Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Для быстрого выполнения просмотрите [семинар](#).

Models: Sentence Sentiment Classification

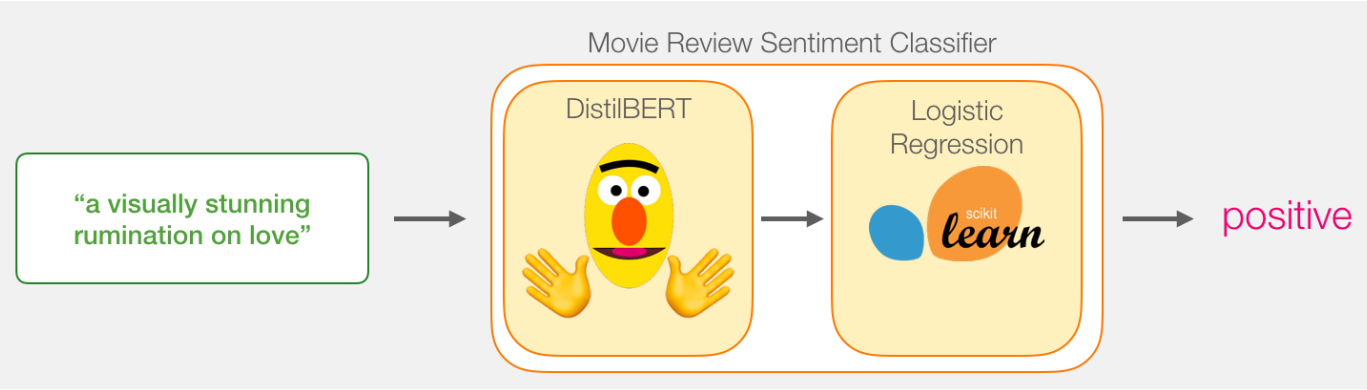
Our goal is to create a model that takes a sentence (just like the ones in our dataset) and produces either 1 (indicating the sentence carries a positive sentiment) or a 0 (indicating the sentence carries a negative sentiment). We can think of it as looking like this:

Under the hood, the model is actually made up of two model.

- DistilBERT processes the sentence and passes along some information it extracted from it on to the next model. DistilBERT is a smaller version of BERT developed and open sourced

by the team at HuggingFace. It's a lighter and faster version of BERT that roughly matches its performance.
- The next model, a basic Logistic Regression model from scikit learn will take in the result of DistilBERT's processing, and classify the sentence as either positive or negative (1 or 0, respectively).

The data we pass between the two models is a vector of size 768. We can think of this of vector as an embedding for the sentence that we can use for classification.



Dataset

The dataset we will use in this example is [SST2](), which contains sentences from movie reviews, each labeled as either positive (has the value 1) or negative (has the value 0):

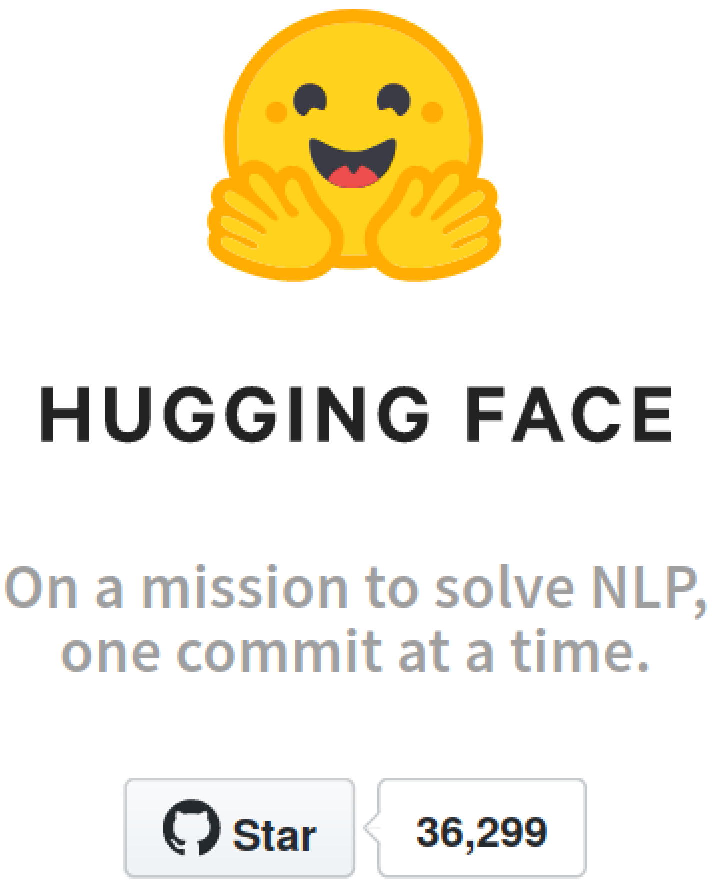
sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

