Implementation Description

Consider the Snappfood dataset provided in the attachment. This dataset contains user reviews about active restaurants in this collection, which have been labeled as positive or negative in one of the corresponding classes.

1. Importing Required Libraries:

```
# Import required packages

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score, ConfusionMatrixDisplay
from sklearn.utils import shuffle

from transformers import BertConfig, BertTokenizer
from transformers import BertModel

from transformers import AdamW
from transformers import get_linear_schedule_with_warmup

import torch
import torch.nn as nn
import torch.nn.functional as F

import hazm
from cleantext import clean

import matplotlib.pyplot as plt

from tqdm.notebook import tqdm

import os
import re
import json
import copy
import collections
```

We add the necessary libraries to import the dataset, perform preprocessing on the data, and build and train a neural network for the project. A brief description of each library used is provided below:

- Numpy: A well-known and useful library for performing fast and parallel numerical computations.

- Pandas: Used for reading and preparing datasets.

- Sklearn: A machine learning library that provides various tools for classification, regression, and clustering tasks.

- Transformers: A library for natural language processing that provides pre-trained models such as BERT.

- Torch: The main library used for building and training neural networks.

- nn: A sub-library that provides implementations of various neural network architectures and loss functions.

- F: A PyTorch sub-library that provides various activation functions and other useful functions.

- Hazm: A library for natural language processing that provides tools for tokenization, normalization, and other language-specific tasks related to Persian text.

- Cleantext: A library for cleaning or organizing textual data by removing unwanted characters, stop words, URLs, and other patterns.

- Matplotlib: A library for creating plots.

Other libraries used during the implementation process include:

- Optim: A subset of PyTorch that provides various optimization algorithms for training neural networks.

- DataLoader: A PyTorch class that loads data from a dataset and provides batches to a neural network.

-Tqdm: A library for creating progress bars in Python.

1. Preprocessing:

1.1- What are the necessary preprocessing steps for the given task? Name them and perform each step to prepare the data for input to the model.

The list of preprocessing steps performed in order:

- In each row of the dataset, each review was mistakenly stored in two cells of the CSV file. This problem was fixed when reading the dataset.

- To increase the model's accuracy, we use data augmentation techniques.

- Reviewing the length of customer reviews and finding the truncation limit. As mentioned in the descriptive questions section, padding and truncation help the model manage inputs with different lengths for better evaluation results.

- Preprocessing steps related to Persian texts, such as managing spaces, half-spaces, Persian numbers, right-to-left correction, etc.

- Removing characters related to Emojis, Symbols, Flags, and #

2- Importing the Dataset into the Project Environment:

```
# Upload Snappfood dataset and load Data

!unzip Snappfood-Dataset.zip

def load_csv(file_path):
    data_df = pd.DataFrame(columns=['comment', 'label', 'label_id'])
    print(f'Loading {file_path}')
    with open(file_path) as f:
        for line in tqdm(f.readlines()[1:]):
            line = line.strip().replace('\t', ',').split(',')
            line = [part for part in line if len(part) != 0]
            if len(line) > 4:
                line[1:-2] = [' , '.join(line[1:-2])]

            data_df.loc[int(line[0])] = line[1:]

    print('Done.')
    return data_df


train_df = load_csv('Snappfood-Dataset/train.csv')
valid_df = load_csv('Snappfood-Dataset/dev.csv')
test_df = load_csv('Snappfood-Dataset/test.csv')
```

In this code snippet, we first load the dataset file and then extract it from the compressed state. Then, each row is examined separately using the load_csv() function, and information such as row number, review, and label are saved in the corresponding variables, train_df, valid_df, and test_df. In some rows, each review was mistakenly stored in two cells of the CSV file. To perform preprocessing related to this part, we consider the first column as the row number, the two last columns as the label, and the middle part as the customer reviews.

3- Implementation of Data Augmentation:

```
!pip install PersianNLPAug

import persiannlpaug.augmenter.char as pna

# Define augmenter
aug = pna.NoiseAug()

# Define a function to augment comments in a data frame
def augment_comments(df):
    # Create a copy of the data frame
    augmented_df = df.copy()

    # Augment comments
    for i, row in augmented_df.iterrows():
        comment = row['comment']
        augmented_comment = aug.augment(comment)
        augmented_df.at[i, 'comment'] = augmented_comment

    return augmented_df

# Load data frames
train_df = load_csv('Snappfood-Dataset/train.csv')
valid_df = load_csv('Snappfood-Dataset/dev.csv')

# Augment comments in the data frames
train_df = augment_comments(train_df)
valid_df = augment_comments(valid_df)
```

First, we install the PersianNLPAug library to perform data augmentation on Persian language text data. Then, we use it to enhance the existing data in the variables of the training and validation data or add new data to them. Using this method in the learning process helps the model achieve better accuracy in the evaluation process.

4- Checking the Truncation Limit and Truncation Operation:

```
#Preprocessing

train_df = train_df[['comment', 'label_id']]
valid_df = valid_df[['comment', 'label_id']]
test_df = test_df[['comment', 'label_id']]

def preprocess(data):
    data_preprocessed = data.copy()
    data_preprocessed['comment'] = data['comment'].apply(hazm.word_tokenize)
    data_preprocessed['comment_len'] = data_preprocessed['comment'].apply(len)

    plt.hist(data_preprocessed['comment_len'], bins=100)

preprocess(train_df)
```

First, we keep only the columns related to customer reviews and their labels in the variables train_df, valid_df, and test_df, and we do not need columns such as row number or label in words (Happy-Sad), so we remove them. Then, in the prepocess() function, we tokenize the words in the dataset, and then plot the frequency distribution of different word lengths.

```
#Preprocessing

COMMENT_MAX_LEN=50

def preprocess(data):
    data_preprocessed = data.copy()
    data_preprocessed['comment'] = data['comment'].apply(lambda x: hazm.word_tokenize(x)[:COMMENT_MAX_LEN])
    data_preprocessed['comment_len'] = data_preprocessed['comment'].apply(len)

    plt.hist(data_preprocessed['comment_len'], bins=100)

preprocess(train_df)
```

In this section, after performing the necessary checks, the appropriate maximum length for customer reviews to enter the model was found to be 50. In the above code snippet, the truncation operation is performed on sentences with a length greater than 50, and then the frequency distribution of different sentence lengths is plotted again. Naturally, in this plot, the maximum sentence length is equal to 50.

5- Preprocessing for Persian Texts:

```python
def preprocess(data):
    def cleaning(text):
        # regular cleaning
        normalizer = hazm.Normalizer()
        text_new = normalizer.normalize(text)

        wierd_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u'\U00010000-\U0010ffff'
        u"\u200d"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\u3030"
        u"\ufe0f"
        u"\u2069"
        u"\u2066"
        u"\u200c"
        u"\u2068"
        u"\u2067"
        "]+", flags=re.UNICODE)

        text = wierd_pattern.sub(r'', text)

        # removing extra spaces, hashtags
        text = re.sub("#", "", text)
        text = re.sub("\s+", " ", text)

        return text

    data['comment'] = data['comment'].apply(cleaning)
    return data

train_df = preprocess(train_df)
valid_df = preprocess(valid_df)
test_df = preprocess(test_df)
```

In the above code snippet, the Hazm library is used to perform preprocessing steps for Persian texts. These steps include managing spaces, half-spaces, Persian numbers, right-to-left correction, and more. Additionally, characters related to Emojis, Symbols, Flags, and # are removed in the following steps.

