Assignment 2: Sentiment Classification Using Logistic Regression

Programming Assignment (100 Points scaled to 40)

For this assignment we will be implementing a naive bayes baseline classifier. Additionally, we will be using pytorch to implement a binary logistic regression classifier. Our task is sentiment classification for hotel reviews. The input to your model will be a text review, and the output label is a 1 or 0 marking it as positive or negative.

We have provided a util.py file for loading the data, and some of the basic modeling. Your task is to fill in the functions below in order to train as accurate a classifier as possible!

We suggest browsing the util.py script first. Additionally, make sure to install dependencies from the provided requirements.txt file in a similar fashion to the pytorch tutorial. With your environment activated int he terminal, run:

```
pip install -r requirements.txt
```

In [30]:
```python
from typing import List
import spacy
import torch
import random
```

Section 1: Sentiment Classification Dataset (Total: 20 Points)

The training data for this task consists of a collection of short hotel reviews. The data is formatted as one review per line. Each line starts with a unique identifier for the review (as in ID-2001) followed by tab and the text of the review. The reviews are not tokenized or sentence segmented in any way (the words are space separated). The positive reviews and negative reviews appear in separate files namely hotelPosT-train.txt and hotelNegT-train.txt.

In [37]:
```python
from util import load_train_data
pos_datapath = "data/hotelPosT-train.txt"
neg_datapath = "data/hotelNegT-train.txt"
all_texts, all_labels = load_train_data(pos_datapath, neg_datapath)
```

Lets look at what is in the data

In [40]:
```python
def random_sample(texts, labels, label):
    data_by_label = {}
    for lab, text in zip(labels, texts):
        if lab not in data_by_label:
            data_by_label[lab] = []
        data_by_label[lab].append(text)
    return random.choice(data_by_label[label])

print("--- Positive Example ---")
print(random_sample(all_texts, all_labels, label=1))
print("\n--- Negative Example ---")
print(random_sample(all_texts, all_labels, label=0))
```

```
--- Positive Example ---
I have no complaints and I'm a picky hotel guest. The room was clean, the sheets were not old, and the bathroom
was spotless. Everything worked properly which made for a great experience while getting ready for an intervie
w. The front desk staff was more the helpful. They gave us advice on where to eat, information on the city, and
the place I was interviewing at. The continental breakfast had lots of unique options; it went beyond the avera
ge spread at a hotel. I'm impressed.

--- Negative Example ---
I went to Syracuse last winter to watch my team play in the Carrier Dome and chose this Econo Lodge because the
prices were cheaper (because I found out later that it's in a pretty run-down part of town) and mainly cause mo
st of the better hotels were already booked. I drove up there and arrived pretty late, so I was very tired and
ready to get some sleep before the game the next day. After pulling into the small parking lot, I realized afte
r driving around that all the Free Parking spaces were gone, so I had to drive down the street to a paid parkin
g lot. Then, after carrying my luggage all the way to the hotel, I checked in and, in a sleepy haze, threw my s
tuff down in the room and went to sleep.Not more than two hours later, I woke up from incredible itching on my
arms and legs, so I quickly turned on the light and realized how awfully dirty my room was. Then I looked close
r at my bed near the sheets and I literally could see many, if not hundreds, of tiny bedbugs crawling everywher
e. I immediately ran to the desk, where no one was there of course, and after waiting for 30 minutes for someon
e to show up, I demanded another room. I then got the new room key, went up to it, and after opening the door,
I saw that this new room was no more clean than the last. And yes, there were just as many bedbugs in this bed.
I grabbed my stuff and walked back to my car and slept there for the rest of the night. Needless to say, not on
ly will I never visit this Econo Lodge again, I doubt I will ever even visit Syracuse again.
```

Test Data (WAIT TILL DEADLINE)

This is the test dataset that you will need to use to report the results on. This set is the unseen dataset meaning, you are not in anyway supoose to look what is in this dataset. We will release this dataset on the last day of the assignment's deadline.

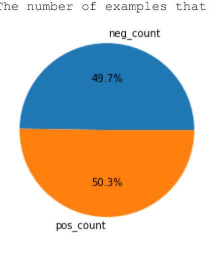
In [45]:
```python
### RUN THIS ONLY ON DEADLINE ###
# Load the test data

from typing import List, Tuple, Any


def load_test_data(filepath: str) -> tuple[list[Any], list[Any]]:
    """Load the test data, producing a List of texts, labels

    Args:
        filepath (str): Path to the training file

    Returns:
        tuple[list[Any], list[Any]]: The texts and labels
    """
    lab_map = {'POS': 1, 'NEG': 0}
    texts = []
    labels = []
    with open(filepath, "r") as file:
        for line in file:
            idx, text, label = line.rstrip().split("\t")
            texts.append(text)
            labels.append(lab_map[label])

    return texts, labels


test_texts, test_labels = load_test_data('./data/HW2-testset.txt')
```

Task 1.1: Print the number of "positive" and "negative" samples (5 Points)

It is important to know the distribution of the training examples. More often than not, you will have to work with datasets that are not "balanced" with respect to the labels of the samples. For this task, print out the number of examples that have label = 1 and label = 0, respectively, in std:out or plot a pie chart.

In [41]:
```python
### ENTER CODE HERE ###
import matplotlib.pyplot as plt
import numpy as np
# Note since we have them in two seperate files,
# this can also be done with bash commands
def label_distribution(labels):
    """
    TODO: Replace the line `raise NotImplementedError` with your code
    to print the labels distribution.
    """
    neg_count = pos_count = 0
    for num in labels:
        if num == 0:
            neg_count+=1;
        elif num == 1:
            pos_count+=1;
    print("The number of examples that have label = 0: ", neg_count)
    print("The number of examples that have label = 1: ", pos_count)

    y = np.array([neg_count, pos_count])
    mylabels = ["neg_count", "pos_count"]

    plt.pie(y, labels = mylabels, autopct='%.1f%%')
    plt.show()

label_distribution(all_labels)
```

```
The number of examples that have label = 0:  94
The number of examples that have label = 1:  95
```

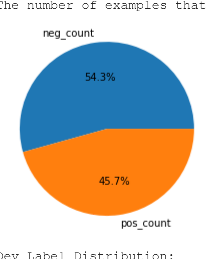


Task 1.2: Split Training and Development Sets (5 Points)

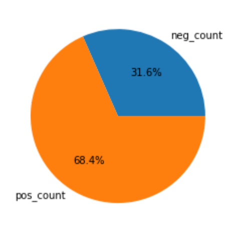
For the purpose of coming with the best parameters for the model you will have to split the dataset into training and development sets. Make sure the splits follow the same distribution.