Installing the transformers library

Let's start by installing the huggingface transformers library so we can load our deep learning NLP model.

```
!pip install transformers
```

[Transformers library doc]()




```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import warnings
warnings.filterwarnings('ignore')
```

```
!nvidia-smi
```

```
Mon Nov 22 22:00:37 2021
+-----------------------------------------------------------------------------
| NVIDIA-SMI 450.119.04   Driver Version: 450.119.04   CUDA Version: 11.0
|-------------------------------+----------------------+----------------------
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|                               |                      |               MIG M.
|===============================+======================+======================
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0
| N/A   60C    P0    39W / 250W |   3511MiB / 16280MiB |      0%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------
```

```
+-----------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|        ID   ID                                                   Usage
|=============================================================================
+-----------------------------------------------------------------------------
```

Importing the dataset

```
df = pd.read_csv(
    'https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/S
    delimiter='\t',
    header=None
)
print(df.shape)
df.head()
```

```
(6920, 2)
```

	0	1
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1

```
np.array(df[1])[1]
```

```
0
```

Using BERT for text classification.

Let's now load a pre-trained BERT model.

```
# For DistilBERT, Load pretrained model/tokenizer:

model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.Disti
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

```
# look at the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
model.eval()
```

```
              ,
              (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tr
```

```
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affin
      )
      (3): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tr
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affin
      )
      (4): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tr
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affin
      )
      (5): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tr
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affin
      )
    )
  )
```

```
print(tokenizer.encode("Hi"))
```

```
[101, 7632, 102]
```

```
from termcolor import colored

colors = ['red', 'green', 'blue', 'yellow']

def model_structure(layer, margin=0, item_color=0):
    for name, next_layer in layer.named_children():

        next = (0 if not list(next_layer.named_children()) else 1)
        print(colored(' ' * margin + name, colors[item_color]) + ':' * next)
        model_structure(next_layer, margin + len(name) + 2, (item_color + 1) % 4)

# model_structure(model)
```

Preparing the dataset

```
from torch.utils.data import Dataset, random_split

class ReviewsDataset(Dataset):
    def __init__(self, reviews, tokenizer, labels):
        self.labels = labels
        # tokenized reviews
        self.tokenized = {i: tokenizer.encode(reviews[i]) for i in range(len(revie

    def __getitem__(self, idx):
        return {"tokenized": self.tokenized[idx], "label": self.labels[idx]}

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

dataset = ReviewsDataset(np.array(df[0]), tokenizer, np.array(df[1]))

# DON'T CHANGE, PLEASE
train_size, val_size = int(.8 * len(dataset)), int(.1 * len(dataset))
torch.manual_seed(2)
train_data, valid_data, test_data = random_split(dataset, [train_size, val_size, le

print(f"Number of training examples: {len(train_data)}")
print(f"Number of validation examples: {len(valid_data)}")
print(f"Number of testing examples: {len(test_data)}")
```

```
Number of training examples: 5536
Number of validation examples: 692
Number of testing examples: 692
```

```
from torch.utils.data import Sampler

class ReviewsSampler(Sampler):
    def __init__(self, subset, batch_size=32):
        self.batch_size = batch_size
```

```
        self.subset = subset

        self.indices = subset.indices
        # tokenized for our data
        self.tokenized = []
        for i in self.indices:
            self.tokenized.append(subset.dataset.tokenized[i])
        self.tokenized = np.array(self.tokenized)
    def __iter__(self):

        batch_idx = []
        # index in sorted data
        for index in np.argsort(list(map(len, self.tokenized))):
            batch_idx.append(index)
            if len(batch_idx) == self.batch_size:
                yield batch_idx
                batch_idx = []

