NLP Assignment 1

Team Members: Theodoros Lambrou, Georgia Zavou, Ilaria Curzi

1. Objective

"Can you identify question pairs that have the same intent?"

This project is about the Quora Question Pairs challenge: given two questions, predict whether they are semantically duplicate.

2. Dataset and Splits

We used the provided **`quora_data.csv`** file, which lacks labeled validation/test sets. As instructed in the assignment guide, we split the data like so:

```
A_df, test_df = train_test_split(quora_df, test_size=0.05,
random_state=123)
train_df, val_df = train_test_split(A_df, test_size=0.05,
random_state=123)
```

3. Feature Engineering

We engineered features of increasing complexity to capture various similarity signals:

Baseline Features:

- **Jaccard Similarity:** Token overlap ratio between two cleaned questions.

Improved Features:

- **Length Difference** – Absolute difference in token counts.
- **TF-IDF Cosine Similarity** – Using a shared vectorizer for semantic similarity.
- **Levenshtein Ratio** – Character-level edit distance.
- **Shared Bigrams Ratio** – Overlap of word pairs.
- **Average Word Length Difference** – Style/complexity signal.
- **SpaCy Cosine Similarity** – Vector-based similarity using pretrained embeddings.

Notes:

- Cosine similarity is a commonly used measure in text analysis, as it captures the angle between two vector representations, not their magnitude. This makes it ideal for comparing TF-IDF vectors or dense embeddings, where the absolute values vary widely across documents but direction reflects meaning.

- Levenshtein distance focuses on character-level similarity and helps capture typographical variations and near-duplicate questions.
- Jaccard and shared bigram ratios offer surface-level similarity insights, especially useful when questions share vocabulary but differ structurally.
- Feature extraction is computationally expensive. SpaCy embeddings and Levenshtein distance slowed down execution time. To try mitigate this, we implemented a caching method for SpaCy vectors (`_spacy_cache`) and reused the TF-IDF vectorizer across train/val/test splits.
- We focused on features with high interpretability and impact, such as lexical overlap, TF-IDF cosine similarity, Levenshtein ratio, and SpaCy vectors. However we also considered more granular linguistic features (e.g. syllable counts, Flesch readability scores, named entity overlap). We decided to not include these due to high computational cost and limited expected performance gain on this task.

4. Modelling

We trained and compared two classifiers:

1. Logistic Regression:
    - Baseline (with Jaccard)
    - Improved (with all features)
2. Random Forest:
    - Trained on full feature set
    - Hyperparameters tuned via GridSearchCV

Notes:

- All models were trained on the training set and evaluated on validation and test sets using ROC AUC, precision and recall.
- Hyperparameter tuning was performed using **`GridSearchCV`** with 3-fold cross-validation.

5. Results

Logistic Regression (Improved features)

	ROC AUC	Precision	Recall
Train	0.7763	0.6118	0.5057
Validation	0.7660	0.5959	0.4929
Test	0.7678	0.6012	0.5012

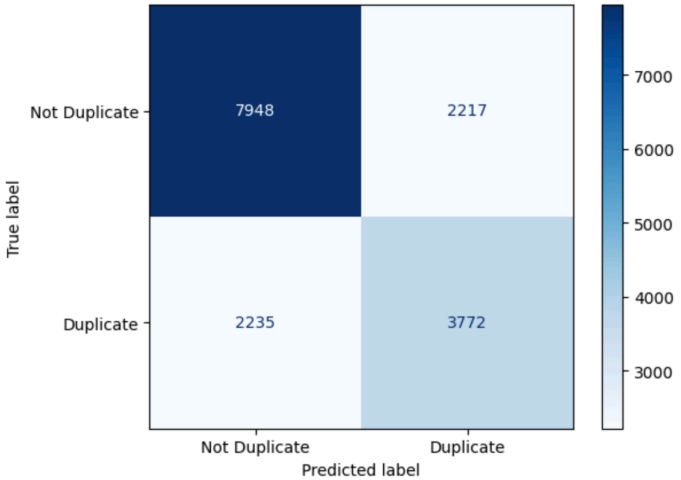
Random Forest (Improved Features)

	ROC AUC	Precision	Recall
Train	0.9997	0.9991	0.9999
Validation	0.8136	0.6342	0.6226
Test	0.8137	0.6298	0.6279

Random Forest consistently outperformed Logistic Regression on validation and test sets. This likely is because of its ability to capture nonlinear interactions between our engineered features, especially those involving lexical patterns and vector similarities. Logistic Regression, while interpretable and efficient, has a hard time when decision boundaries are not linearly separable.

Other classifiers (e.g XGBoost, gradient boosting) which might increase accuracy further could be explored, but we focused on Random Forests and Logistic Regression to balance model complexity with interpretability.

As shown in the confusion matrix, our model makes balanced errors across both classes, with approximately 2,200 false positives and 2,200 false negatives. This indicates good handling of class imbalance, and aligns with our recall score of ~0.63 on the test set.



6. Limitations/Challenges

- Feature engineering accounted for the majority of processing time, and especially when using SpaCy embeddings and Levenshtein distances on the full dataset.
- We did not use deep learning approaches (e.g.LSTMs or transformers), which might perform better but would require GPU resources and more training time.
- An analysis of false predictions (in **`reproduce_results.ipynb`**) shows that false positives are often topically related but not truly duplicate, while false negatives usually involve semantically identical questions with varied phrasing. This highlights the challenge of relying solely on lexical and structural features, and suggests that deep semantic models could improve results.

7. Team Contributions

- Theodoros Lambrou
  - Created the project structure and baseline pipeline.
  - Implemented preprocessing, Jaccard similarity, and TF-IDF cosine similarity.
  - Optimized vector caching.
- Georgia Zavou
  - Built the ML training and evaluation functions.
  - Designed the `extract_improved_features()` pipeline.
- Ilaria Curzi
  - Developed custom advanced features: Levenshtein, shared bigrams, average word length, and SpaCy cosine similarity.

All members contributed to feature engineering and testing, focusing on different feature types and model evaluation.

8. Conclusion

Our best-performing model, Random Forest with engineered features, achieved a test ROC AUC of 0.8137. This demonstrates strong performance in detecting duplicate questions by leveraging a combination of lexical and semantic similarity features. The model generalized well to unseen data and maintained a balanced error profile. Some semantically equivalent questions with dissimilar phrasing were missed, so possible future improvements could include deep contextual models like transformers. Our approach is interpretable and practical, making it a good option for question similarity tasks.