In [47]:
```python
### ENTER CODE HERE ###
import numpy as np
import pandas as pd
def split_dataset(texts, labels):
    """
    Split the dataset randomly into 80% training and 20% development set
    Make sure the splits have the same label distribution
    """
    indices = list(range(len(texts)))
    num_training_indices = int(0.8 * len(texts))
    np.random.shuffle(indices)
    train_indices = indices[:num_training_indices]
    dev_indices = indices[num_training_indices:]
    # split the actual data
    train_texts, train_labels = texts.iloc[train_indices], labels.iloc[train_indices]
    dev_texts, dev_labels = texts.iloc[dev_indices], labels.iloc[dev_indices]
    return train_texts[0].values.tolist(), train_labels[0].values.tolist(), dev_texts[0].values.tolist(), dev_l

train_texts = pd.DataFrame(all_texts)
train_labels = pd.DataFrame(all_labels)

train_texts, train_labels, dev_texts, dev_labels = split_dataset(train_texts, train_labels)

print('Train Label Distribution:')
label_distribution(train_labels)

print('Dev Label Distribution:')
label_distribution(dev_labels)
```

```
Train Label Distribution:
The number of examples that have label = 0:  82
The number of examples that have label = 1:  69
```




```
Dev Label Distribution:
The number of examples that have label = 0:  12
The number of examples that have label = 1:  26
```



Task 1.3: Evaluation Metrics (10 Points)

Implement the evaulation metrics: Accuracy, Precision, Recall and F1 score

In [6]:
```python
### ENTER CODE HERE ###

def accuracy(predicted_labels, true_labels):
    """
    Accuracy is correct predictions / all predicitons
    """
    correct = 0
    for i in range(len(true_labels)):
        if predicted_labels[i] == true_labels[i]:
            correct += 1
    return correct / len(true_labels)

def precision(predicted_labels, true_labels):
    """
    Precision is True Positives / All Positives Predictions
    """
    TP = 0
    for i in range(len(true_labels)):
        if predicted_labels[i] == true_labels[i] and true_labels[i] == 1:
            TP += 1
    return TP / sum(predicted_labels)


def recall(predicted_labels, true_labels):
    """
    Recall is True Positives / All Positive Labels
    """
    TP = 0
    for i in range(len(true_labels)):
        if predicted_labels[i] == true_labels[i] and true_labels[i] == 1:
            TP += 1
    return TP / sum(true_labels)

def f1_score(predicted_labels, true_labels):
    """
    F1 score is the harmonic mean of precision and recall
    """
    prec = precision(predicted_labels, true_labels)
    reca = recall(predicted_labels, true_labels)
    return 2 * prec * reca / (prec + reca)
```

Section 2: Baselines (Total: 20 Points)

It is important to come up with baselines for the classifications to compare the more complicated models with. The baselines are also useful as a debugging method for your actual classfication model. You will create two baselines:

1. Random Chance
2. Naive Bayes Classifier

Task 2.1: Random Chance Classifier (5 Points)

A random chance classifier predicts the label according to the label's distribution. As an example, if the label 1 appears 70% of the times in the training set, you predict 70 out of 100 times the label 1 and label 0 30% of the times

In [7]:
```python
### ENTER CODE HERE ###

def predict_random(train_labels, num_samples):
    result = np.random.choice([0, 1], size=num_samples,
                    p=[1 - (sum(train_labels)/len(train_labels)), sum(train_labels)/len(train_labels)])
    return result
```

Task 2.2: Naive Bayes Classifier (Total: 10 Points)

In the class, Jim went over how to implement a Naive Bayes Classifier using the tokens in the training samples. In this task, you will do the same. As a preprocessing step, you might want to remove the stop words and lemmatize/stem the words of the texts.

Spacy Model https://spacy.io

To tokenize the text and help extract features from text, we will use the popular spaCy model

In [8]:
```python
### DO NOT EDIT ###

# Initialize the spacy model
nlp = spacy.load('en_core_web_sm')
```

Task 2.2.1: Play around with spacy (0 Points)

In [9]:
```python
### ENTER CODE HERE ###

test_string = "This is an amazing sentence"

# parse the string with spacy model
test_doc = nlp(test_string)

print('Token', 'Lemma', 'Is_Stopword?')
for token in test_doc:
    print(token, token.lemma_, token.is_stop)
```

```
Token Lemma Is_Stopword?
This this True
is be True
an an True
amazing amazing False
sentence sentence False
```

Task 2.2.2: Preprocessing (5 Points)

Remove stopwords and lemmatize the words of a text

In [10]:
```python
### ENTER CODE HERE ###

def pre_process(text: str) -> List[str]:
    """
    remove stopwords and lemmatize and return an array of lemmas
    """
    my_doc = nlp(text)

    lemma_nonstop = []

    for token in my_doc:
        if token.is_stop == False:
            lemma_nonstop.append(token.lemma_)
    return lemma_nonstop

test_string = "This sentence needs to be lemmatized"

assert len({'sentence', 'need', 'lemmatize', 'lemmatiz'}.intersection(pre_process(test_string))) >= 3

print('All Test Cases Passed!')
```

```
All Test Cases Passed!
```

Task 2.2.3: The Naive Bayes Class (5 Points)

The standard way of implementing classifiers like Naive Bayes is to implement the two methods: "fit" and "predict". The fit method expects the training data along with labels, and the predict method predicts the labels for the provides texts of samples.

In [24]:
```python
### ENTER CODE HERE ###
import operator
import math
class NaiveBayesClassifier:
    def __init__(self, num_classes):
        self.num_classes = num_classes
        self.label_words_count_pos = dict()
        self.label_words_count_neg = dict()

    def fit(self, texts, labels):
        """
        1. Group samples by their labels
        2. Preprocess each text
        3. Count the words of the text for each label
        """

        for i in range(len(texts)):

            vector = texts[i]
            class_value = int(labels[i])

            words = pre_process(vector)

            if (class_value == 1):
                for word in words:
                    self.label_words_count_pos[word] = self.label_words_count_pos.get(word, 0) + 1
            elif (class_value == 0):
                for word in words:
                    self.label_words_count_neg[word] = self.label_words_count_neg.get(word, 0) + 1

    def predict(self, texts):
        """
        1. Preprocess the texts
        2. Predict the class by using the likelihood with Bayes Method and Laplace Smoothing
        """

        pos_count_sum = sum(self.label_words_count_pos.values())
        neg_count_sum = sum(self.label_words_count_neg.values())

        label_words_prob_pos = self.label_words_count_pos.copy()
        label_words_prob_neg = self.label_words_count_neg.copy()

        label_words_prob_pos = {key: value / pos_count_sum for key, value in label_words_prob_pos.items()}
        label_words_prob_neg = {key: value / neg_count_sum for key, value in label_words_prob_neg.items()}

        result = []

        for i in range(len(texts)):

            vector = texts[i]
            words = pre_process(vector)

            pos_score = 0
            neg_score = 0

            for word in words:

                if word in self.label_words_count_pos:
                    pos_score += math.log(label_words_prob_pos[word])
                elif word not in self.label_words_count_pos:
                    pos_score += math.log(float(1/pos_count_sum))

                if word in self.label_words_count_neg:
                    neg_score += math.log(label_words_prob_neg[word])
                elif word not in self.label_words_count_neg:
                    neg_score += math.log(float(1/neg_count_sum))

            if pos_score > neg_score:
                result.append(1)
            else:
                result.append(0)

        return result
```

