**Original Source File:**

The original source file is an excel worksheet with a list of the text similar to this one:

*“The first ABC Corp - Date: 2023-01-05 - Vendor: ABC Corporation - Invoice Number: INV001 - Items/Services: Round-trip Flight to New York, Hotel Accommodation (4 nights), Meals (Per Diem), Transportation (Taxi, Subway) - Total Amount: $2,500.00 - Contact: John Doe, johndoe@email.com - Bill-To: XYZ Inc - Payment Due Date: 2023-01-20, Travel”*

**Read file:**

import pandas as pd

# Specify the full path to the dataset.csv file

csv\_file\_path = "C:/path/to/your/file/dataset.csv"

# Load the CSV file into a DataFrame

df = pd.read\_csv(csv\_file\_path)

# Split the 'Original Column' into 'Text of Invoice' and 'Expense Category' columns

df[['Text of Invoice', 'Expense Category']] = df['Original Column'].str.split(',', 1, expand=True)

# Create a list of tuples representing your dataset

dataset = [(row['Text of Invoice'], row['Expense Category']) for \_, row in df.iterrows()]

# Now, 'dataset' is a list of tuples where each tuple contains the text and

**Other code:**

import torch

from transformers import BertTokenizer, BertForSequenceClassification, AdamW

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, accuracy\_score

# Load and preprocess your data here

# Assuming you have a list of text descriptions 'text\_data' and corresponding labels 'labels'

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(text\_data, labels, test\_size=0.2, random\_state=42)

# Load the pre-trained BERT model and tokenizer

model\_name = "bert-base-uncased" # You can choose different variations of BERT

tokenizer = BertTokenizer.from\_pretrained(model\_name)

model = BertForSequenceClassification.from\_pretrained(model\_name, num\_labels=len(set(labels)))

# Tokenize and encode the text data

train\_encodings = tokenizer(X\_train, truncation=True, padding=True, return\_tensors='pt', max\_length=512)

test\_encodings = tokenizer(X\_test, truncation=True, padding=True, return\_tensors='pt', max\_length=512)

# Create PyTorch datasets

train\_dataset = torch.utils.data.TensorDataset(train\_encodings.input\_ids, train\_encodings.attention\_mask, torch.tensor(y\_train))

test\_dataset = torch.utils.data.TensorDataset(test\_encodings.input\_ids, test\_encodings.attention\_mask, torch.tensor(y\_test))

# Define data loaders

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=8, shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=8, shuffle=False)

# Define the optimizer and loss function

optimizer = AdamW(model.parameters(), lr=1e-5)

loss\_fn = torch.nn.CrossEntropyLoss()

# Training loop

num\_epochs = 3

for epoch in range(num\_epochs):

model.train()

for batch in train\_loader:

input\_ids, attention\_mask, labels = batch

optimizer.zero\_grad()

output = model(input\_ids, attention\_mask=attention\_mask)[0]

loss = loss\_fn(output, labels)

loss.backward()

optimizer.step()

# Evaluation

model.eval()

predictions = []

with torch.no\_grad():

for batch in test\_loader:

input\_ids, attention\_mask, \_ = batch

output = model(input\_ids, attention\_mask=attention\_mask)[0]

\_, predicted = output.max(1)

predictions.extend(predicted.tolist())

# Calculate metrics

print("Accuracy:", accuracy\_score(y\_test, predictions))

print(classification\_report(y\_test, predictions))

In this code:

We load a pre-trained BERT model for text classification.

Tokenize and encode the text data.

Create PyTorch datasets and data loaders for training and testing.

Define the optimizer and loss function.

Train the BERT model on your training data.

Evaluate the model's performance on the test data using accuracy and a classification report.

Make sure to preprocess and load your data according to your specific dataset. You can fine-tune the BERT model further if needed. Additionally, you can adjust hyperparameters and experiment with different pre-trained models to achieve the best results for your text classification task.

**Features are embedded by BERT. BERT, as a deep neural network, automatically learns the weights during the training process, and the features are embedded representations of the input text.automatically:**

Here's how this works:

1. Feature Extraction:

In this example, BERT tokenizes and encodes the input text data into numerical features. These features are the dense vector representations of the words and subwords in your text data.

2. Model Training:

The BERT model you use in the example is already pre-trained on a large corpus of text data, which means it has learned the relationships and patterns in the data. During fine-tuning on your specific text classification task, BERT's weights are adjusted to fit your data.

3. Weight (w):

The weights (denoted as "w") in deep neural networks like BERT are the learnable parameters of the model. These weights are automatically adjusted during the training process to minimize the loss (error) between the predicted labels and the actual labels in your training data.

Here's how you can access and understand the weights in a BERT model:

python

Copy code

# Access BERT model weights

bert\_weights = model.state\_dict()

# You can inspect the learned weights, but they are typically very large and not human-interpretable.

# For example, you can view the weights of the first layer like this:

print("Weights of the first layer of BERT:")

print(bert\_weights['bert.embeddings.word\_embeddings.weight'])

However, manually calculating and setting these weights is generally not done in practice. The power of deep learning models like BERT lies in their ability to learn the appropriate weights during training by optimizing a loss function. The fine-tuning process automatically adjusts the weights based on the gradients, backpropagation, and the specified loss function to make predictions that match the provided labels.

In summary, you don't need to manually calculate the weights or features when using pre-trained deep learning models like BERT. The model automatically learns the weights during training, and the features are derived from the input text data as dense embeddings.

**Features generated by BERT model:**

The embedded features generated by the BERT model are dense vector representations of the input text data. These embeddings capture the semantic meaning of words, subwords, and their context within the text. BERT uses a bidirectional context window to understand how each word or subword relates to the words around it.

Each token in the input text is mapped to a high-dimensional vector in the embedding space. The specific dimensionality of the embeddings depends on the BERT model variant you are using (e.g., BERT-base has 768-dimensional embeddings, while BERT-large has 1024-dimensional embeddings).

The embedded features typically have the following characteristics:

High Dimensionality: BERT embeddings have a relatively high dimensionality (e.g., 768 or 1024 dimensions), which allows them to capture fine-grained semantic information.

Dense Vectors: These are dense vectors in contrast to one-hot encodings. Each dimension of the vector contains a real-valued number, allowing BERT to capture the relationships and nuances between words.

Contextual Information: BERT embeddings are contextual, meaning the same word may have different embeddings depending on its context within the sentence. This allows BERT to understand word polysemy (multiple meanings of a word).

Pre-trained Knowledge: BERT embeddings are pre-trained on a large corpus of text data, giving them knowledge of the world and common language usage.

Here's an example of what the BERT embeddings for a simple sentence might look like:

Input Sentence: "I love machine learning."

BERT Embeddings (Simplified for Illustration):

"I" → [0.12, 0.45, -0.67, ...] (768-dimensional vector)

"love" → [0.56, -0.23, 0.78, ...] (768-dimensional vector)

"machine" → [0.31, 0.87, -0.42, ...] (768-dimensional vector)

"learning" → [0.71, -0.62, 0.95, ...] (768-dimensional vector)

These embeddings are produced by BERT as part of the tokenization and encoding process. They are the numerical features that the model uses to understand and make predictions about the text data. The embeddings contain information about the meaning and context of the words in the sentence, which is crucial for tasks like text classification.