2- Checking the Data Balance and Drawing the Distribution Plot:

```
plt.hist([train_df['label_id'], valid_df['label_id'], test_df['label_id']], label=['Train', 'Validation', 'Test'])
plt.legend()
plt.xticks([0.05, 0.95], id_to_label)
```

Using the above code snippet, we draw the distribution plot of each class's samples. Based on the plotted graph, we can examine the balance of the dataset in the training, validation, and evaluation sets. You can see the results of this step in section 3, or in the Results Analysis section.

1- Building the Model:

To analyze sentiments from user reviews, we use ParsBERT, a language model that is used in natural language processing and capable of tasks such as sentiment analysis, named entity recognition, and more.

1- Performing Calculations on GPU:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f'device: {device}')

train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

First, we check the existence of a GPU and, if available, prepare it for calculations and consider it as the device in use.

2- Importing the Model HooshvareLab/bert-fa-base-uncased-sentiment-snappfood:

```
# General config
TRAIN_BATCH_SIZE = 16
VALID_BATCH_SIZE = 16
TEST_BATCH_SIZE = 16

CLIP = 0.0

MODEL_NAME_OR_PATH = 'HooshvareLab/bert-fa-base-uncased-sentiment-snappfood'
OUTPUT_PATH = '/content/bert-fa-base-uncased-sentiment-snappfood/pytorch_model.bin'

os.makedirs(os.path.dirname(OUTPUT_PATH), exist_ok=True)

tokenizer = BertTokenizer.from_pretrained(MODEL_NAME_OR_PATH)
```

Initially, we set the required hyperparameters and then import the ParsBERT pre-trained model as the desired model into the project environment. Then, we create the tokenizer or separator using BertToken.

```
config = BertConfig.from_pretrained(
    MODEL_NAME_OR_PATH, **{
        'label2id': label_to_id,
        'id2label': id_to_label,
    })

print(config.to_json_string())
```

Next, we set the settings related to the BERT model as shown in the above image. As you can see, we specify the model name and the mapping method between label_to_id and id_to_label.

```
idx = np.random.randint(0, len(train_df))
sample_comment = train_df.iloc[idx]['comment']
sample_label = train_df.iloc[idx]['label_id']

print(f'Sample: \n{sample_comment}\n{sample_label}')

tokens = tokenizer.tokenize(sample_comment)
token_ids = tokenizer.convert_tokens_to_ids(tokens)

print(f'  Comment: {sample_comment}')
print(f'   Tokens: {tokenizer.convert_tokens_to_string(tokens)}')
print(f'Token IDs: {token_ids}')
```

In the above code snippet, we consider a random line from the dataset and test the separation operation on it. Then, we print the review along with the Token and Token Ids in the output.

3- Encoding Operation:

```
encoding = tokenizer.encode_plus(
    sample_comment,
    max_length=COMMENT_MAX_LEN,
    truncation=True,
    add_special_tokens=True, # Add '[CLS]' and '[SEP]'
    return_token_type_ids=True,
    return_attention_mask=True,
    padding='max_length',
    return_tensors='pt',  # Return PyTorch tensors
)

print(f'Keys: {encoding.keys()}\n')
for k in encoding.keys():
    print(f'{k}:\n{encoding[k]}')
```

The encoding operation is performed as shown in the above image. As discussed in the preprocessing section, we consider the maximum length of the review in the truncation operation. We add special tokens such as sentence starting and separation tokens for words, padding, and then consider an array as an attention

mask to identify the regular tokens related to the main words in the review. Then, we return all the outputs in the form of a tensor in the output of this step.

In the output, arrays related to input_ids, token_type_ids, and attention_mask are printed. You can see the results of this step in section 3 or the Results Analysis section.

3- Implementation of Dataloader:

```
#Dataloader

class SnappFoodDataset(Dataset):
    def __init__(self, tokenizer, dataframe):
        self.dataframe = dataframe
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        item = self.dataframe.iloc[idx]
        comment = item['comment']
        label = item['label_id']

        encoding = self.tokenizer.encode_plus(
            comment,
            add_special_tokens=True,
            truncation=True,
            max_length=COMMENT_MAX_LEN,
            return_token_type_ids=True,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt')

        return {
                'comment': comment,
                'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'token_type_ids': encoding['token_type_ids'].flatten(),
                'lable': int(label),
                }


train_dataset = SnappFoodDataset(tokenizer, train_df)
valid_dataset = SnappFoodDataset(tokenizer, valid_df)
test_dataset = SnappFoodDataset(tokenizer, test_df)
```

In the above code snippet, each row of the dataset is read and then processed using the Encode code snippet with the ParsBERT tokenizer. The following items are saved in the training, validation, and evaluation datasets for each row of the dataset:

- Review
- Input_ids
- Attention_mask
- Token_type_ids
- Label

```
train_dataloder = DataLoader(train_dataset, batch_size=TRAIN_BATCH_SIZE)
valid_dataloder = DataLoader(valid_dataset, batch_size=VALID_BATCH_SIZE)
test_dataloder = DataLoader(test_dataset, batch_size=TEST_BATCH_SIZE)
```

Then, using the DataLoader class and the settings related to the batch size, we create the training, validation, and evaluation datasets.

4- Trained ParsBERT model:

```
# Pretrained Bert features
bert = BertModel.from_pretrained('HooshvareLab/bert-fa-base-uncased-sentiment-snappfood')

bert = bert.eval().to(device)

bert_outputs_train = torch.zeros(len(train_df), 768)
bert_outputs_valid = torch.zeros(len(valid_df), 768)
bert_outputs_test = torch.zeros(len(test_df), 768)

# Training feature extraction
i = 0
for batch in tqdm(train_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = bert(input_ids, attention_mask, token_type_ids)['pooler_output'].cpu()
    bert_outputs_train[i: i + out.size(0)] = out
    i += out.size(0)

# Validation feature extraction
i = 0
for batch in tqdm(valid_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = bert(input_ids, attention_mask, token_type_ids)['pooler_output'].cpu()
    bert_outputs_valid[i: i + out.size(0)] = out
    i += out.size(0)

# Test feature extraction
i = 0
for batch in tqdm(test_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = bert(input_ids, attention_mask, token_type_ids)['pooler_output'].cpu()
    bert_outputs_test[i: i + out.size(0)] = out
    i += out.size(0)
```

After entering each of the datasets into the trained ParsBERT model, a feature vector with dimensions of 1x768 is output for each sentence or review in the pooler section. In this section, we create three matrices with dimensions of the number of reviews in the dataset by 768, and in each of them, we store the output of the model based on the entered reviews.