Task 2.3: Baseline Results (5 Points)

Since there is not hyperparameter-tuing required for the baselines, we can use the entirety of the training set (no need to split the dataset into train and development). Report the results you achieve with the two baselines by running the following cell:

In [12]:
```python
### DO NOT EDIT ###

### DEV SET RESULTS
testset_prediction_random = predict_random(train_labels, num_samples=len(dev_labels))
print('Random Chance F1:', f1_score(testset_prediction_random, dev_labels))

naive_bayes_classifier = NaiveBayesClassifier(num_classes=2)
naive_bayes_classifier.fit(train_texts, train_labels)
testset_predictions_nb = naive_bayes_classifier.predict(dev_texts)
print('Naive Bayes F1:', f1_score(testset_predictions_nb, dev_labels))
```

```
Random Chance F1: 0.5945945945945946
Naive Bayes F1: 0.9090909090909091
```

In [27]:
```python
### DO NOT EDIT ###
### RUN THIS ONLY ON DEADLINE ###
### TEST SET RESULTS

testset_prediction_random = predict_random(all_labels, num_samples=len(test_labels))
print('Random Chance F1:', f1_score(testset_prediction_random, test_labels))

naive_bayes_classifier = NaiveBayesClassifier(num_classes=2)
naive_bayes_classifier.fit(all_texts, all_labels)
testset_predictions_nb = naive_bayes_classifier.predict(test_texts)
print('Naive Bayes F1:', f1_score(testset_predictions_nb, test_labels))
```

```
Random Chance F1: 0.4912280701754386
Naive Bayes F1: 0.9019607843137256
```

Section 3: Logistic Regression on Features (Total: 60 Points)

Now let's try building a logistic regression based classifier on hand-engineered features.

The following tasks are going to be the implementation of the components required in building a Logistic Regressor.

Task 3.0: Feature Extraction (20 points)

This is perhaps the most challenging part of this assignment. In the class, we went over how to featurize text for a classification system for sentiment analysis. In this assignment, you should implement and build upon this to accuractely classify the hotel reviews.

This task requires a thorough understanding of the dataset to answer the important question, "What is in the data?". Please go through some of the datapoints and convert the signals that you think might help in identifying "sentiment" as features.

Please refer to the section in Jim's book that illustrates the process of feature engineering for this task. We have attached an image of the table below:

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Please use the files with postive and negative words attached in the assignment: positive_words.txt and negative_words.txt

In [13]:
```python
from util import load_train_data
pos_datapath = "data/positive-words.txt"
neg_datapath = "data/negative-words.txt"

my_file = open(pos_datapath, "r")
content = my_file.read()
pos_lex = content.split("\n")
my_file.close()

my_file = open(neg_datapath, "r")
content = my_file.read()
neg_lex = content.split("\n")
my_file.close()

def make_test_feature_1(text: spacy.tokens.doc.Doc):
    count = 0
    for t in text:
        tem = t.lemma_
        if t.lemma_ in pos_lex:
            count += 1
    return count

def make_test_feature_2(text: spacy.tokens.doc.Doc):
    count = 0
    for t in text:
        if t.lemma_ in neg_lex:
            count += 1
    return count

def make_test_feature_3(text: spacy.tokens.doc.Doc):
    count = 0
    for t in text:
        if t.lemma_ == 'no':
            count += 1
    return count

def make_test_feature_4(text: spacy.tokens.doc.Doc):
    count = 0
    pronouns = ['I', 'me', 'my', 'My' , 'mine', 'myself', 'You', 'you', 'your', 'Your', 'yours', 'yourself']
    for t in text:
        if t.lemma_ in pronouns:
            count += 1
    return count

def make_test_feature_5(text: spacy.tokens.doc.Doc):
    count = 0
    for t in text:
        if t.lemma_ == '!':
            count += 1
    return count

def make_test_feature_6(text: spacy.tokens.doc.Doc):
    count = 0
    for t in text:
        count += 1
    return np.log(count)


def extract_features(text: spacy.tokens.doc.Doc):
    features = []
    # TODO: Replace this with your own feature extraction functions.
    # TODO: add more features to the feature vector
    features.append(make_test_feature_1(text))
    features.append(make_test_feature_2(text))
    features.append(make_test_feature_3(text))
    features.append(make_test_feature_4(text))
    features.append(make_test_feature_5(text))
    features.append(make_test_feature_6(text))

    return features
```

In [14]:
```python
### ENTER CODE HERE ###
### DO NOT CHANGE THE SIGNATURE OF THE function THOUGH ###

def featurize_data(texts, labels):
    features = [
        extract_features(doc) for doc in nlp.pipe(texts)
    ]
    return torch.FloatTensor(features), torch.FloatTensor(labels)
```

Task 3.0.2: Feature Scaling (10 Points)

In this task we will use the data normalization technique to ensure the scales of the feature are consistent. After featurizing the dataset, we need to call the following function before passing it to the classifier

Normalization Formula

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In [15]:
```python
### ENTER CODE HERE ###

def normalize(features: torch.Tensor) -> torch.Tensor:
    """
    return the features transformed by the above formula of normalization
    """
    feature_min = torch.min(features)
    feature_max = torch.max(features)
    for row in range(features.shape[0]):
        for col in range(features.shape[1]):
            features[row][col] = (features[row][col].item() - feature_min) / (feature_max - feature_min)
    return features
```

Training a Logistic Regression Classifier (Total: 30 Points)

In this section, you will implement the components needed to train the binary classifier using logistic regression

Here we define our pytorch logistic regression classifier (DO NOT EDIT THIS)

In [16]:
```python
class SentimentClassifier(torch.nn.Module):
    def __init__(self, input_dim: int):
        super().__init__()
        # We force output to be one, since we are doing binary logistic regression
        self.output_size = 1
        self.coefficients = torch.nn.Linear(input_dim, self.output_size)
        # Initialize weights. Note that this is not strictly necessary,
        # but you should test different initializations per lecture
        initialize_weights(self.coefficients)

    def forward(self, features: torch.Tensor):
        # We predict a number by multipling by the coefficients
        # and then take the sigmoid to turn the score as logits
        return torch.sigmoid(self.coefficients(features))
```

Task 3.1: Initialize the weights. (5 Points)

Initialization of the parameters is an important step to ensure the SGD algorithm converges to a global optimum. Typically, we need to try different initialization methods and compare the accuracy we achieve for the development set. In this task, implement the function that initializes the parameters to ...

