ChatGPT

Sentiment Analysis Pipeline Explanation

Libraries and Imports

The code begins by importing several libraries. **Pandas** is imported as `pd` to handle data frames (e.g. `pd.read_csv()` reads a CSV file into a DataFrame [1]). **NumPy** (`np`) is used for numerical operations. The `re` module provides regular expressions for text cleaning. TensorFlow's **Keras** modules (`Sequential`, `Dense`, `Dropout`) are used to build the neural network. From scikit-learn, `TfidfVectorizer` converts text into TF-IDF features [2], while `train_test_split` splits data into training and testing sets [3]. Metrics like `classification_report`, `accuracy_score`, and `confusion_matrix` are imported from `sklearn.metrics` for evaluation. Finally, **Matplotlib** (`plt`) and **Seaborn** (`sns`) are imported for plotting (e.g. to draw the confusion matrix heatmap).

Loading and Inspecting Data

The dataset of IMDb reviews is loaded with Pandas: for example, `df = pd.read_csv('imdb.csv')` reads the CSV file into a DataFrame [1]. We then inspect it using:
- `df.head()`: shows the first few rows of the DataFrame [4].
- `df.info()`: prints a concise summary of the DataFrame, including the index, column names, data types, and non-null counts [5].
- `df.describe()`: computes summary statistics (count, mean, std, min, max, quartiles) for numeric columns [6].
- `df.isnull().sum()`: counts missing values in each column by checking for `NaN` [7].

These steps help verify that the data loaded correctly, shows its structure, and ensures there are no missing values before training.

Text Cleaning (`clean_text`)

The code defines a `clean_text(text)` function to preprocess each review. It uses Python's `re.sub()` to remove unwanted characters:
- `re.sub(r'<.*?>', '', text)`: removes HTML tags by replacing any `<...>` pattern with an empty string [8]. This strips out any HTML formatting.
- `re.sub('[^a-zA-Z]', ' ', text)`: replaces **non-letter characters** (anything other than `a–z` or `A–Z`) with spaces [9]. This removes punctuation, digits, and symbols, leaving only letters.
- `text.lower()`: converts all letters to lowercase, standardizing the text.

After these steps, each review contains only lowercase words separated by spaces. This clean text is then stored back into `df['review']` for feature extraction.

Feature Extraction with TF-IDF

Next, the cleaned text is transformed into numeric features using **TF-IDF (Term Frequency–Inverse Document Frequency)**. A `TfidfVectorizer` object is created (often with parameters like `max_features=5000` and `stop_words='english'`). The vectorizer tokenizes the text, builds a vocabulary, and converts each document into a TF-IDF weighted vector [2].

Key points about TF-IDF:
- **TF-IDF weighting:** TF-IDF scores each word based on its frequency in a document (Term Frequency) and how unique it is across the corpus (Inverse Document Frequency) [10]. Common words that appear in many documents get lower weight.
- `TfidfVectorizer`**:** This scikit-learn class creates a matrix of TF-IDF features [2]. Each row corresponds to a document (review) and each column to a word. The value is the TF-IDF score of that word in that document.
- **Feature parameters:** Often we remove common English stop words (`stop_words='english'`) and limit `max_features` to control dimensionality.

After fitting the vectorizer on the training text, we get a NumPy array `X` of TF-IDF features. The labels `y` are encoded as 1 for positive reviews and 0 for negative.

Splitting Data into Training and Test Sets

The feature matrix `X` and labels `y` are split into training and test subsets using `train_test_split`. For example, `train_test_split(X, y, test_size=0.2, random_state=42)` randomly divides the data: 80% for training and 20% for testing [11] [3]. The `test_size=0.2` means 20% of samples are held out for testing [11]. The `random_state` ensures that the split is reproducible. Splitting the data allows us to train the model on one subset and evaluate its performance on unseen data, which helps prevent overfitting.