5- BERT model output visualization:

```
from sklearn.decomposition import PCA

bert_outputs_train = bert_outputs_train.numpy()
bert_outputs_valid = bert_outputs_valid.numpy()
bert_outputs_test = bert_outputs_test.numpy()

dimentin_reduction = PCA(n_components=2)
bert_outputs_train_2d = dimentin_reduction.fit_transform(bert_outputs_train)
bert_outputs_valid_2d = dimentin_reduction.transform(bert_outputs_valid)
bert_outputs_test_2d = dimentin_reduction.transform(bert_outputs_test)

y_train = train_df['label_id'].astype(int)
y_valid = valid_df['label_id'].astype(int)
y_test = test_df['label_id'].astype(int)


plt.plot(bert_outputs_train_2d[y_train == 0, 0], bert_outputs_train_2d[y_train == 0, 1], 'b.', alpha=0.1, label='HAPPY')
plt.plot(bert_outputs_train_2d[y_train == 1, 0], bert_outputs_train_2d[y_train == 1, 1], 'r.', alpha=0.1, label='SAD')
plt.grid()
plt.legend()
plt.title('Train data')


plt.plot(bert_outputs_valid_2d[y_valid == 0, 0], bert_outputs_valid_2d[y_valid == 0, 1], 'b.', alpha=0.1, label='HAPPY')
plt.plot(bert_outputs_valid_2d[y_valid == 1, 0], bert_outputs_valid_2d[y_valid == 1, 1], 'r.', alpha=0.1, label='SAD')
plt.grid()
plt.legend()
plt.title('Validation data')


plt.plot(bert_outputs_test_2d[y_test == 0, 0], bert_outputs_test_2d[y_test == 0, 1], 'b.', alpha=0.1, label='HAPPY')
plt.plot(bert_outputs_test_2d[y_test == 1, 0], bert_outputs_test_2d[y_test == 1, 1], 'r.', alpha=0.1, label='SAD')
plt.grid()
plt.legend()
plt.title('Test data')
```

After storing the outputs of the trained ParsBERT model, we need to use dimension reduction methods since the model outputs are in 768 dimensions. For this purpose, we use the PCA library and reduce their dimensions to 2. After dimension reduction, we plot them on a graph.

6- Classification of BERT model outputs:

```python
# Train classifier on Bert outputs

from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(bert_outputs_train, y_train)

h_train = clf.predict_proba(bert_outputs_train)
h_valid = clf.predict_proba(bert_outputs_valid)
h_test = clf.predict_proba(bert_outputs_test)

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

print('Accuracy:')

print('Train: ', accuracy_score(y_train, h_train[:, 1] > 0.5))
print('Valid: ', accuracy_score(y_valid, h_valid[:, 1] > 0.5))
print('Test: ', accuracy_score(y_test, h_test[:, 1] > 0.5))

print('F1-score:')

print('Train: ', f1_score(y_train, h_train[:, 1] > 0.5))
print('Valid: ', f1_score(y_valid, h_valid[:, 1] > 0.5))
print('Test: ', f1_score(y_test, h_test[:, 1] > 0.5))

print('ROC AUC:')

print('Train: ', roc_auc_score(y_train, h_train[:, 1]))
print('Valid: ', roc_auc_score(y_valid, h_valid[:, 1]))
print('Test: ', roc_auc_score(y_test, h_test[:, 1]))
```

After entering the training, validation, and evaluation datasets and storing their outputs, we use logistic regression to classify them. We fit a logistic regression model on the model outputs using the Sklearn library. Then, we compare the predicted labels with the actual labels and evaluate the model with them.

7- Building a new model for training:

```python
# Build model

from transformers.pipelines import zero_shot_classification

class SnappFoodModel(nn.Module):
    def __init__(self, config):
        super().__init__()

        self.bert = BertModel.from_pretrained(MODEL_NAME_OR_PATH)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, 1)

    def forward(self, input_ids, attention_mask, token_type_ids):
        z = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids)

        pooled_output = self.dropout(z['pooler_output'])
        out = self.classifier(pooled_output)
        return out

sf_model = SnappFoodModel(config=config)
sf_model = sf_model.to(device)
```

In this section, a new model is implemented to fine-tune the previous model. As seen in the SnappFoodModel class in the above code snippet, the ParsBERT trained model is used in the deeper layers, and then a Dropout layer is used to prevent overfitting, and finally, a neuron is used in the last layer to classify the input.

After entering the input into the network and generating the feature vector using the BERT model, it enters the classifier after passing through the Dropout layer to determine the input class.

8- Model training:

```python
# Train model

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

def train_model(model, dataloaders, criterion, optimizer, epochs, valid_steps=1000):
    train_history = {'loss': [], 'accuracy': [], 'f1-score': [], 'roc-auc': []}
    valid_history = {'loss': [], 'accuracy': [], 'f1-score': [], 'roc-auc': []}

    for e in range(epochs):
        sum_loss_train = 0
        sum_acc_train = 0
        sum_f1_train = 0
        sum_roc_train = 0
        train_steps = 0
        model.train()
        for batch in tqdm(dataloaders['train']):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            token_type_ids = batch['token_type_ids'].to(device)
            label = batch['lable'].to(device).view(-1, 1).float()

            optimizer.zero_grad()
            output = model(input_ids, attention_mask, token_type_ids)
            loss = criterion(output, label)
            loss.backward()
            optimizer.step()

            sum_loss_train += loss.item()
            output = nn.functional.sigmoid(output.detach())
            metrics = score_model(label.cpu().numpy(), output.cpu().numpy())
            sum_acc_train += metrics['acc']
            sum_f1_train += metrics['f1-score']
            sum_roc_train += metrics['roc-auc']
            train_steps += 1

            if train_steps % valid_steps == 0:
                print('Validation...')
                sum_loss_valid = 0
                sum_acc_valid = 0
                sum_f1_valid = 0
                sum_roc_valid = 0
                validation_steps = 0
                model.eval()
                for batch in tqdm(dataloaders['valid']):
                    input_ids = batch['input_ids'].to(device)
                    attention_mask = batch['attention_mask'].to(device)
                    token_type_ids = batch['token_type_ids'].to(device)
                    label = batch['lable'].to(device).view(-1, 1).float()

                    with torch.no_grad():
                        output = model(input_ids, attention_mask, token_type_ids)
                        loss = criterion(output, label)

                        sum_loss_valid += loss.item()
                        output = nn.functional.sigmoid(output)
                        metrics = score_model(label.cpu().numpy(), output.cpu().numpy())
                        sum_acc_valid += metrics['acc']
                        sum_f1_valid += metrics['f1-score']
                        sum_roc_valid += metrics['roc-auc']
                        validation_steps += 1

                train_history['loss'].append(sum_loss_train / train_steps)
                train_history['accuracy'].append(sum_acc_train / train_steps)
                train_history['f1-score'].append(sum_f1_train / train_steps)
                train_history['roc-auc'].append(sum_roc_train / train_steps)

                valid_history['loss'].append(sum_loss_valid / validation_steps)
                valid_history['accuracy'].append(sum_acc_valid / validation_steps)
                valid_history['f1-score'].append(sum_f1_valid / validation_steps)
                valid_history['roc-auc'].append(sum_roc_valid / validation_steps)

                sum_loss_train = 0
                sum_acc_train = 0
                sum_f1_train = 0
                sum_roc_train = 0
                train_steps = 0

    return model, train_history, valid_history
```

In this section, the code related to the model training is implemented:

• We create lists for evaluating the model during training.

