Experiment report: TextAnalysis

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**
Course: **Big Data Analysis Technology**
Date: **October 2024**

Task

1. Learn and understand the word embedding, draw a diagram to show the mechanism and work flow of word embedding;
2. Finish the experiment and write the experiment report.

Word embedding

Word embedding is a technique which converts a text into a numerical representation, mapping each word to a vector. This vector captures semantic meaning and distinctive features of every word. The main feature is that the closer words are in the vector space, more similar they are expected to be in meaning. This method is widely used in text analysis.

The work principle is shown in the diagram below.

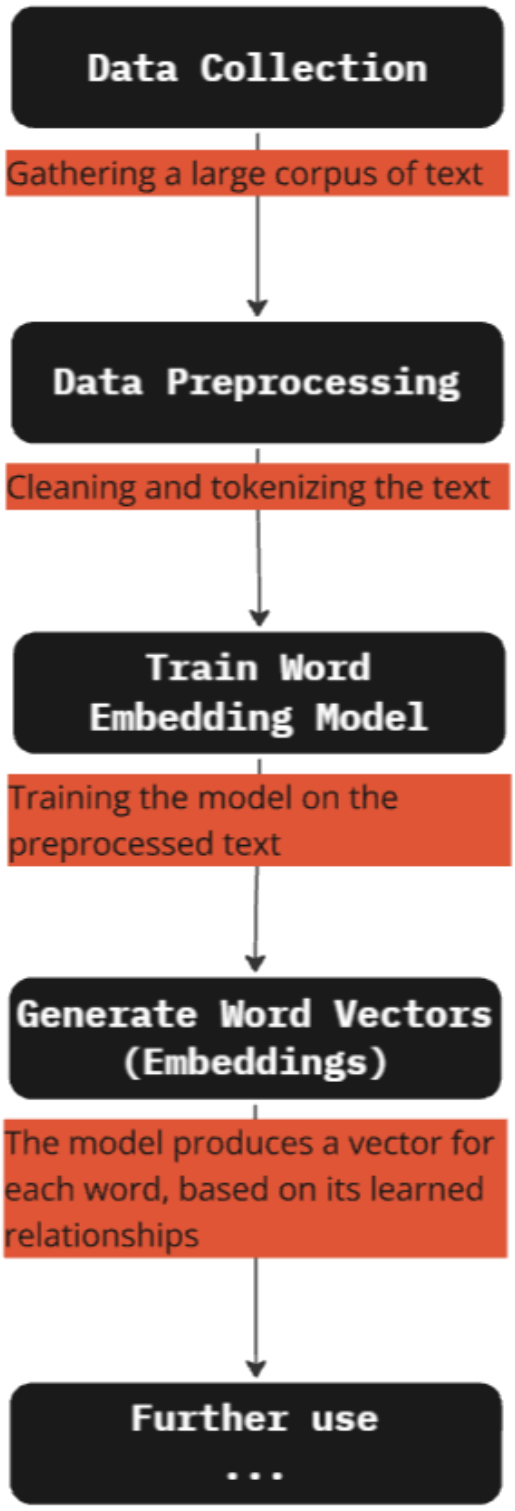

Figure 1: Diagram of word embedding

Code

```
import tensorflow as tf
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import time
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import movie_reviews
import tensorflow as tf

# Tokenizer from Keras, used to convert text into sequences
tokenizer = tf.keras.preprocessing.text.Tokenizer()

# Using Keras API components from TensorFlow for model creation
pad_sequences = tf.keras.preprocessing.sequence.pad_sequences
Embedding = tf.keras.layers.Embedding
LSTM = tf.keras.layers.LSTM
Dense = tf.keras.layers.Dense
Sequential = tf.keras.models.Sequential

# Downloading movie reviews dataset from NLTK
# nltk.download('movie_reviews')

# Converting the raw text data into a structured DataFrame with two columns: 'text' and 'label'
# 'text' contains the review, and 'label' is the sentiment (positive/negative)
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
df = pd.DataFrame(documents, columns=['text', 'label'])
# Mapping labels to binary values (1 for positive, 0 for negative)
df['label'] = df['label'].map({'pos': 1, 'neg': 0})
# Joining the review words into a single string
df['text'] = df['text'].apply(lambda x: ' '.join(x))

# TF-IDF vectorization: transforming the text data into numerical features
vectorizer = TfidfVectorizer(max_features=5000)  # Using max 5000 features
# Transforming text into TF-IDF feature vectors
X_tfidf = vectorizer.fit_transform(df['text']).toarray()
y = df['label'].values  # Extracting labels

# Splitting the data into training and testing sets
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(
    X_tfidf, y, test_size=0.2, random_state=42)

# Training a Logistic Regression model using the TF-IDF features
start_time = time.time()
model_tfidf = LogisticRegression()  # Initializing the model
model_tfidf.fit(X_train_tfidf, y_train)  # Training the model
```

```
training_time_tfidf = time.time() - start_time  # Calculating training time
# Making predictions on the test set
y_pred_tfidf = model_tfidf.predict(X_test_tfidf)
accuracy_tfidf = accuracy_score(
    y_test, y_pred_tfidf)  # Calculating the accuracy
print(f"Accuracy of TF-IDF model: {accuracy_tfidf*100:.2f}%")

# Tokenizing the text for LSTM
tokenizer = tf.keras.preprocessing.text.Tokenizer(
    num_words=5000)  # Using max 5000 words
tokenizer.fit_on_texts(df['text'])  # Fitting the tokenizer on the dataset
# Converting text to sequences of integers
X_seq = tokenizer.texts_to_sequences(df['text'])