In [17]:
```python
### ENTER CODE HERE ###
import math
def initialize_weights(coefficients):
    """
    TODO: Replace the line `raise NotImplementedError` with your code.
    Initialize the weights of the coefficients by assigning the parameter
    coefficients.weights.data = ...
    """
    return torch.nn.init.xavier_uniform_(coefficients.weight)
```

Let's build a training function similar to the linear regressor from the tutorial

Task 3.2: Logistic Loss Function (10 Points)

In [18]:
```python
### ENTER CODE HERE ###

def logistic_loss(prediction: torch.Tensor, label: torch.Tensor) -> torch.Tensor:
    """
    TODO: Implement the logistic loss function between a prediction and label.
    """
    l1 = torch.nn.BCEWithLogitsLoss()
    return l1(prediction, label)
```

Task 3.3: Create an SGD optimizer (0 Points)

We have already provided the implementation of how to create the SGD optimizer

You may try different optimizers refering to the docs provided

In [19]:
```python
### ENTER CODE HERE ###

def make_optimizer(model, learning_rate) -> torch.optim:
    """
    Returns an Stocastic Gradient Descent Optimizer
    See here for algorithms you can import: https://pytorch.org/docs/stable/optim.html
    """
    return torch.optim.SGD(model.parameters(), learning_rate)
```

Task 3.5: Converting Logits into Predictions (5 Points)

In [20]:
```python
### ENTER CODE HERE ###

def predict(model, features):
    with torch.no_grad():
        """
        TODO: Replace the line `raise NotImplementedError`
        with the logic of converting the logits into prediction labels (0, 1)
        """
        logits = model(features)
        return torch.round(logits)
```

Training Function (DO NOT EDIT THIS)

In [21]:
```python
### DO NOT EDIT ###

from tqdm.autonotebook import tqdm
import random


def training_loop(
    num_epochs,
    batch_size,
    train_features,
    train_labels,
    dev_features,
    dev_labels,
    optimizer,
    model
):
    samples = list(zip(train_features, train_labels))
    random.shuffle(samples)
    batches = []
    for i in range(0, len(samples), batch_size):
        batches.append(samples[i:i+batch_size])
    print("Training...")
    for i in range(num_epochs):
        losses = []
        for batch in tqdm(batches):
            # Empty the dynamic computation graph
            features, labels = zip(*batch)
            features = torch.stack(features)
            labels = torch.stack(labels)
            optimizer.zero_grad()
            # Run the model
            logits = model(features)
            # Compute loss
            loss = logistic_loss(torch.squeeze(logits), labels)
            # In this logistic regression example,
            # this entails computing a single gradient
            loss.backward()
            # Backpropogate the loss through our model

            # Update our coefficients in the direction of the gradient.
            optimizer.step()
             # For logging
            losses.append(loss.item())

        # Estimate the f1 score for the development set
        dev_f1 = f1_score(predict(model, dev_features), dev_labels)
        print(f"epoch {i}, loss: {sum(losses)/len(losses)}")
        print(f"Dev F1 {dev_f1}")

    # Return the trained model
    return model
```

Task 3.6: Train the classifier (10 Points)

Run the following cell to train a logistic regressor on your hand-engineered features.

