Project Documentation
Visual Sentence Complexity Prediction

Dilirici Mihai

Group 411

For this project, I have trained two machine learning models: Neural Network and XGBoost Regressor. Both successful and unsuccessful attempts are described below.

Neural Network

The first model trained for this task is NN, implemented through the following steps:

1. Data Preprocessing

Before applying the TF-IDF transformation, several preprocessing steps were taken to ensure that the text data is suitable for machine learning:

1. **Tokenization**:
   - The raw text was split into individual words (tokens), allowing the model to process the structure of the language.
2. **Stop Word Removal**:
   - Common, uninformative words such as "and", "the", "is" were removed from the text to reduce noise in the data.
3. **TF-IDF Transformation**:
   - The **TF-IDF vectorizer** was applied to convert the text data into numerical feature vectors. The vectorizer was configured to retain only the top **750 features** based on their importance.
   - This transformation was applied to both the training, validation, and test sets to ensure consistency.
4. **Feature Scaling**:
   - The numerical features were scaled using the **StandardScaler** to normalize the feature values. This step is crucial for neural networks

as it helps improve the training efficiency and performance by ensuring that the features have a similar scale.

2. **Feature Set Description**

The text data is transformed using **Term Frequency-Inverse Document Frequency (TF-IDF)**, which is a statistical technique that evaluates the importance of a word in a document relative to its frequency in a collection of documents. This helps capturing how significant are specific words while diminishing the effect of commonly used, less informative words (like "the", "I", etc.).

In this project, I have selected the top **750 most important words** based on their TF-IDF scores to reduce the dimensionality and ensure that the most meaningful terms are used as features. The TF-IDF feature set is then used as input to the neural network for score prediction.

3. **Model Architecture**

After experimenting layers number and dimensions, I used this final form of the neural network, with the following components:

- **Input Layer:**
  - Takes TF-IDF features (750 dimensions) as input.
- **Hidden Layers:**
  - **Layer 1:** 128 neurons, activated using the Rectified Linear Unit (ReLU) function.
  - **Layer 2:** 64 neurons, activated using the ReLU function.
- **Output Layer:**
  - Single neuron for regression output, activated using a linear activation function.
- **Optimization and Loss Function:**
  - Adam optimizer with a learning rate of 0.001.
  - Mean Squared Error (MSE) as the loss function to minimize the prediction error.

4. **Training evaluation**

The training was made with various hyper parameter changes, and in the following table, I put two sets with their results, followed by the best set of parameters and the final result.

I have made the parameter search manually for this model, as there are not many variables, so I could track the differences between different sets.

Hyperparameter	MAE	MSE	Spearman's Correlation	Kendall's Tau
TF_IDF_max_features = 500 Epochs = 20 Batch_size = 32 Learning rate = 0.05	0.5473	0.5023	0.5506	0.3817
TF_IDF_max_features = 750 Epochs = 10 Batch_size = 32 Learning rate = 0.003	0.5304	0.4509	0.5862	0.4137
TF_IDF_max_features = 750 Epochs = 10 Batch_size = 16 Learning rate = 0.001	0.5240	0.4306	0.5896	0.4134

The public score for the first attempt was 0.369, and the private score was 0.423, while the best one had public score 0.528 and private score 0.549.

XGBoost Regressor

The second model trained for this task is **XGBoost** (Extreme Gradient Boosting), implemented through the following steps:

1. Data Preprocessing

Before giving the text data to the model, the following preprocessing steps were applied:

- **Tokenization**:
  - The raw text was split into individual words (tokens), enabling the model to process the language structure.
- **Stop Word Removal**:
  - Common used words such as "and", "the", and "is" were removed to reduce noise in the data.
- **Text Cleaning**:
  - Special characters, numbers, and extra spaces were removed.
  - Words were lemmatized (reduced to their base form), and stopwords were filtered out.
- **Word2Vec Embeddings**:
  - A Word2Vec model was trained on the entire text dataset, capturing word embeddings that encode semantic relationships between words.
- **TF-IDF Transformation**:
  - The TF-IDF vectorizer was used to assign weights to the most important terms in the text, with the top 700 features being retained based on TF-IDF scores.
- **Feature Representation**:
  - Each document was represented by combining Word2Vec embeddings with TF-IDF scores. This was done by multiplying the embeddings with the corresponding TF-IDF scores for each word in the document.

2. Feature Set Description

The feature set consists of a combination of **TF-IDF** and **Word2Vec embeddings**. The TF-IDF scores represent the importance of words in each document, while the Word2Vec embeddings capture semantic relationships between words. These features were then combined and normalized to form a unified feature set. This has led to better results than using only the first or the second representations.

In this project, I have selected the top **700 most important words** based on their TF-IDF scores to reduce the dimensionality and ensure that the most meaningful terms are used as features. The TF-IDF feature set is then used as input to the neural network for score prediction.

For the **Word2Vec** feature set, I trained a **Word2Vec model** on the entire text, which should capture the semantic relationships between words. This model represents each word as a dense vector in a high-dimensional space, where words with similar meanings are placed closer together.

To reduce the dimensionality and avoid using unnecessary features, I selected a vector size of 150 dimensions for each word embedding. This allowed the model to capture sufficient information while keeping the feature size small enough to be model that trains in a short time.

3. Model Architecture

The **XGBoost Regressor** was chosen for this task with the following hyperparameters:

- **Objective**:
  - reg:squarederror for regression tasks (squared error loss function).
- **Learning Rate**:
  - 0.005, which controls the step size in updating the weights during training.
- **Max Depth**:
  - 10, determining the maximum depth of each tree in the ensemble.
- **Number of Estimators**:
  - 500, specifying the number of boosting rounds (trees).
- **Subsample**:
  - 0.7, meaning 70% of the training data is used to grow each tree.
- **Colsample_bytree**:
  - 0.8, meaning 80% of features are sampled for each tree.
- **Lambda**:
  - 0.5, controlling L2 regularization to prevent overfitting.
- **Alpha**:

- 1, controlling L1 regularization.

4. **Training and Evaluation**

The training was made with various hyper parameter changes, and in the following table, I put two sets with their results, followed by the best set of parameters and the final result.

Initially, I have implemented a grid search to find the best parameters because there are a lot of possible sets, but after trying a few hundred random sets, I started to search manually, modifying the best one.

Hyperparameter	MAE	MSE	Spearman's Correlation	Kendall's Tau
learning_rate = 0.003, max_depth = 6, n_estimators = 300, subsample = 0.9, colsample_bytree = 0.6	0.6034	0.4967	0.5818	0.4059
learning_rate = 0.003, max_depth = 8, n_estimators = 500, subsample = 0.7, colsample_bytree = 0.8	0.5547	0.4529	0.5941	0.4162
learning_rate = 0.005, max_depth = 10, n_estimators = 500, subsample = 0.7, colsample_bytree = 0.8	0.5241	0.4295	0.6066	0.4273

The public score for the first attempt was 0.480, and the private score was 0.461, while the best one had public score 0.576 and private score 0.500.