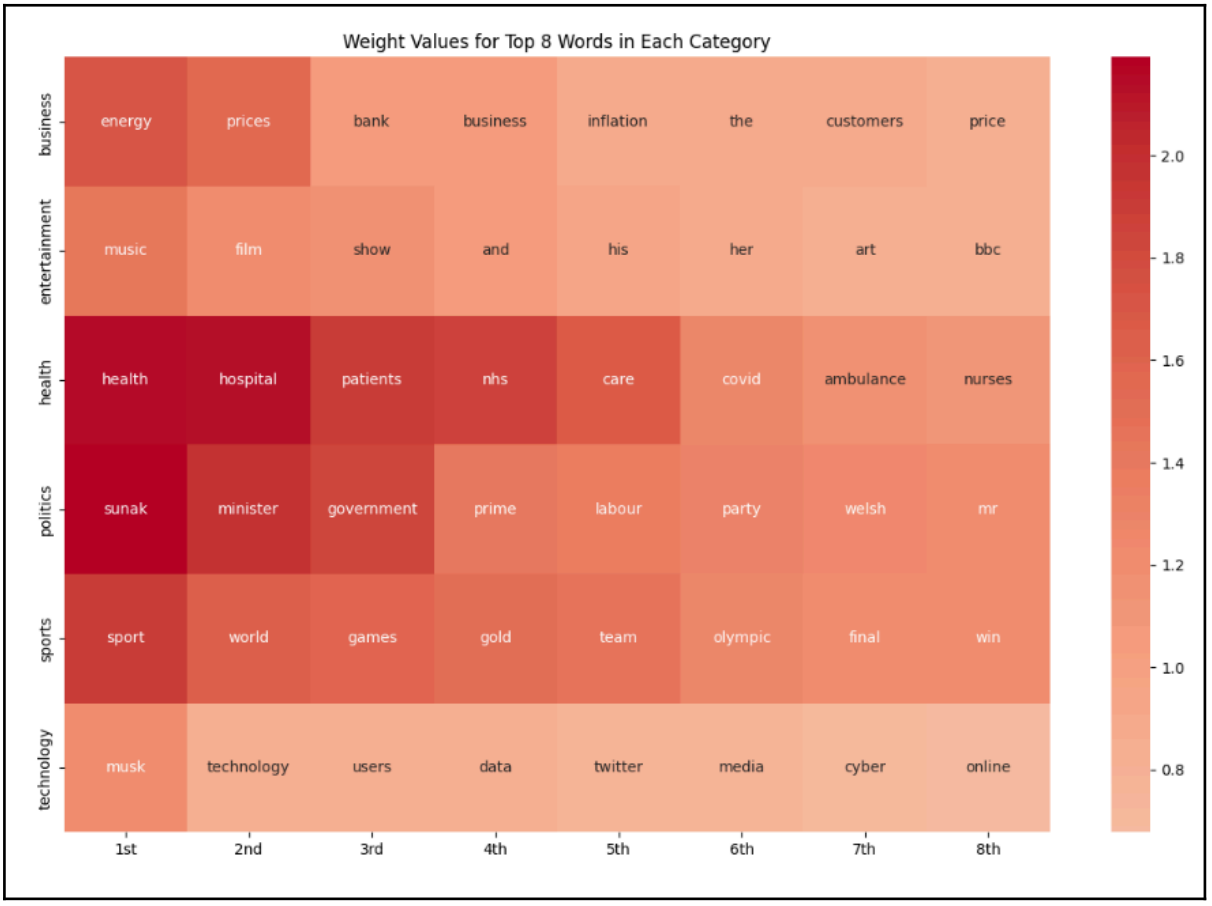
These very high results are mirrored in the strong diagonal seen in both the confusion matrices, where the predicted classes equal the true class values consistently, with little stray from this trend. The model is particularly good at classifying sports news, but struggles to correctly differentiate technology and business news, although only slightly so. The high accuracy, precision, and recall indicate that our model learned meaningful patterns and is good at correctly classifying news in both English and isiXhosa. The similar micro and macro F1-scores show that the classes are relatively balanced; in the case of isiXhosa data, this is thanks to upsampling.

Weight Analysis:

The purpose of weight analysis is to analyze learned softmax weights and identify key discriminative words that drive classification decisions, revealing the model's understanding of category-specific vocabulary.

Design Implementation:

1. **Weight Extraction:** retrieves the learned linear weights from the English model.
2. **Top words per category:** For each class, the weights per word are sorted to find the top 10 words with the highest positive contributions. The fact that Rishi Sunak's name is one of these points to some bias towards British politics.
3. **Heatmap:** displays the top 8 heavily weighted words per class, with color representing their influence. This visualization makes our results easier to interpret.



Results Analysis:

For the most part, the most important words per category are consistent with expectation, with jargon per sector expectedly being most deterministic (eg, hospital -> health, minister -> politics, etc.). The weight analysis mostly serves to confirm that our model learned meaningful relationships between these sector-specific words and the related label.