Neural Network Architecture

The model is defined as a sequential feed-forward neural network. Its layers are:
- `Dense(64, activation='relu')`**:** A fully-connected layer with 64 neurons. It uses the **ReLU** activation function, which outputs the input directly if positive, otherwise outputs zero [12]. ReLU is popular because it helps models train faster and mitigate vanishing gradients.
- `Dropout(0.5)`**:** A dropout layer with rate 0.5, which randomly sets half of the layer's neurons to zero during each training step [13]. This prevents the network from overfitting by ensuring the model can't rely too heavily on any single neuron.
- `Dense(32, activation='relu')`**:** Another fully-connected hidden layer with 32 neurons and ReLU activation.
- `Dropout(0.5)`**:** Another dropout layer (rate 0.5) after the second hidden layer.
- `Dense(1, activation='sigmoid')`**:** The output layer with a single neuron and **sigmoid** activation. The sigmoid function squashes the output to a value between 0 and 1, which can be interpreted as the probability of the review being positive [14]. Sigmoid is used here because this is a binary classification problem (positive vs negative sentiment).

Each `Dense` layer is a fully-connected layer (all inputs connect to all outputs). The use of ReLU in hidden layers and sigmoid in the output is a standard choice for binary classification networks.

Compiling the Model

After defining the architecture, the model is compiled with a loss function and optimizer. The code uses `loss='binary_crossentropy'` and `optimizer='adam'`. **Binary cross-entropy** is the appropriate loss for binary classification [15]: it measures how well the predicted probabilities match the true labels (0 or 1). Lower cross-entropy means better predictions. The **Adam optimizer** (Adaptive Moment Estimation) is a widely used gradient descent algorithm that adaptively adjusts learning rates for each parameter [16]. Adam is popular because it is efficient and often leads to good convergence in deep learning. We also specify `metrics=['accuracy']` so that training will report the accuracy on the training/validation sets each epoch.

Training the Model

The model is trained with `model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)`. This runs 10 training epochs, processing data in batches of 32 samples. The `validation_split=0.2` argument means that 20% of the *training* data is set aside as a validation set [17]. In other words, each epoch trains on 80% of the training data and evaluates on the remaining 20%, providing a quick check of the model's performance on unseen data during training. The `fit()` method returns a `History` object (often saved as `history`) that contains the loss and accuracy for each epoch on both the training and validation sets. These metrics can be plotted later to visualize learning progress.

Predictions and Thresholding

After training, we get the predicted probabilities on the test set by `y_pred_prob = model.predict(X_test)`. This yields values between 0 and 1 (due to the sigmoid output). To convert these probabilities into class labels, we apply a threshold: `y_pred = (y_pred_prob > 0.5).astype(int)`. This means any probability above 0.5 is classified as positive (1), otherwise negative (0). Thresholding at 0.5 is a natural choice for sigmoid output, since sigmoid outputs represent estimated probabilities [14]. The result `y_pred` is an array of 0s and 1s corresponding to negative/positive predictions for each test review.

Evaluation Metrics

We then evaluate the model's predictions using several metrics:
- **Classification report:** `classification_report(y_test, y_pred)` prints precision, recall, F1-score, and support for each class (negative vs positive) [18]. This gives a detailed view of how many positive reviews were correctly identified (recall), how many of the predicted positives were correct (precision), and their harmonic mean (F1-score).
- **Accuracy:** `accuracy_score(y_test, y_pred)` computes the overall fraction of correct predictions [19]. This is simply the number of correct labels divided by the total number of samples.
- **Confusion matrix:** `confusion_matrix(y_test, y_pred)` produces a 2×2 table showing counts of true negatives, false positives, false negatives, and true positives [20]. This matrix summarizes the raw

performance: how many of each actual class were predicted correctly or incorrectly. We often visualize the confusion matrix as a heatmap (using `sns.heatmap`) to easily see where the model is making mistakes. A heatmap highlights larger values in darker colors, so it's easier to spot, for instance, if many negatives are mislabeled as positives or vice versa [21].