X_pad = tf.keras.preprocessing.sequence.pad_sequences(
    X_seq, maxlen=500)  # Padding sequences to ensure equal length

# Loading GloVe embeddings for the LSTM model
embedding_index = {}
with open('glove.6B.100d.txt', 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]  # The word
        coefs = np.asarray(values[1:], dtype='float32')  # The embedding vector
        embedding_index[word] = coefs  # Storing the word and its vector

# Creating an embedding matrix where each word is represented by its GloVe embedding
word_index = tokenizer.word_index
# Matrix size of (vocab_size, embedding_dim)
embedding_matrix = np.zeros((5000, 100))
for word, i in word_index.items():
    if i < 5000:
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            # Filling in the embedding matrix
            embedding_matrix[i] = embedding_vector

# Building the LSTM model
model_lstm = Sequential()  # Initializing a sequential model
model_lstm.add(Embedding(input_dim=5000, output_dim=100, weights=[
               embedding_matrix], trainable=False))  # Embedding layer with GloVe weights
# LSTM layer with dropout for regularization
model_lstm.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
# Dense output layer with sigmoid activation for binary classification
model_lstm.add(Dense(1, activation='sigmoid'))

# Compiling the LSTM model
model_lstm.compile(loss='binary_crossentropy',
                   optimizer='adam', metrics=['accuracy'])

# Splitting the padded sequences into training and testing sets
X_train_pad, X_test_pad, y_train, y_test = train_test_split(
    X_pad, y, test_size=0.2, random_state=42)
```

```
104
105 # Training the LSTM model
106 start_time = time.time()
107 model_lstm.fit(X_train_pad, y_train, validation_data=(
108     X_test_pad, y_test), epochs=15, batch_size=128)
109     # X_test_pad, y_test), epochs=3, batch_size=128)
110 # Calculating training time for LSTM
111 training_time_lstm = time.time() - start_time
112 # Evaluating the LSTM model
113 scores = model_lstm.evaluate(X_test_pad, y_test, verbose=0)
114 accuracy_lstm = scores[1]  # Extracting accuracy
115 print(f"Accuracy of LSTM: {accuracy_lstm*100:.2f}%")
116
117 # Results comparison between TF-IDF + Logistic Regression and LSTM + GloVe
118 results = pd.DataFrame({
119     'Model': ['TF-IDF + Logistic Regression', 'LSTM + GloVe'],
120     'Accuracy': [accuracy_tfidf, accuracy_lstm],
121     'Training Time (seconds)': [training_time_tfidf, training_time_lstm]
122 })
123
124 print(results)
125
126 # Plotting accuracy comparison
127 plt.figure(figsize=(10, 5))
128 plt.bar(results['Model'], results['Accuracy'], color=['blue', 'green'])
129 plt.title('Accuracy Comparison')
130 plt.ylabel('Accuracy')
131 plt.show()
132
133 # Plotting training time comparison
134 plt.figure(figsize=(10, 5))
135 plt.bar(results['Model'], results['Training Time (seconds)'],
136         color=['blue', 'green'])
137 plt.title('Training Time Comparison')
138 plt.ylabel('Training Time (seconds)')
139 plt.show()
```

Experiment

In this experiment after the code execution we get the accuracy and training time output for each model (TF-IDF+ Logistic regression and LSTM + GloVe). The accuracy of TF-IDF: 80.75%.