In [52]:
```python
num_epochs = 500

train_features, train_labels_tensor = featurize_data(train_texts, train_labels)
train_features = normalize(train_features)
dev_features, dev_labels_tensor = featurize_data(dev_texts, dev_labels)
dev_features = normalize(dev_features)
model = SentimentClassifier(train_features.shape[1])
optimizer = make_optimizer(model, learning_rate=0.5)

trained_model = training_loop(
    num_epochs,
    16,
    train_features,
    train_labels_tensor,
    dev_features,
    dev_labels_tensor,
    optimizer,
    model
)
```

```
Training...
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 0, loss: 0.7417197108268738
Dev F1 tensor([0.3226])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 1, loss: 0.7333073854446411
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 2, loss: 0.7263218641281128
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 3, loss: 0.7206681430339813
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 4, loss: 0.7161434412002563
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 5, loss: 0.7125257790088654
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 6, loss: 0.7096173286437988
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 7, loss: 0.7072575688362122
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 8, loss: 0.705322003364563
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 9, loss: 0.7037158846855164
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 10, loss: 0.7023677825927734
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 11, loss: 0.701223623752594
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 12, loss: 0.7002422451972962
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 13, loss: 0.6993921220302581
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 14, loss: 0.6986488223075866
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 15, loss: 0.697993278503418
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 16, loss: 0.6974104404449463
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 17, loss: 0.696888267993927
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 18, loss: 0.696417099237442
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 19, loss: 0.6959891319274902
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 20, loss: 0.695597892999649
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 21, loss: 0.6952381193637848
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 22, loss: 0.694905412197113
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 23, loss: 0.6945960581302643
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 24, loss: 0.6943069815635681
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 25, loss: 0.6940354943275452
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 26, loss: 0.6937793254852295
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 27, loss: 0.6935365200042725
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 28, loss: 0.6933054566383362
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 29, loss: 0.6930846154689789
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 30, loss: 0.6928727149963378
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 31, loss: 0.6926686227321625
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 32, loss: 0.6924713492393494
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 33, loss: 0.6922800660133361
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 34, loss: 0.6920938551425934
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 35, loss: 0.6919121026992798
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 36, loss: 0.691734117269516
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 37, loss: 0.6915593326091767
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 38, loss: 0.6913872182369232
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 39, loss: 0.6912172257900238
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 40, loss: 0.6910490095615387
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 41, loss: 0.6908820867538452
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 42, loss: 0.6907160639762878
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 43, loss: 0.6905506193637848
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 44, loss: 0.6903853952884674
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 45, loss: 0.6902200996875762
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 46, loss: 0.6900543987751007
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 47, loss: 0.6898880362510681
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 48, loss: 0.6897207081317902
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 49, loss: 0.6895521819591522
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 50, loss: 0.6893822431564331
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 51, loss: 0.6892105996608734
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 52, loss: 0.6890370845794678
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 53, loss: 0.688861483335495
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 54, loss: 0.688683545589447
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 55, loss: 0.6885031044483185
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 56, loss: 0.6883199572563171
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 57, loss: 0.6881339848041534
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 58, loss: 0.6879449427127838
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 59, loss: 0.687752777338028
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 60, loss: 0.6875571846961975
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 61, loss: 0.6873581647872925
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 62, loss: 0.6871555209159851
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 63, loss: 0.6869491696357727
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 64, loss: 0.6867389857769013
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 65, loss: 0.6865248620510102
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 66, loss: 0.6863066971302032
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 67, loss: 0.6860845148563385
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 68, loss: 0.6858581602573395
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 69, loss: 0.6856276154518127
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 70, loss: 0.6853928923606872
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 71, loss: 0.6851539373397827
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 72, loss: 0.6849107801914215
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 73, loss: 0.6846634387969971
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 74, loss: 0.6844119250774383
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 75, loss: 0.6841563165187836
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 76, loss: 0.6838966727256774
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 77, loss: 0.6836330473423005
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 78, loss: 0.6833655774593353
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 79, loss: 0.6830943644046783
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 80, loss: 0.6828195035457612
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 81, loss: 0.6825411736965179
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 82, loss: 0.682259476184845
Dev F1 tensor([nan])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 83, loss: 0.6819745838642121
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 84, loss: 0.6816866993904114
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 85, loss: 0.6813959598541259
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 86, loss: 0.6811024785041809
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 87, loss: 0.6808065593242645
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 88, loss: 0.6805082798004151
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 89, loss: 0.6802078723907471
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 90, loss: 0.6799054920673371
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 91, loss: 0.6796013295650483
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 92, loss: 0.6792955935001374
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 93, loss: 0.6789883971214294
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 94, loss: 0.6786799252033233
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 95, loss: 0.6783703446388245
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 96, loss: 0.6780597984790802
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 97, loss: 0.6777484655380249
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 98, loss: 0.6774364292621613
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 99, loss: 0.6771238088607788
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 100, loss: 0.6768108129501342
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 101, loss: 0.6764975190162659
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 102, loss: 0.6761840164661408
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 103, loss: 0.6758703827857971
Dev F1 tensor([0.0741])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 104, loss: 0.6755567848682403
Dev F1 tensor([0.1429])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 105, loss: 0.6752432346343994
Dev F1 tensor([0.1429])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 106, loss: 0.6749298751354218
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 107, loss: 0.6746167540550232
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 108, loss: 0.674303936958313
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 109, loss: 0.6739915132522583
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 110, loss: 0.6736795365810394
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 111, loss: 0.6733680605888367
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 112, loss: 0.673057097196579
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 113, loss: 0.6727467536926269
Dev F1 tensor([0.2069])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 114, loss: 0.672437059879303
Dev F1 tensor([0.2667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 115, loss: 0.6721280515193939
Dev F1 tensor([0.3750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 116, loss: 0.6718197584152221
Dev F1 tensor([0.3750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 117, loss: 0.6715122640132904
Dev F1 tensor([0.3750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 118, loss: 0.6712055444717407
Dev F1 tensor([0.3750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 119, loss: 0.6708996534347534
Dev F1 tensor([0.4242])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 120, loss: 0.6705946624279022
Dev F1 tensor([0.4242])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 121, loss: 0.6702905416488647
Dev F1 tensor([0.4242])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 122, loss: 0.6699873387813569
Dev F1 tensor([0.4242])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 123, loss: 0.6696851372718811
Dev F1 tensor([0.4242])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 124, loss: 0.669383829832077
Dev F1 tensor([0.4706])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 125, loss: 0.6690835475921631
Dev F1 tensor([0.4706])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 126, loss: 0.6687842726707458
Dev F1 tensor([0.4706])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 127, loss: 0.6684860527515412
Dev F1 tensor([0.4706])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 128, loss: 0.6681888282299042
Dev F1 tensor([0.4706])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 129, loss: 0.6678927361965179
Dev F1 tensor([0.5143])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 130, loss: 0.6675977051258087
Dev F1 tensor([0.5143])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 131, loss: 0.6673037707805634
Dev F1 tensor([0.5143])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 132, loss: 0.6670109570026398
Dev F1 tensor([0.5556])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 133, loss: 0.666719233995018
Dev F1 tensor([0.5556])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 134, loss: 0.6664286434650422
Dev F1 tensor([0.5556])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 135, loss: 0.6661392629146576
Dev F1 tensor([0.5556])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 136, loss: 0.665850967168808
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 137, loss: 0.6655638754367829
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 138, loss: 0.6652779519557953
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 139, loss: 0.6649932205677033
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 140, loss: 0.6647096574306488
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 141, loss: 0.6644272863864898
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 142, loss: 0.6641461193561554
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 143, loss: 0.6638661861419678
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 144, loss: 0.6635874330997467
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 145, loss: 0.6633099019527435
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 146, loss: 0.6630335986614228
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 147, loss: 0.6627584755420685
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 148, loss: 0.66248459815979
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 149, loss: 0.662211936712265
Dev F1 tensor([0.6316])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 150, loss: 0.6619405150413513
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 151, loss: 0.6616703152656556
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 152, loss: 0.661401343345642
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 153, loss: 0.6611335933208465
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 154, loss: 0.6608670771121978
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 155, loss: 0.6606017649173737
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 156, loss: 0.6603376924991607
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 157, loss: 0.6600748479366303
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 158, loss: 0.6598132073879241