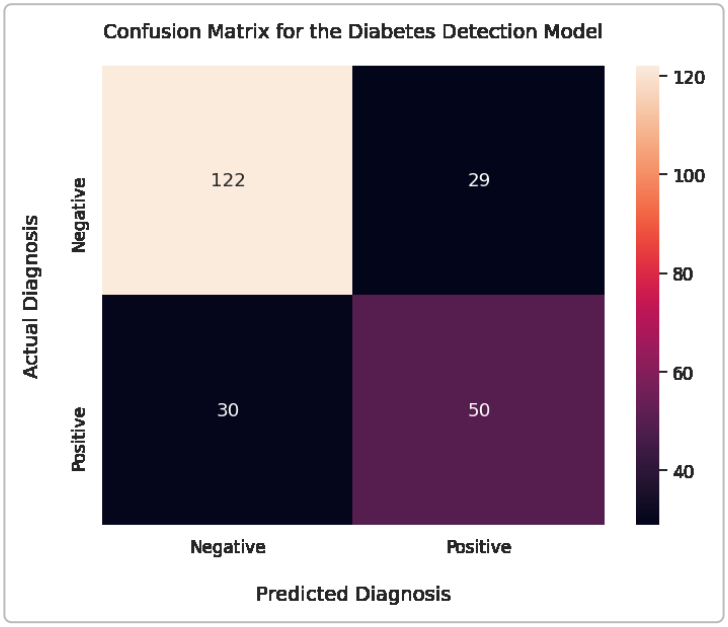

Figure: Example confusion matrix heatmap for a binary classification task. The confusion matrix compares predicted labels (columns) to actual labels (rows) [20]. Each cell shows the count of reviews in that category. For binary sentiment (e.g. "Negative" vs "Positive"), the top-left cell is True Negatives, top-right is False Positives, bottom-left is False Negatives, and bottom-right is True Positives. A heatmap coloring makes it easy to see which cells have high counts. Here, darker off-diagonal cells indicate higher misclassification rates. In practice, this visual tool helps us quickly identify which classes are often confused by the model [22] [21], complementing the numerical accuracy and F1-scores.

Figure: A confusion matrix heatmap shows actual vs predicted classes [22] [20]. The diagonal cells (top-left and bottom-right) are correct predictions; off-diagonal cells are errors. The heatmap's color intensity highlights larger counts, making it easy to spot patterns of errors [21].

---

1. **pandas.read_csv — pandas 2.2.3 documentation**
   https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
2. **TfidfVectorizer — scikit-learn 1.6.1 documentation**
   https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
3. **train_test_split — scikit-learn 1.6.1 documentation**
   https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
4. **pandas.DataFrame.head — pandas 2.2.3 documentation - PyData |**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html
5. **pandas.DataFrame.info — pandas 2.2.3 documentation - PyData |**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html
6. **pandas.DataFrame.describe — pandas 2.2.3 documentation**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
7. **pandas.DataFrame.isnull — pandas 2.2.3 documentation**
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isnull.html
8. **How to Remove HTML Tags from String in Python | GeeksforGeeks**
   https://www.geeksforgeeks.org/how-to-remove-html-tags-from-string-in-python/
9. **java - Replacing all non-alphanumeric characters with empty strings - Stack Overflow**
   https://stackoverflow.com/questions/1805518/replacing-all-non-alphanumeric-characters-with-empty-strings
10. **Understanding TF-IDF (Term Frequency-Inverse Document Frequency) | GeeksforGeeks**
    https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
11. **Split Your Dataset With scikit-learn's train_test_split() – Real Python**
    https://realpython.com/train-test-split-python-data/
12. **A Gentle Introduction to the Rectified Linear Unit (ReLU) - MachineLearningMastery.com**
    https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
13. **Dropout layer**
    https://keras.io/api/layers/regularization_layers/dropout/
14. **deep learning - What are the best activation functions for Binary text classification in neural networks? - Data Science Stack Exchange**
    https://datascience.stackexchange.com/questions/56830/what-are-the-best-activation-functions-for-binary-text-classification-in-neural
15. **Binary Cross Entropy/Log Loss for Binary Classification | GeeksforGeeks**
    https://www.geeksforgeeks.org/binary-cross-entropy-log-loss-for-binary-classification/
16. **Adam**
    https://keras.io/api/optimizers/adam/
17. **data - How does the validation_split parameter of Keras' fit function work? - Data Science Stack Exchange**
    https://datascience.stackexchange.com/questions/38955/how-does-the-validation-split-parameter-of-keras-fit-function-work
18. **classification_report — scikit-learn 1.6.1 documentation**
    https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
19. **accuracy_score — scikit-learn 1.6.1 documentation**
    https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

20 **confusion_matrix — scikit-learn 1.6.1 documentation**
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

21 22 **Confusion Matrix: How To Use It & Interpret Results [Examples]**
https://www.v7labs.com/blog/confusion-matrix-guide