• For each batch, we enter the data into the model.

• At the end of each iteration in the training process, we receive the model predictions and calculate the model error.

• We enter the model error into the loss function.

• The derivatives are calculated, and the model parameters are updated according to the new weights.

• At the end of each iteration in the training process, we update the list related to the model evaluation metrics.

9- Model evaluation:

```
# Train model

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

def score_model(y_true, y_pred):
    # y_pred after sigmoid
    return {'acc': accuracy_score(y_true, y_pred > 0),
            'f1-score': f1_score(y_true, y_pred > 0),
            'roc-auc': roc_auc_score(y_true, y_pred)
    }
```

This code snippet is implemented to evaluate the trained model. It compares the model predictions with the actual labels of each observation and then evaluates the model using accuracy, F1-score, and ROC-AUC metrics.

10- Classification section training:

```
# Part1, training classifier

dataloders = {
    'train': train_dataloder,
    'valid': valid_dataloder
}

criterion = nn.BCEWithLogitsLoss()
sf_model.bert.requires_grad_ = False
optimizer = torch.optim.Adam(sf_model.classifier.parameters() ,lr=0.001)

sf_model, history_train, history_valid = train_model(sf_model, dataloders, criterion, optimizer, epochs=5)


#Plotting
plt.plot(history_train['loss'], 'bo-', label='Train')
plt.plot(history_valid['loss'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('Loss')

plt.plot(history_train['accuracy'], 'bo-', label='Train')
plt.plot(history_valid['accuracy'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('Accuracy')

plt.plot(history_train['f1-score'], 'bo-', label='Train')
plt.plot(history_valid['f1-score'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('F1-score')

plt.plot(history_train['roc-auc'], 'bo-', label='Train')
plt.plot(history_valid['roc-auc'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('ROC-AUC')
```

In this section, the goal of the implemented code snippet is to train the classification section of the model. We set sf_model.bert.require_grad_ to False at the beginning of training so that the derivatives related to the BERT model section are not calculated, and only the classification section of the model is trained. Then, we perform the training process. After training, we plot the graphs related to the evaluation metrics.

11. Model Fine-tuning:

```
# Part2, finetuning

dataloders = {
    'train': train_dataloder,
    'valid': valid_dataloder
}

criterion = nn.BCEWithLogitsLoss()
sf_model.bert.requires_grad_ = True
optimizer = torch.optim.Adam(sf_model.parameters() ,lr=0.00001)

sf_model, history_train, history_valid = train_model(sf_model, dataloders, criterion, optimizer, epochs=5)

#Plotting

plt.plot(history_train['loss'], 'bo-', label='Train')
plt.plot(history_valid['loss'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('Loss')

plt.plot(history_train['accuracy'], 'bo-', label='Train')
plt.plot(history_valid['accuracy'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('Accuracy')

plt.plot(history_train['f1-score'], 'bo-', label='Train')
plt.plot(history_valid['f1-score'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('F1-score')

plt.plot(history_train['roc-auc'], 'bo-', label='Train')
plt.plot(history_valid['roc-auc'], 'ro-', label='Valid')
plt.grid()
plt.legend()
plt.title('ROC-AUC')
```

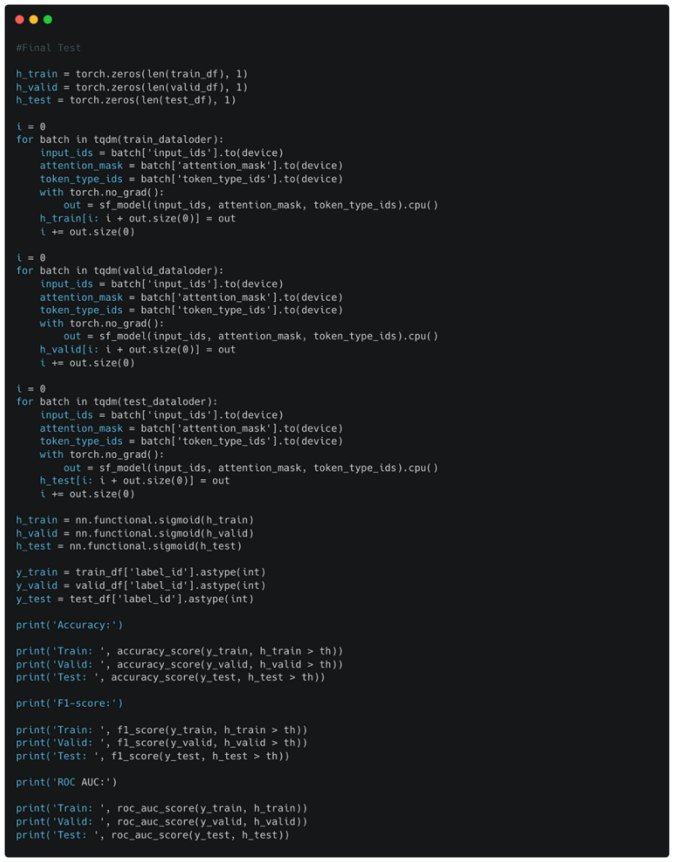
After the classification training phase, we set the value of `sf_model.bert.require_grad_` to `True` and repeat the training process to fine-tune the pre-trained ParsBERT model on our own data. In the fine-tuning process, we set the learning rate to `0.00001` because we need to fine-tune the pre-trained model with a very low learning rate to avoid diverging from the minimum point.

3. Model Evaluation:

3.1. To evaluate the model, we use evaluation metrics such as F1-score, accuracy, and AUC-ROC. We chose these metrics because they are commonly used to evaluate the performance of classification models. We chose AUC-ROC as our

primary evaluation metric because it measures the ability of the model to distinguish between positive and negative classes across all possible threshold values. When using a classification model, the model assigns a probability score to each input, indicating the probability of belonging to the positive class, and then uses a decision threshold to convert these probabilities into class labels (for example, if the probability is greater than 0.5, assign the input to the positive class). AUC-ROC can help us find the best threshold that maximizes the balance between true positive rate (TPR) and false positive rate (FPR) by evaluating the overall performance of the model across all possible thresholds. Therefore, using this metric can help us find the best threshold for separating the two classes "Happy" and "SAD" and improve the accuracy of the model.