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 159, loss: 0.6595528185367584
Dev F1 tensor([0.6667])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 160, loss: 0.6592936396598816
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 161, loss: 0.6590356707572937
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 162, loss: 0.6587789237499238
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 163, loss: 0.6585233867168426
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 164, loss: 0.658269077539444
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 165, loss: 0.6580159544944764
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 166, loss: 0.657764059305191
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 167, loss: 0.6575133621692657
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 168, loss: 0.6572638154029846
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 169, loss: 0.6570155203342438
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 170, loss: 0.656768411397934
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 171, loss: 0.6565224707126618
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 172, loss: 0.6562776863574982
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 173, loss: 0.656034117937088
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 174, loss: 0.6557917296886444
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 175, loss: 0.6555504918098449
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 176, loss: 0.6553104400634766
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 177, loss: 0.6550715446472168
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 178, loss: 0.6548337996006012
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 179, loss: 0.6545972168445587
Dev F1 tensor([0.7000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 180, loss: 0.6543617725372315
Dev F1 tensor([0.7317])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 181, loss: 0.6541274666786194
Dev F1 tensor([0.7317])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 182, loss: 0.6538943231105805
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 183, loss: 0.6536622881889343
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 184, loss: 0.6534313976764679
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 185, loss: 0.6532016277313233
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 186, loss: 0.652972960472107
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 187, loss: 0.6527454495429993
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 188, loss: 0.6525190055370331
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 189, loss: 0.6522936701774598
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 190, loss: 0.6520694434642792
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 191, loss: 0.6518463015556335
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 192, loss: 0.6516242623329163
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 193, loss: 0.6514032781124115
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 194, loss: 0.6511834144592286
Dev F1 tensor([0.7619])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 195, loss: 0.6509645760059357
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 196, loss: 0.6507468163967133
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 197, loss: 0.6505301296710968
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 198, loss: 0.6503144860267639
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 199, loss: 0.6500999152660369
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 200, loss: 0.6498863518238067
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 201, loss: 0.6496738612651825
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 202, loss: 0.6494623959064484
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 203, loss: 0.6492519378662109
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 204, loss: 0.6490424990653991
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 205, loss: 0.648834091424942
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 206, loss: 0.6486266791820526
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 207, loss: 0.6484202980995178
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 208, loss: 0.6482148885726928
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 209, loss: 0.6480104982852936
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 210, loss: 0.6478070795536042
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 211, loss: 0.6476046562194824
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 212, loss: 0.6474031925201416
Dev F1 tensor([0.7727])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 213, loss: 0.6472027361392975
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 214, loss: 0.6470032155513763
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 215, loss: 0.6468046605587006
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 216, loss: 0.6466070652008057
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 217, loss: 0.6464104056358337
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 218, loss: 0.6462147116661072
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 219, loss: 0.6460199356079102
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 220, loss: 0.645826107263565
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 221, loss: 0.6456332087516785
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 222, loss: 0.6454412281513214
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 223, loss: 0.6452501595020295
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 224, loss: 0.6450600266456604
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 225, loss: 0.6448707699775695
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 226, loss: 0.6446824312210083
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 227, loss: 0.6444949865341186
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 228, loss: 0.6443084418773651
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 229, loss: 0.6441227555274963
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 230, loss: 0.6439379751682281
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 231, loss: 0.6437540531158448
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 232, loss: 0.6435710191726685
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 233, loss: 0.6433888375759125
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 234, loss: 0.6432075083255768
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 235, loss: 0.6430270314216614
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 236, loss: 0.6428473949432373
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 237, loss: 0.6426686406135559
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 238, loss: 0.642490702867508
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 239, loss: 0.6423135876655579
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 240, loss: 0.6421373009681701
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 241, loss: 0.6419618725776672
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 242, loss: 0.6417872488498688
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 243, loss: 0.6416134059429168
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 244, loss: 0.6414403975009918
Dev F1 tensor([0.8000])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 245, loss: 0.6412681996822357
Dev F1 tensor([0.8261])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 246, loss: 0.6410968005657196
Dev F1 tensor([0.8261])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 247, loss: 0.6409261763095856
Dev F1 tensor([0.8261])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 248, loss: 0.640756344795227
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 249, loss: 0.6405873000621796
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 250, loss: 0.6404190480709075
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 251, loss: 0.6402515649795533
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 252, loss: 0.6400848448276519
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 253, loss: 0.6399189114570618
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 254, loss: 0.6397537171840668
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 255, loss: 0.6395892381668091
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 256, loss: 0.6394256055355072
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 257, loss: 0.6392626404762268
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 258, loss: 0.639100456237793
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 259, loss: 0.6389390051364898
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 260, loss: 0.6387782752513885
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 261, loss: 0.6386182904243469
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 262, loss: 0.6384590029716491
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 263, loss: 0.6383004724979401
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 264, loss: 0.6381425976753234
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 265, loss: 0.637985497713089
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 266, loss: 0.6378290772438049
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 267, loss: 0.6376734018325806
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 268, loss: 0.6375183403491974
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 269, loss: 0.6373640239238739
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 270, loss: 0.6372103750705719
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 271, loss: 0.6370574593544006
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 272, loss: 0.636905175447464
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 273, loss: 0.6367535769939423
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 274, loss: 0.6366026639938355
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 275, loss: 0.6364524304866791
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 276, loss: 0.6363028109073638
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 277, loss: 0.6361538827419281
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 278, loss: 0.6360056340694428
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 279, loss: 0.635858005285263
Dev F1 tensor([0.8511])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 280, loss: 0.635711032152176
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 281, loss: 0.6355646967887878
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 282, loss: 0.6354190409183502
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 283, loss: 0.6352739691734314
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 284, loss: 0.6351295709609985
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 285, loss: 0.6349857568740844
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 286, loss: 0.6348426043987274
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 287, loss: 0.6347000658512115
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 288, loss: 0.6345581471920013
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 289, loss: 0.6344168305397033
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 290, loss: 0.6342761158943176
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 291, loss: 0.6341360509395599
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 292, loss: 0.633996558189392
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 293, loss: 0.633857649564743
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 294, loss: 0.6337193429470063
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 295, loss: 0.6335816621780396
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 296, loss: 0.6334445297718048
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 297, loss: 0.6333080112934113
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 298, loss: 0.6331720411777496
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 299, loss: 0.6330366611480713
```

```
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 300, loss: 0.6329018354415894
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 301, loss: 0.6327676236629486
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 302, loss: 0.6326339602470398
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 303, loss: 0.6325008511543274
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 304, loss: 0.6323683202266693
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 305, loss: 0.6322363078594208
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 306, loss: 0.632104879617691
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 307, loss: 0.6319739997386933
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 308, loss: 0.6318436682224273
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 309, loss: 0.6317138791084289
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 310, loss: 0.6315846025943757
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 311, loss: 0.6314558863639832
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 312, loss: 0.6313277065753937
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 313, loss: 0.6312000632286072
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 314, loss: 0.6310729146003723
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 315, loss: 0.6309463024139405
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 316, loss: 0.6308202266693115
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 317, loss: 0.6306946456432343
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 318, loss: 0.630569589138031
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 319, loss: 0.6304450631141663
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 320, loss: 0.6303210139274598
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 321, loss: 0.6301974892616272
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 322, loss: 0.6300744593143464
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 323, loss: 0.629951936006546
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 324, loss: 0.6298299133777618
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 325, loss: 0.6297083914279937
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 326, loss: 0.6295873463153839
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 327, loss: 0.6294667840003967