        if len(batch_idx) > 0:
            yield batch_idx

    def __len__(self):
        return len(self.dataset)
```

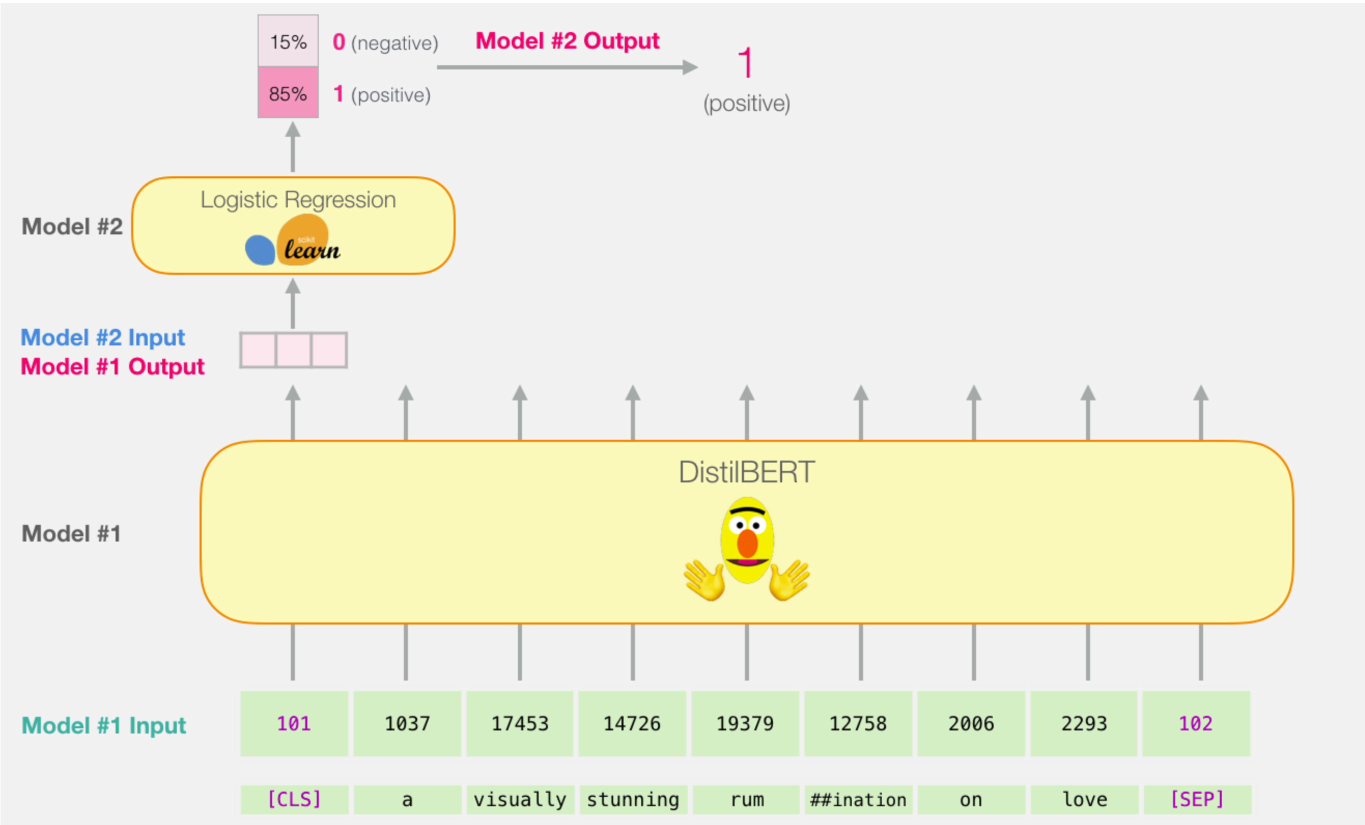
```
a = []
for i in train_data.indices:
    a.append(np.array(train_data.dataset.tokenized[i]))
a = np.array(a)
print(a)
print(a.shape)
```

```
[array([  101, 11552,  2135,  2550,  1998, 22570,  2135,  2864,  1010,
        1996,  2416,  3315,  3616,  6121,  3669,  4371,  3145,  5436,
        5312,  2046,  3371,  2135,  6851, 16278,  1997,  3959, 10359,
       19069,   102])
 array([  101,  2074,  2178, 12391,  3689,  2008,  2038,  2498,  2183,
        2005,  2009,  2060,  2084,  2049, 18077,  3512,  9140,  1997,
       26471, 10036, 16959,  2015,   102])
 array([  101,  1037, 10973, 17197,  4038,  2895, 17037,  2008,  1005,
        2222,  2404, 13606,  2006,  2115,  3108,   102])
 ... array([  101, 12391, 10874, 18015,   102])
 array([  101,  1037,  2143,  1997,  8680,  4094,  2007,  2019, 10305,
        3110,  1010,  1037, 12312,  1997,  1996, 11139,  1998, 12482,
        1997, 28956,   102])
 array([ 101, 2025, 2012, 2035, 3154, 2054, 2009, 1005, 1055, 2667, 2000,
       2360, 1998, 2130, 2065, 2009, 2020, 1045, 4797, 2009, 2052, 2022,
       2035, 2008, 5875,  102])]
(5536,)
```