3.2. We plot the learning curve of the model on the training, validation, and test data based on the model accuracy and analyze the results.



```
#Final Test

h_train = torch.zeros(len(train_df), 1)
h_valid = torch.zeros(len(valid_df), 1)
h_test = torch.zeros(len(test_df), 1)

i = 0
for batch in tqdm(train_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = sf_model(input_ids, attention_mask, token_type_ids).cpu()
    h_train[i: i + out.size(0)] = out
    i += out.size(0)

i = 0
for batch in tqdm(valid_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = sf_model(input_ids, attention_mask, token_type_ids).cpu()
    h_valid[i: i + out.size(0)] = out
    i += out.size(0)

i = 0
for batch in tqdm(test_dataloder):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        out = sf_model(input_ids, attention_mask, token_type_ids).cpu()
    h_test[i: i + out.size(0)] = out
    i += out.size(0)

h_train = nn.functional.sigmoid(h_train)
h_valid = nn.functional.sigmoid(h_valid)
h_test = nn.functional.sigmoid(h_test)

y_train = train_df['label_id'].astype(int)
y_valid = valid_df['label_id'].astype(int)
y_test = test_df['label_id'].astype(int)

print('Accuracy:')

print('Train: ', accuracy_score(y_train, h_train > th))
print('Valid: ', accuracy_score(y_valid, h_valid > th))
print('Test: ', accuracy_score(y_test, h_test > th))

print('F1-score:')

print('Train: ', f1_score(y_train, h_train > th))
print('Valid: ', f1_score(y_valid, h_valid > th))
print('Test: ', f1_score(y_test, h_test > th))

print('ROC AUC:')

print('Train: ', roc_auc_score(y_train, h_train))
print('Valid: ', roc_auc_score(y_valid, h_valid))
print('Test: ', roc_auc_score(y_test, h_test))
```

Model Evaluation:

In the final section, we create matrices to store the output of the model and then store the output of the model in the corresponding matrices by entering the training, validation, and test dataset for each category. We then calculate the evaluation metrics of the model based on the model predictions and the actual labels of the data and print them in the output.

Extra Credit: If your F1-score reaches or exceeds 91, you will receive extra credit for this exercise. To increase the accuracy of the model during implementation, we used various techniques such as data augmentation, using the Hazm library for preprocessing Persian texts, using the AUC-ROC evaluation metric to find the best threshold for classification, and examining the errors of the model. After examining the errors of the model, we found that some of the data in the evaluation dataset were mislabeled, and the model correctly identified the correct labels with confidence. We corrected these errors.

1. Finding the Best Classification Threshold:

```
#Finding the best threshold

thresholds = np.arange(0, 1.001, 0.01)
f1s_train = []
f1s_valid = []
f1s_test = []
for th in tqdm(thresholds):
    f1s_train.append(f1_score(y_train, h_train[:, 1] > th))
    f1s_valid.append(f1_score(y_valid, h_valid[:, 1] > th))
    f1s_test.append(f1_score(y_test, h_test[:, 1] > th))

f1s_train = np.array(f1s_train)
f1s_valid = np.array(f1s_valid)
f1s_test = np.array(f1s_test)

plt.plot(thresholds, f1s_train, label='Train')
plt.plot(thresholds, f1s_valid, label='Valid')
plt.plot(thresholds, f1s_test, label='Test')

th = thresholds[f1s_valid.argmax()]
print(th)


print('Accuracy:')

print('Train: ', accuracy_score(y_train, h_train[:, 1] > th))
print('Valid: ', accuracy_score(y_valid, h_valid[:, 1] > th))
print('Test: ', accuracy_score(y_test, h_test[:, 1] > th))

print('F1-score:')

print('Train: ', f1_score(y_train, h_train[:, 1] > th))
print('Valid: ', f1_score(y_valid, h_valid[:, 1] > th))
print('Test: ', f1_score(y_test, h_test[:, 1] > th))
```

The above code snippet is implemented to find the best threshold for classification. We evaluate the model at different threshold values and store the corresponding F1-scores. We then save the threshold that leads to the best result and use it.

2. Examining Model Errors:

```
# Error analysis

df_error_analysis = test_df.copy()
df_error_analysis['SAD prediction'] = h_test[:, 1]

df_error_analysis_SAD = df_error_analysis[df_error_analysis['label_id'] == '1']
df_error_analysis_SAD = df_error_analysis_SAD.sort_values('SAD prediction')

df_error_analysis_HAPPY = df_error_analysis[df_error_analysis['label_id'] == '0']
df_error_analysis_HAPPY = df_error_analysis_HAPPY.sort_values('SAD prediction', ascending=False)

for i in range(50):
    row = df_error_analysis_SAD.iloc[i]
    print(row['comment'])
    print(id_to_label[int(row['label_id'])])
    print(f'Model prediction probablity of beeing SAD: {100 * row["SAD prediction"]:f} %')
    print('\n----------------------------------------------------\n')


for i in range(50):
    row = df_error_analysis_HAPPY.iloc[i]
    print(row['comment'])
    print(id_to_label[int(row['label_id'])])
    print(f'Model prediction probablity of beeing SAD: {100 * row["SAD prediction"]:f} %')
    print('\n----------------------------------------------------\n')
```

By using the above code snippet, we can observe the errors of the model. We display the comments that the model misclassified with confidence opposite to the corresponding label to examine in which types of comments the model makes mistakes. After examining this section, we found that there are many comments in the dataset that are mislabeled. You can see the results of this model in section 3 or results analysis.

Conclusion and Results Analysis

In this report, we first reviewed the concepts of BERT and GPT-2 networks, and then discussed the implementation functions. Now, we move on to the conclusion and analysis of the results, where we present the results of the implemented models and analyze them. In this section, we will summarize the questions asked and examine the results of each section.

1.1 - What are the required preprocessing steps based on the presented task? Name them and prepare the data for input into the model by performing each one.

In the implementation section, we discussed the preprocessing steps in detail, but in summary, we can have a look at them again in the following list:

•Each opinion in each row of the dataset is mistakenly saved in two cells of the CSV file. We fixed this issue when reading the dataset.

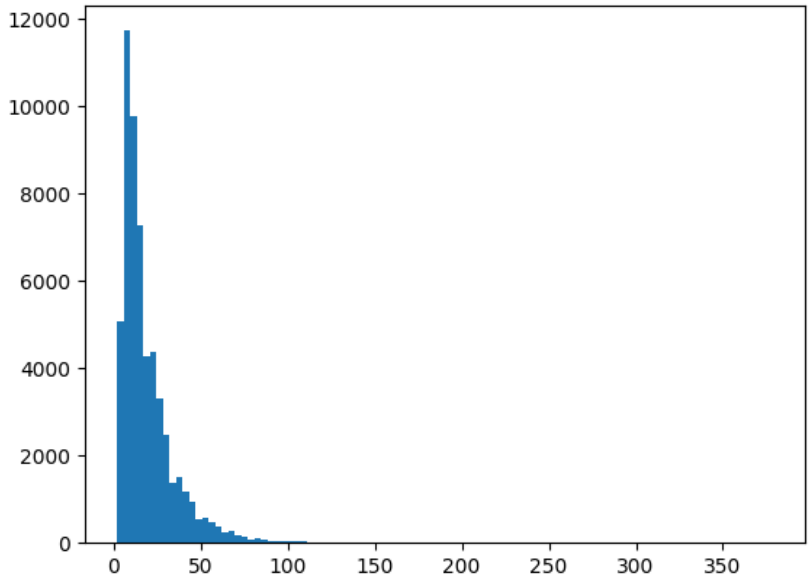
• We use the Data Augmentation method to increase the accuracy of the model.