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 328, loss: 0.6293467223644257
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 329, loss: 0.6292271316051483
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 330, loss: 0.6291080296039582
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 331, loss: 0.6289893805980682
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 332, loss: 0.6288712203502655
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 333, loss: 0.6287535429000854
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 334, loss: 0.6286363244056702
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 335, loss: 0.6285195708274841
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 336, loss: 0.6284032881259918
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 337, loss: 0.6282874345779419
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 338, loss: 0.6281720697879791
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 339, loss: 0.6280571699142456
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 340, loss: 0.62794269323349
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 341, loss: 0.6278286814689636
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 342, loss: 0.6277150630950927
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 343, loss: 0.6276019752025604
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 344, loss: 0.6274892687797546
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 345, loss: 0.6273770093917846
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 346, loss: 0.627265202999115
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 347, loss: 0.6271538436412811
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 348, loss: 0.6270429074764252
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 349, loss: 0.6269323647022247
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 350, loss: 0.6268222749233245
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 351, loss: 0.6267126142978668
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 352, loss: 0.6266033768653869
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 353, loss: 0.6264945328235626
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 354, loss: 0.6263861358165741
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 355, loss: 0.6262781381607055
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 356, loss: 0.6261705696582794
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 357, loss: 0.6260634005069733
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 358, loss: 0.6259566128253937
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 359, loss: 0.6258502781391144
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 360, loss: 0.6257443249225616
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 361, loss: 0.625638735294342
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 362, loss: 0.6255336046218872
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 363, loss: 0.6254288256168365
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 364, loss: 0.6253244698047637
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 365, loss: 0.6252205073833466
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 366, loss: 0.6251169204711914
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 367, loss: 0.6250137329101563
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 368, loss: 0.6249109327793121
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 369, loss: 0.6248084843158722
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 370, loss: 0.6247064232826233
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 371, loss: 0.6246047496795655
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 372, loss: 0.6245034575462342
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 373, loss: 0.6244025290012359
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 374, loss: 0.6243019819259643
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 375, loss: 0.6242017984390259
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 376, loss: 0.6241020023822784
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 377, loss: 0.6240025460720062
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 378, loss: 0.6239034712314606
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 379, loss: 0.6238047361373902
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 380, loss: 0.6237063884735108
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 381, loss: 0.6236083984375
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 382, loss: 0.6235107243061065
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 383, loss: 0.6234134554862976
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 384, loss: 0.6233165085315704
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 385, loss: 0.6232199132442474
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 386, loss: 0.6231236457824707
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 387, loss: 0.6230277836322784
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 388, loss: 0.6229322373867034
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 389, loss: 0.6228370249271393
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 390, loss: 0.6227421641349793
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 391, loss: 0.622647625207901
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 392, loss: 0.6225534379482269
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 393, loss: 0.6224595785140992
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 394, loss: 0.622366088628769
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 395, loss: 0.6222728848457336
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 396, loss: 0.6221800029277802
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 397, loss: 0.6220874845981598
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 398, loss: 0.6219952762126922
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 399, loss: 0.6219034016132354
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 400, loss: 0.6218118071556091
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 401, loss: 0.6217206239700317
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 402, loss: 0.6216296970844268
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 403, loss: 0.6215390861034393
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 404, loss: 0.6214488208293915
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 405, loss: 0.6213588416576385
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 406, loss: 0.6212691605091095
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 407, loss: 0.6211798250675201
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 408, loss: 0.6210907757282257
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 409, loss: 0.6210020482540131
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 410, loss: 0.6209136426448822
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 411, loss: 0.6208255350589752
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 412, loss: 0.6207377135753631
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 413, loss: 0.620650178194046
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 414, loss: 0.6205629825592041
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 415, loss: 0.6204760611057282
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 416, loss: 0.6203894555568695
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 417, loss: 0.6203031241893768
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 418, loss: 0.620217090845108
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 419, loss: 0.6201313257217407
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 420, loss: 0.6200458705425262
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 421, loss: 0.6199607491493225
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 422, loss: 0.61987584233284
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 423, loss: 0.6197912693023682
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 424, loss: 0.6197069466114045
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 425, loss: 0.6196229577064514
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 426, loss: 0.6195391893386841
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 427, loss: 0.6194557309150696
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 428, loss: 0.6193725824356079
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 429, loss: 0.6192896604537964
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 430, loss: 0.6192070424556733
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 431, loss: 0.6191246747970581
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 432, loss: 0.619042593240738
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 433, loss: 0.6189607739448547
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 434, loss: 0.618879246711731
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 435, loss: 0.6187979638576507
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 436, loss: 0.6187169432640076
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 437, loss: 0.6186362147331238
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 438, loss: 0.6185557603836059
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 439, loss: 0.6184755265712738
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 440, loss: 0.6183955907821655
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 441, loss: 0.6183159112930298
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 442, loss: 0.6182364523410797
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 443, loss: 0.6181572914123535
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 444, loss: 0.6180783808231354
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 445, loss: 0.6179997384548187
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 446, loss: 0.6179213285446167
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 447, loss: 0.6178431808948517
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 448, loss: 0.6177652835845947
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 449, loss: 0.6176876187324524
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 450, loss: 0.6176102340221405
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 451, loss: 0.6175330638885498
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 452, loss: 0.6174561560153962
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 453, loss: 0.617379504442215
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 454, loss: 0.617303091287613
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 455, loss: 0.617226904630661
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 456, loss: 0.6171509683132171
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 457, loss: 0.6170752882957459
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 458, loss: 0.6169998288154602
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 459, loss: 0.6169246375560761
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 460, loss: 0.6168496370315552
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 461, loss: 0.6167748808860779
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 462, loss: 0.616700392961502
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 463, loss: 0.6166261076927185
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 464, loss: 0.616552060842514
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 465, loss: 0.6164782226085663
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 466, loss: 0.6164046823978424
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 467, loss: 0.6163313269615174
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 468, loss: 0.6162581861019134
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 469, loss: 0.6161853075027466
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 470, loss: 0.6161126077175141
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 471, loss: 0.616040176153183
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 472, loss: 0.6159679532051087
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 473, loss: 0.615895938873291
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 474, loss: 0.6158241510391236
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 475, loss: 0.6157526075839996
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 476, loss: 0.6156812727451324
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 477, loss: 0.615610134601593
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 478, loss: 0.6155392408370972
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 479, loss: 0.6154685199260712
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 480, loss: 0.6153980553150177
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 481, loss: 0.6153278112411499
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 482, loss: 0.6152577519416809
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 483, loss: 0.6151879310607911
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 484, loss: 0.6151182770729064
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 485, loss: 0.6150488734245301
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 486, loss: 0.6149796485900879
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 487, loss: 0.6149106681346893
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 488, loss: 0.614841878414154
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 489, loss: 0.614773279428482
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 490, loss: 0.6147048950195313
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 491, loss: 0.6146367490291595
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 492, loss: 0.6145687401294708
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 493, loss: 0.6145009815692901
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 494, loss: 0.6144334256649018
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 495, loss: 0.6143660366535186
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 496, loss: 0.6142988502979279
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 497, loss: 0.6142318964004516
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 498, loss: 0.6141651332378387
Dev F1 tensor([0.8750])
  0%|          | 0/10 [00:00<?, ?it/s]
epoch 499, loss: 0.6140985250473022
Dev F1 tensor([0.8750])
```