At first, I ran the code with the number of epochs = 15. The accuracy of LSTM for this case is 75.00%. It is shown that the accuracy is higher using TF-IDF + Logistic Regression than LSTM + GloVe. But the training time is much higher for LSTM + GloVe.

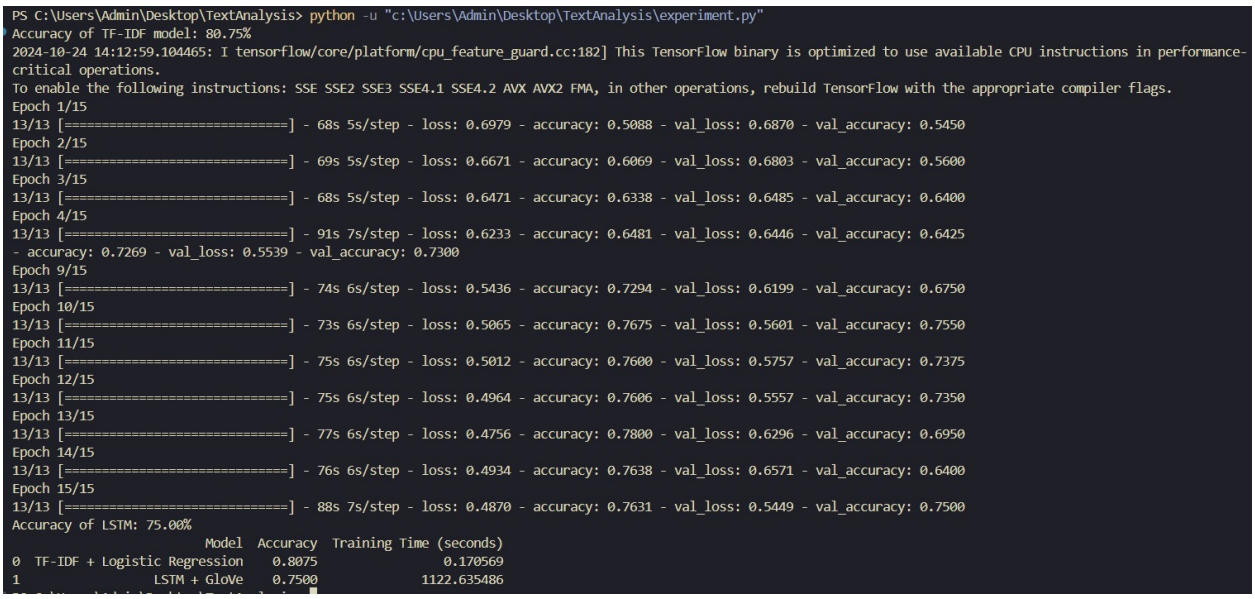

Figure 2: Code execution: 15 epochs

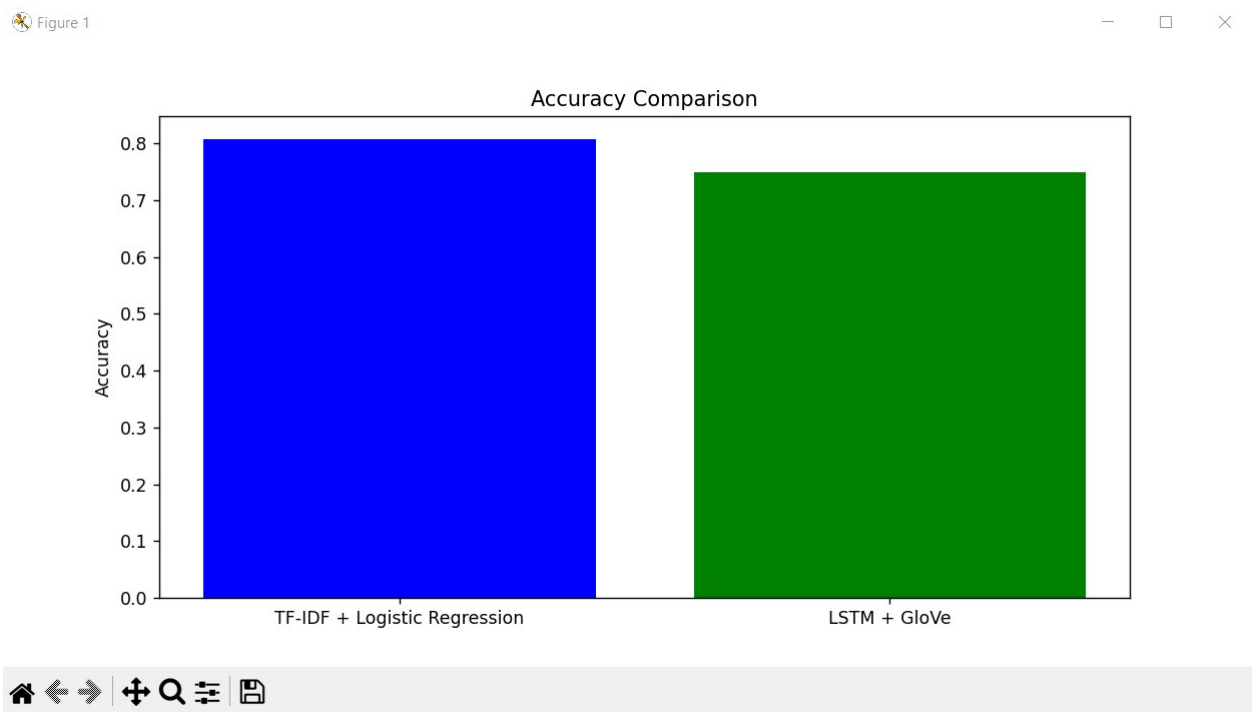

Figure 3: Accuracy comparison: 15 epochs

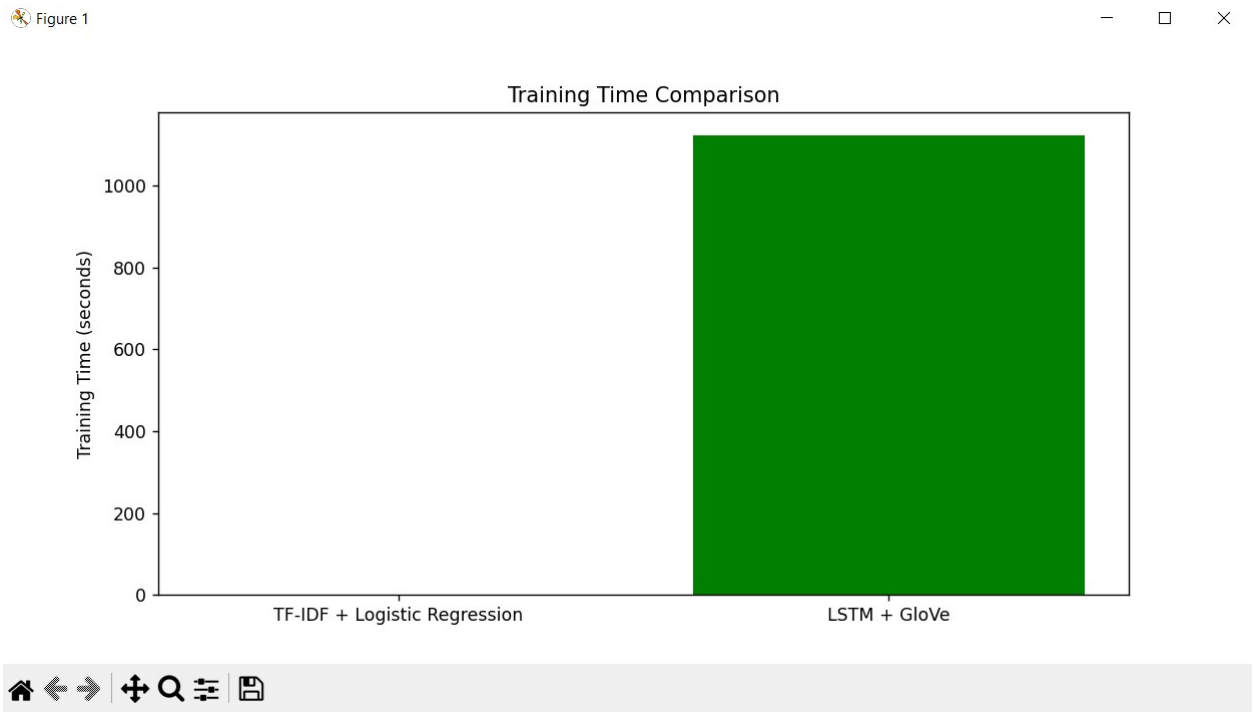

Figure 4: Training time comparison: 15 epochs

Then I ran the code with the number of epochs = 3. The accuracy of LSTM for this case is 61.00%. The situations with the accuracy and the training time is the same.