•Check the length of customer opinions and find the truncation limit for truncating sentences. As mentioned in the descriptive questions section, Padding and Truncation help the model manage inputs of different lengths to record the best results in the evaluation section.

•Preprocessing related to Persian texts to manage spaces, half spaces, Persian numbers, left-to-right and right-to-left corrections, etc.

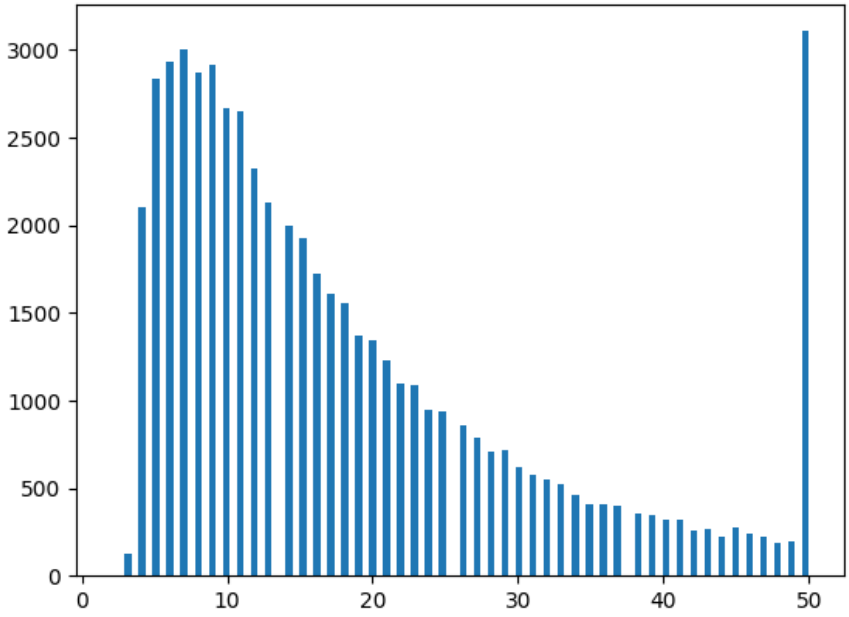
•Removing characters related to Emojy, Symbol, Flag, and #

Distribution chart of opinions with different lengths:



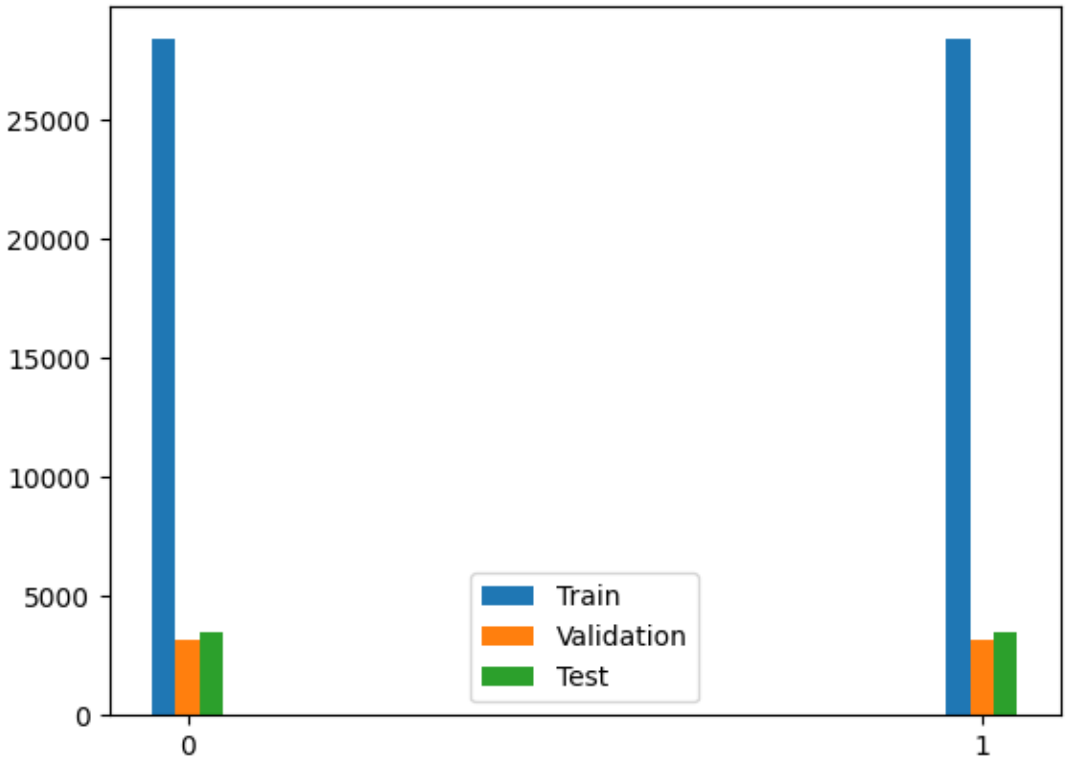
In this stage, after performing the necessary checks, we came to the conclusion that we perform Truncation with a truncation limit of 50 tokens.

Distribution chart of opinions with different lengths after performing Truncation:



1.2 - Check the balance of the dataset and draw the distribution chart of samples for each class.

In the following image, you can see the distribution chart of samples for each class in the training, validation, and evaluation datasets:



As you can see in the above image, the samples are distributed well and balanced in each dataset.

A few opinions along with their labels after performing the required preprocessing:

0	...خواهشا خواهشا خواهشا واسه ارسال شیرینیها یه فک	0
1	غذا گرم رسید کیفیت و پخت گوشت عالی بود	0
2	...قیمت سس شکلات روی بسته بندی ۱۵۰۰۰ بود ولی قیمت	1
3	...عکس توی پیج یه شیرینی شکری روشن رنگ هست که همم	1
4	باز هم میگم، پیتزا نباید اینقد چرب باشه	1
...	...	...
6995	...سطح پیتزا سوخته بود متاسفانه و ارزش اون پولو ن	1
6996	بسته بندی خیلی بسیار بد شده	1
6997	...سلام خسته نباشید همه چی خوب بود فقط یک مقدار م	0
6998	غذا سرد و کیفیت قابل قبول نبود	1
6999	یه کم خشک بود ولی در مجموع خوشمزه بود	0

7000 rows × 2 columns

Output of word separation and encoding section:

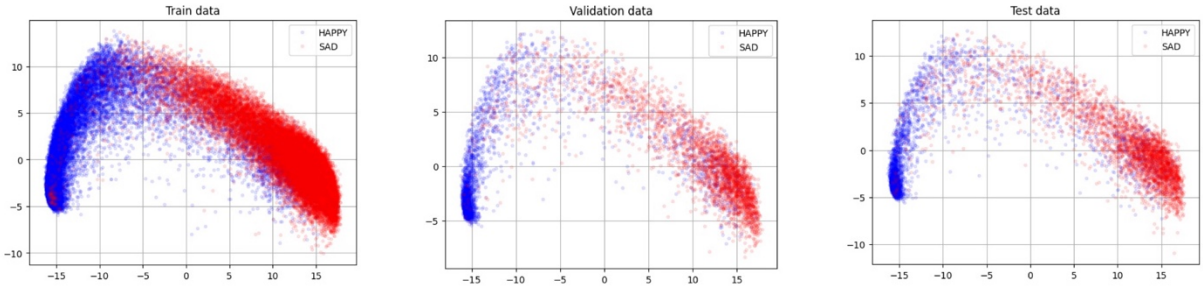
```
{'comment': 'قیمت سس شکلات روی بسته بندی ۱۵۰۰۰ بود ولی قیمت اعلامی در اسنپ ۱۸۰۰۰ بود. ۳۰۰۰ تومن تفاوت قیمت روی یک محصول خیلی زیاد است!',
 'input_ids': tensor([    2,  3377, 11186, 11255,  3040,  4341,  5936, 34020,  2834,  3362,
         3377, 12003,  2786, 10733, 48717,  2834,  1012, 12386, 42140,  5000,
         3377,  3040,  2829,  3573,  3805,  4216,  2806,  1001,     4,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0]),
 'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0]),
 'token_type_ids': tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0]),
 'lable': 1}
```

A random opinion from the evaluation dataset has been selected, and the Tokenizing and Encoding processes have been applied to it. As you can see in the above image, alongside the customer opinion, the input_ids, attention_mask, token_type_id arrays, and its label are printed in the output.

2 - Model Building

To build a model for sentiment analysis on user reviews, we will use ParsBERT. ParsBERT is a language model used in natural language processing tasks such as sentiment analysis, named entity recognition, etc.

Output of ParsBERT model:



In the previous section, we ran the pre-trained BERT model on our dataset and evaluated it. As mentioned, after entering each review into the pre-trained BERT model, a feature vector with dimensions of 1 by 768 is returned as output. To display the feature vectors of our data, we need to reduce the dimensions of their feature vectors to two dimensions so that they can be plotted. In the implementation section, we used PCA to reduce the dimensions.

Evaluation metrics for the ParsBERT model:

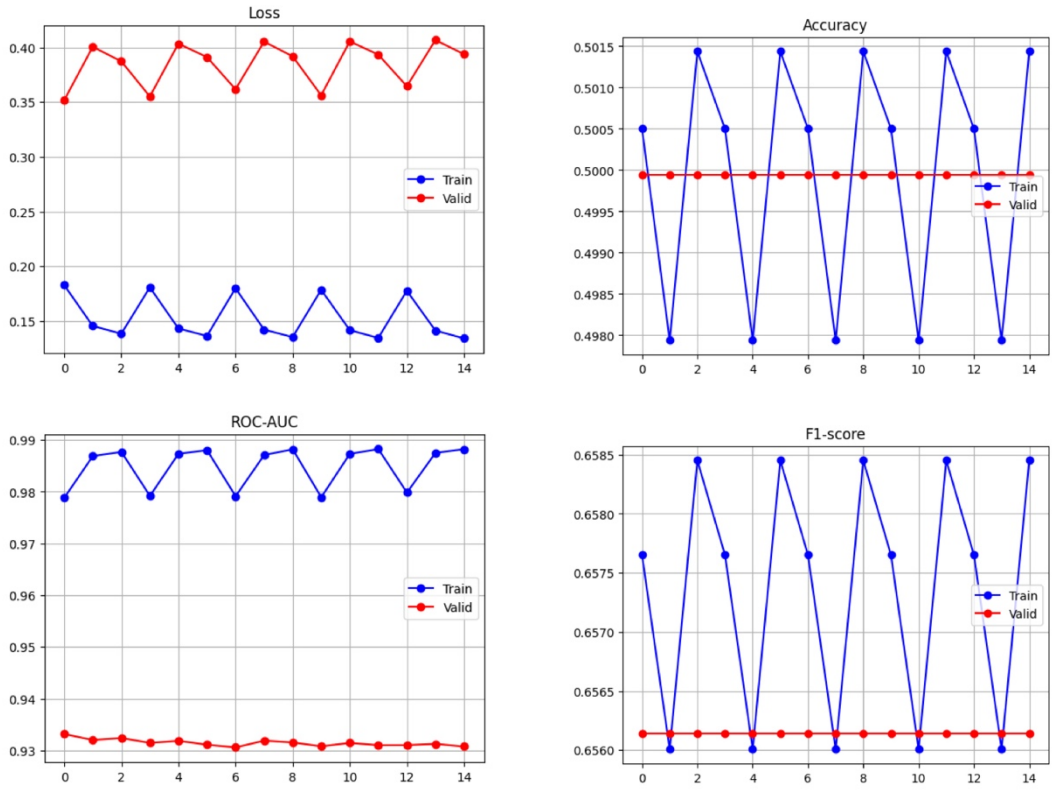
```
Accuracy:
Train:  0.9525573192239859
Valid:  0.8553968253968254
Test:  0.8677142857142857
```

```
F1-score:
Train:  0.9530590165078701
Valid:  0.8575449569976544
Test:  0.8689127972819932
```

```
ROC AUC:
Train:  0.9882107275832144
Valid:  0.9290337112622827
Test:  0.9341915102040816
```

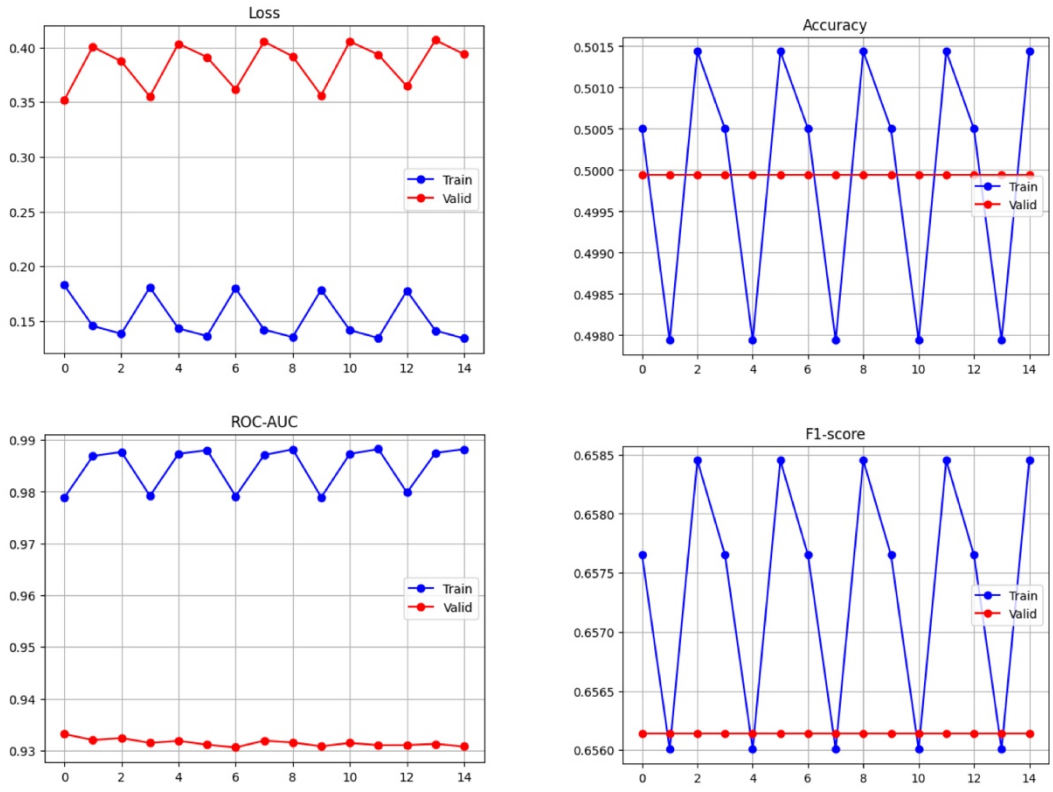
Training a new model to fine-tune ParsBERT:

As detailed in the previous section, the goal of this section is to analyze and evaluate the results of the new model. The training process for the classifier part of the new model is shown in the following graph:



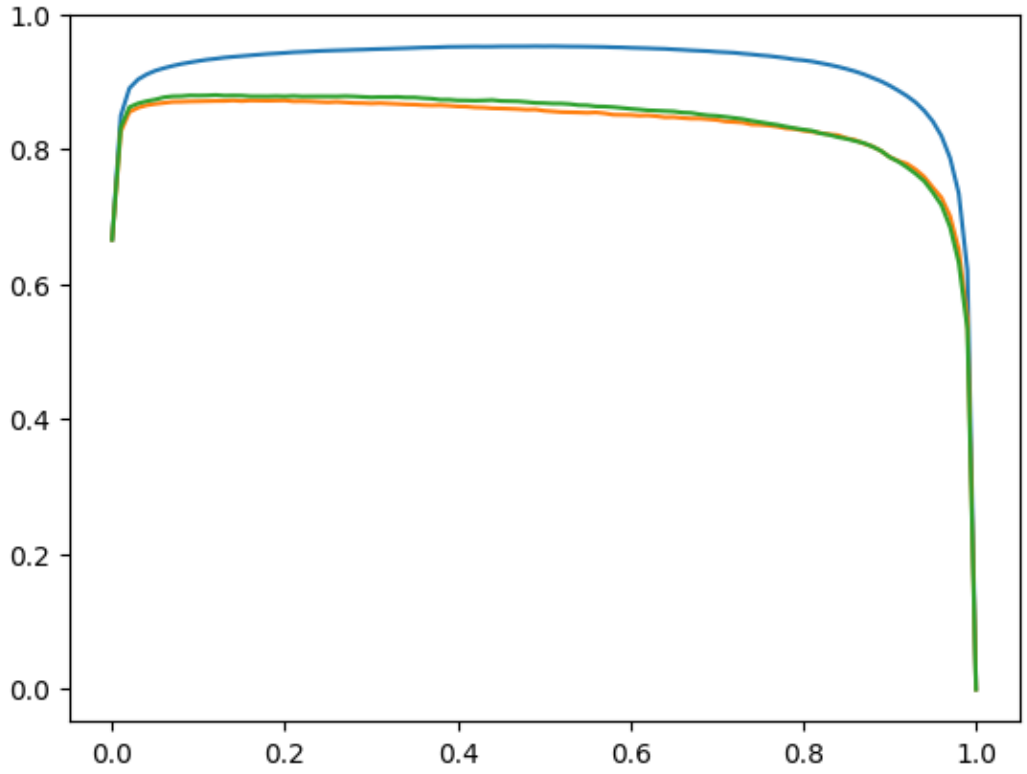
Fine-tuning the new model:

The following graph shows the Loss, Accuracy, F1-score, and ROC-AUC curves during the fine-tuning process:



Determining the best threshold for the model:

The following graph shows the F1-score for each threshold. The goal of determining the best threshold is to increase the accuracy of the model on the evaluation data. We consider the best threshold for performing the model classification operation:



Evaluating the performance of the fine-tuned model:

```
Accuracy:
Train:   0.9407936507936508
Valid:   0.8644444444444445
Test:   0.9122857142857143
```

```
F1-score:
Train:   0.9433427283927698
Valid:   0.8728409767718879
Test:   0.9200520833333332
```

```
ROC AUC:
Train:   0.9882107275832144
Valid:   0.9290337112622827
Test:   0.9745272083031098
```

To increase the accuracy of the model on the evaluation data, we evaluate the model using the training, validation, and evaluation data after using all the methods mentioned in this report and calculate the desired evaluation metrics. As seen in the above graph, after fine-tuning the model on our data, we experienced a significant increase in accuracy for each dataset.

Extra credit section: If your F1-score reaches or exceeds 91, you will receive extra credit for this exercise. As seen in the evaluation section of the fine-tuned model, the F1-score on the evaluation data was 92.

Sample predictions made by the fine-tuned model:

The following images show some sample predictions made by the fine-tuned model. It is notable that despite incorrect labels for some opinions, the model confidently predicted the correct label.

```
کاملا له شده و در هم رسید قابل تشخیص نبود
HAPPY
Model prediction probablity of beeing SAD: 99.409319 %

-----------------------------------------------------

شبیه اون چیزی که توی عکس بود، نبود
HAPPY
Model prediction probablity of beeing SAD: 99.406554 %

-----------------------------------------------------

سیب و خیار افتضاح!! بهتره قید کنید سیب و خیار درجه سه! تا طبق اون سفارش بدیم! نه به این شکل!
HAPPY
Model prediction probablity of beeing SAD: 99.404232 %

-----------------------------------------------------

کیفیت نان بسیار پایین بود
HAPPY
Model prediction probablity of beeing SAD: 99.395220 %

-----------------------------------------------------

نسبت به دفعات قبلی کامل نپخته بود و چنگال برای سرو کردن نداشت. ولی برخورد پیک خیلی عالی بود.
HAPPY
Model prediction probablity of beeing SAD: 99.375003 %
```

```
عالی بود پیشنهاد میکنم
SAD
Model prediction probablity of beeing SAD: 0.196099 %

-----------------------------------------------------

عدم ارسال کوپن پوره سیب زمینی توسط گاردن گریل
SAD
Model prediction probablity of beeing SAD: 0.235244 %

-----------------------------------------------------

درود بر شما حجم پیتزا خوب بود. یک مقدار از پیتزای شما بسیار تلخ بود! به نظرم اصلا نباید یک اشکال اینچنینی رخ بدهد حیف است.
SAD
Model prediction probablity of beeing SAD: 0.239338 %

-----------------------------------------------------

کیفیت خیبیبیبلی عاااالی بود، خوش طعم، ترد و خوشمزه خیلی هم سریع به دستم رسید و کاملا گرم بود ممنون
SAD
Model prediction probablity of beeing SAD: 0.262728 %
```