Task 3.7: Get the predictions on the Test Set using the Trained model and print the F1 score (10 Points)

In [50]:

```python
### DO NOT EDIT ###

### DEV SET RESULTS

dev_features, dev_labels = featurize_data(dev_texts, dev_labels)
print('Logistic Regression Results:')
print('Accuracy:', accuracy(predict(trained_model, dev_features), dev_labels))
print('F1-score', f1_score(predict(trained_model, dev_features), dev_labels))
```

```
Logistic Regression Results:
Accuracy: 0.868421052631579
F1-score tensor([0.9057])
```

In [51]:

```python
### DO NOT EDIT ###
### RUN THIS ONLY ON DEADLINE ###
### TEST SET RESULTS

test_features, test_labels = featurize_data(test_texts, test_labels)
print('Logistic Regression Results:')
print('Accuracy:', accuracy(predict(trained_model, test_features), test_labels))
print('F1-score', f1_score(predict(trained_model, test_features), test_labels))
```

```
Logistic Regression Results:
Accuracy: 0.84
F1-score tensor([0.8621])
```

Written Assignment (60 Points)

Written assignment tests the understanding of the student for the assignment's task. We have split the writing into sections. You will need to write 1-2 paragraphs describing the sections. Please be concise.

In your own words, describe what the task is (20 points)

Describe the task, how is it useful and an example.

Section 1: Sentiment Classification Dataset:

We have two datasets of a collection of hotel reviews (positive and negative), both datasets are one review per line. The reviews are not tokenized, or sentence segmented, and the words are space separated. We need to explore the distribution of the reviews (positive and negative), create training and test datasets (split the original dataset), and define functions for evaluation metrics (compute accuracy, precision, recall, and f1 score)

Section 2: Baselines:

We need to create two baselines for Random Chance and Naïve Bayes Classifier. The Random Chance predict the label according to the labels' distribution. The Naïve Bayes Classifier using tokens in the training samples. We need preprocess the set and implement fit and predict methods. We will compare these two baselines by f1 score on same sample set.

Section 3: Logistic Regression on Features:

We need to build a logistic regression based on hand-engineered features. We need to implement feature extraction to featurize text for a classification system for sentiment analysis. We need to have a normalization formula to normalize dataset for control the scales before passing it to the classifier. We need to implement weights and logistic loss function and convert logits into prediction functions. We need to use different epochs and learning rate to improve the result.

Describe your method for the task (10 points)

Important details about the implementation. Feature engineering, parameter choice etc.

Section 1: Sentiment Classification Dataset:

We use util.py to load the dataset. We use a for-loop to iterate items in labels: whenever we get label == 0, we increase negative count by one, same logic for positive reviews. I use pie chart for visualization. For split the dataset, I transform the list object into DataFrame first, then shuffle indices using numpy.random.shuffle to split the dataset randomly. I use DataFrame.values.tolist() to transform DataFrame back to list object.

For evaluation metrics, we use a for-loop to compare values between predicted_labels and true_labels, whenever we get same values equal to 1, we increase TP (true positive) by 1. Accuracy (correct prediction / all prediction), Precision (True Positive / All Positive Prediction), Recall (True Positive / All Positive Labels), and F1 Score (2 *Precision* Recall / (Recall + Precision))

Section 2: Baselines:

I use NumPy library to randomly generate value set based on training distributions. I use spacy to tokenize the text and help extract features from text. I preprocess text using lemma_ function and remove stopwords using is_stop function.

The fit method for Naïve Bayes Classifier I need to group samples by their labels and preprocess each text. I count the words of the text for each label through a for-loop and dictionary, whenever we get a word, we update the dictionary. The predict method for Naïve Bayes Classifier I need to compute each word probability by calculate the count of that word divided by whole word count. I use LaPlace smoothing, whenever I get a new word not in the training dictionary, I add 1 to it. I use math.log to sum all probability, return result by compared positive probability and negative probability.

Section 3: Logistic Regression on Features:

I implement six features exactly showed in the table: count (positive/negative lexicon), count (1st and 2nd pronouns), add 1 if get 'no' or '!', and log(word count of doc). Normalize torch.Tensor by iterating a 2D array and update values based on the formula. I initialize the weight using built-in library in torch, which is torch.nn.init.xavier*uniform*. I use built-in library in torch to implement logistic loss function, which is torch.nn.BCEWithLogitsLoss. I convert logits into prediction by using built-in library torch, which is torch.round.

Experiment Results (10 points)

Typically a table summarizing all the different experiment results for various parameter choices

Section 1: Sentiment Classification Dataset:

All dataset

# of examples have label = 0	# of examples have label = 1
94	95

Train Label Distribution

# of examples have label = 0	# of examples have label = 1
74	77

Dev Label Distribution

# of examples have label = 0	# of examples have label = 1
20	18

Section 2: Baselines:

Dev Set Result

Random Chance F1	Naïve Bayes F1
0.5945945945945946	0.9090909090909091

Test Set Result (Deadline)

Random Chance F1	Naïve Bayes F1
0.491228	0.90196

Section 3: Logistic Regression on Features:

With Epoch = 500 and learning rate = 0.5
Dev Set Result at Epoch 500

Loss	F1
0.60735	0.9412

Dev Set Result/Logistic Regression Results

Accuracy	F1-score tensor
0.868421052631579	0.9057

Test Set Result/Logistic Regression Results (Deadline)

Accuracy	F1-score tensor
0.84	0.8621

Discussion (20 points)

Key takeaway from the assignment. Why is the method good? shortcomings? how would you improve? Additional thoughts?

Section 1: Sentiment Classification Dataset:

Use pie chart to visualize the distribution of positive and negative reviews. Although there are many other kinds of charts to explore the distribution, pie chart is the most intuitive ways for distribution. Use pandas library to split the dataset into training and test set by random shuffle the indices. However, there are many other ways to split the dataset but I'm more experience with pandas rather to transform original list object into DataFrame object than try other methods. Implement the evaluation metrics based on formula helps me to further understand accuracy, precision, recall and f1 score. I believe there are some libraries can automatically compute these values.

Section 2: Baselines:

Use numpy.random.choice to generate 0 and 1's based on distribution. Understand how to use spacy to remove stop words and tokenize sentences into words. Dictionary is quite useful when I implement fit method in Naïve Bayes Classifier. I update dictionary for counting whenever I iterate a word in the sentence. The predict method I need to create another two dictionaries for probability of each word from previous counting dictionary. The Laplace smoothing method helps us to calculate probability when the word is not in our dictionary. I use log function because sum all log is much faster than multiple all the probabilities.

Section 3: Logistic Regression on Features:

Use three built-in library in torch.tensor to initialize weights, logistic loss function, and covert logits into predictions. Try different combination of epoch and learning rate to get relatively good performance. In this assignment I only use one method to initialize weights and there are many more other built-in function to initialize weights that maybe can improve the result. On the other hand, I can implement more engineering features to improve feature extraction. For example, the count of third pronouns in the whole text, the count of comma in the whole text, and etc. The combination of epochs and learning rate can be tuned using exist library for parameter tuning.