```
PS C:\Users\Admin\Desktop\TextAnalysis> python -u "c:\Users\Admin\Desktop\TextAnalysis\experiment.py"
Accuracy of TF-IDF model: 80.75%
2024-10-24 14:34:53.687950: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perfor
mance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/3
13/13 [==============================] - 58s 4s/step - loss: 0.6869 - accuracy: 0.5394 - val_loss: 0.6822 - val_accuracy: 0.5375
Epoch 2/3
13/13 [==============================] - 56s 4s/step - loss: 0.6630 - accuracy: 0.6006 - val_loss: 0.6674 - val_accuracy: 0.5800
Epoch 3/3
13/13 [==============================] - 53s 4s/step - loss: 0.6478 - accuracy: 0.6244 - val_loss: 0.6517 - val_accuracy: 0.6100
Accuracy of LSTM: 61.00%
                      Model  Accuracy  Training Time (seconds)
0  TF-IDF + Logistic Regression    0.8075                0.174005
1                 LSTM + GloVe    0.6100              168.287405
PS C:\Users\Admin\Desktop\TextAnalysis>
```

Figure 5: Code execution: 3 epochs

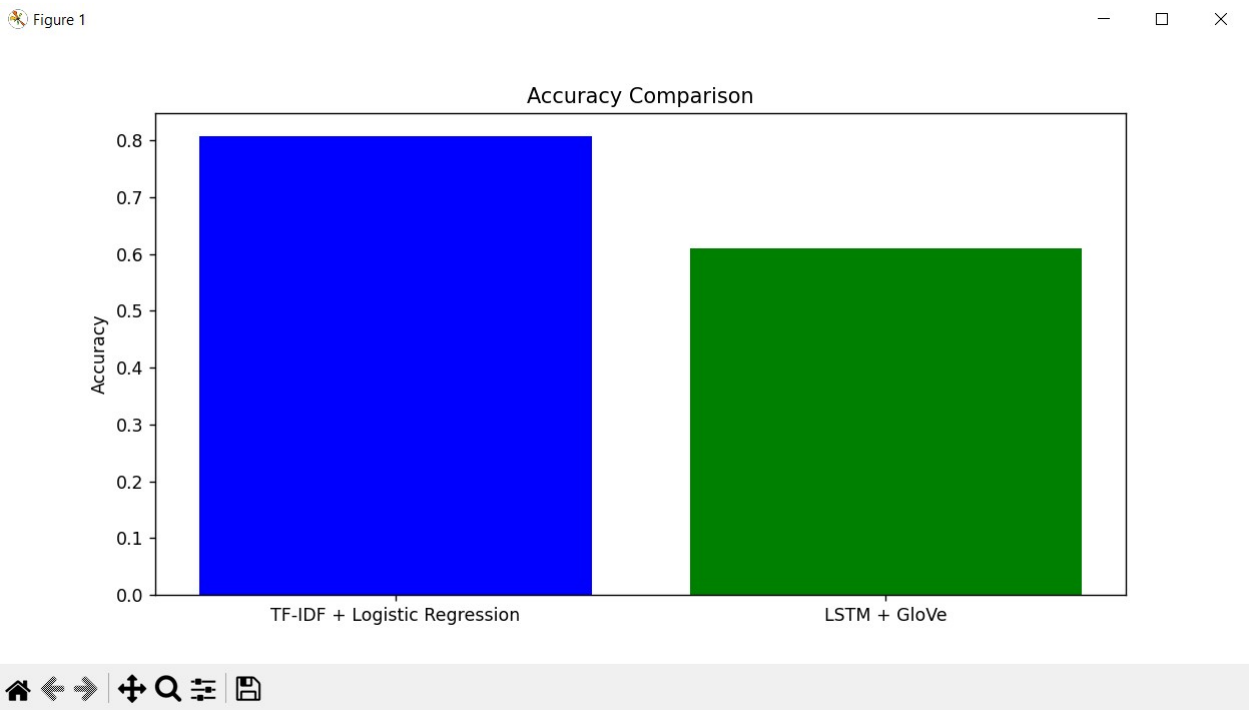

Figure 6: Accuracy comparison: 3 epochs

Figure 7: Training time comparison: 3 epochs