Text Classification

Task 1: Data Loading

For the following project I use Pytesseract OCR model to extract the text images from the images. Pytesseract is a widely used optical character recognition (OCR) tool in Python. It is a wrapper for Google's Tesseract-OCR engine and can be used to extract text from images or scanned documents.

There are two main executable files present in the root directory:

- The main_image.py file performs evaluation of the ML models after converting scanned document images into text. The resulting text is stored in parquet files and undergoes preprocessing and embedding extraction before being fed into the models for training and prediction.

  Command: python main_image.py
  Results Path: …/results/image2text_file_result.txt

- Main_text.py: The main purpose of this file is to evaluate the performance of the ML models. It achieves this by first reading all the provided text files and storing them in parquet files for preprocessing and embeddings extraction. The resulting data is then fed to our models to train and predict their accuracy.

  Command: python main_text.py
  Results Path: …/results/text_file_result.txt

During the evaluation, it was observed that there was no significant difference in the accuracies of the two datasets. This leads me to believe that Pytesseract accurately detected and extracted text from the images. However, it's not solely responsible for the high accuracy provided by the models. Accurate feature embedding also played a crucial role. Although the difference in accuracy was low, the dataset created from text files provided higher accuracy for text classification models. The dataframes were stored in Parquet files as they are fast and occupy less space compared to other relational database formats.

Task 2: Data Preprocessing and Feature Extraction

In this project, I used two methods to extract features from each text. TF-IDF vectorizer is a well-performing method for text classification applications. However, in this case, we were not sure if a word in the document has any semantic relation with other words, which is responsible for its respective class label.
Therefore, we used another feature extraction method, which is getting embeddings using the Word2Vec model. TF-IDF is used to understand the importance of words in a document, while Word2Vec is used to understand the meaning of words and their relationships to other words.

From the results, we observed that TF-IDF vectorizer performs better in extracting rich information from the documents for text classification when compared to the Word2Vec

vectorizer. It is worth noting that the feature space of Word2Vec was selected to be just 128 less than that of the TF-IDF vectors, which is more than 1000. Increasing the feature space for the Word2Vec models might yield fruitful results. However, for the following assignment, we kept things straightforward and simple.

Task 3: Model Evaluation

Both machine learning (ML) and neural network (NN) models can be used for text classification, and the choice of which to use depends on the embedding vectors available and the size of the dataset.

For simpler text classification tasks with a relatively small dataset, traditional ML models such as Naive Bayes, SVM, or Logistic Regression are preferred as they work well and are computationally efficient. These models also work well with TF-IDF vectors as they do not capture the contextual meaning of text and rely purely on probabilities.

On the other hand, vectorizers like Word2Vec work better with NN models. With convolutional filters, these models extract the hidden information or context of the text. However, the downside of NN models is that they are computationally expensive and time-consuming, a trade-off for resources. In the following project, two simple NN architectures were used, one being a simple 1D convolutional neural network and the other being a bidirectional LSTM.

Since the neural network architecture was very basic and the feature space was small, these models failed to grasp the contextual meaning in the document and performed poorly for text classification. Even after hyperparameter tuning, their accuracy improved from 0.50 to 0.58. Surprisingly, the ML models performed significantly better, with accuracies of 0.75+. Of all three ML models, the Support Vector Machines Classifier performed the best.

Review Report:

Key points from the paragraph:

- Task 1: Pytesseract OCR model was used to extract text from images and scanned documents. Two main executable files are present in the root directory: main_image.py and main_text.py. Dataframes were stored in Parquet files for fast and low space consumption.
- Task 2: Two ways were used for feature extraction: TF-IDF vectorizer and word2vec model. TF-IDF vectorizer performs better for text classification in this project. Traditional ML models like Naive Bayes, SVM, or Logistic Regression work well with TF-IDF vectors, while word2vec works better with neural network models.
- Task 3: The choice of ML or NN models for text classification depends on the embedding vectors and dataset size. For simpler text classification tasks with a relatively small dataset, traditional ML models are preferred. For larger datasets, neural network models are more suitable. The two simple NN architectures used in the project were a 1D convolutional neural network and bidirectional LSTM. However, the ML models performed significantly better in this project, with Support Vector Machines Classifier performing the best.