```
from torch.utils.data import DataLoader

def get_padded(values):
    max_len = 80
    # for value in values:
    #     if len(value) > max_len:
```

```
    #         max_len = len(value)

    padded = np.array([value + [0]*(max_len-len(value)) for value in values])

    return padded

def collate_fn(batch):

    inputs = []
    labels = []
    for elem in batch:
        inputs.append(elem['tokenized'])
        labels.append(elem['label'])

    inputs = get_padded(inputs) # padded inputs
    attention_mask = np.where(inputs != 0, 1, 0)

    return {"inputs": torch.tensor(inputs), "labels": torch.FloatTensor(labels), 'a

train_loader = DataLoader(train_data, batch_sampler=ReviewsSampler(train_data), co
valid_loader = DataLoader(valid_data, batch_sampler=ReviewsSampler(valid_data), co
test_loader = DataLoader(test_data, batch_sampler=ReviewsSampler(test_data), colla
```

Baseline




```
from tqdm.notebook import tqdm

def get_xy(loader, model):
```

```
    features = []
    labels = []

    with torch.no_grad():
        for batch in tqdm(loader):

            attn_mask = batch["attention_mask"].to(device)
            inputs = batch["inputs"].to(device)

            with torch.no_grad():
                last_hidden_states = model(inputs, attention_mask=attn_mask)

            features.append(last_hidden_states[0].cpu())
            labels.append(batch["labels"])

    features = torch.cat([elem[:, 0, :] for elem in features], dim=0).numpy()
    labels = torch.cat(labels, dim=0).numpy()

    return features, labels

train_features, train_labels = get_xy(train_loader, model)
valid_features, valid_labels = get_xy(valid_loader, model)
test_features, test_labels = get_xy(test_loader, model)
```

```
0it [00:00, ?it/s]
0it [00:00, ?it/s]
0it [00:00, ?it/s]
```

```
lr_clf = LogisticRegression(C = 1.2, solver='liblinear')
lr_clf.fit(train_features, train_labels.astype(int))
lr_clf.score(test_features, test_labels)
```

```
0.8208092485549133
```

Fine-Tuning BERT

Define the model

```
from torch import nn

class BertClassifier(nn.Module):
    def __init__(self, pretrained_model, hid_dim, dropout=0.1):
        super().__init__()

        self.bert = pretrained_model
        self.dropout = nn.Dropout(p=dropout)
        self.relu = nn.ReLU()
        self.hid_dim = hid_dim

        self.fc = nn.Linear(hid_dim * 80, 1)

        self.sigm = nn.Sigmoid()
```

```
    def forward(self, inputs, attention_mask):

        predictions = self.bert(inputs, attention_mask=attention_mask).last_hidden_
        predictions = self.relu(predictions)
        predictions = predictions.reshape(-1, self.hid_dim*80)
        probs = self.fc(predictions)
        probs = self.sigm(probs)

        # proba = [batch_size, ] - probability to be positive
        return probs

import torch.optim as optim

# DON'T CHANGE
model = model_class.from_pretrained(pretrained_weights).to(device)
bert_clf = BertClassifier(model, 768).to(device)
# you can change
optimizer = optim.Adam(bert_clf.parameters(), lr=2e-5)
criterion = nn.BCELoss()




def train(model, iterator, optimizer, criterion, clip, train_history=None, valid_hi
    model.train()

    epoch_loss = 0
    history = []
    for i, batch in enumerate(iterator):

        # don't forget about .to(device)
        inputs = batch["inputs"].to(device)
        mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        optimizer.zero_grad()

        output = model(inputs, mask).reshape(-1)

        loss = criterion(output, labels)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

        history.append(loss.cpu().data.numpy())
        if (i+1)%10==0:
            fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))

            clear_output(True)
            ax[0].plot(history, label='train loss')
            ax[0].set_xlabel('Batch')
            ax[0].set_title('Train loss')
```

```
            if train_history is not None:
                ax[1].plot(train_history, label='general train history')
                ax[1].set_xlabel('Epoch')
            if valid_history is not None:
                ax[1].plot(valid_history, label='general valid history')
            plt.legend()

            plt.show()


    return epoch_loss / (i + 1)

def evaluate(model, iterator, criterion):

    model.eval()

    epoch_loss = 0

    history = []

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            inputs = batch["inputs"].to(device)
            mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            output = model(inputs, mask).reshape(-1)

            loss = criterion(output, labels)

            epoch_loss += loss.item()

    return epoch_loss / (i + 1)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

```
import time
import math
import matplotlib
matplotlib.rcParams.update({'figure.figsize': (16, 12), 'font.size': 14})
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import clear_output
```

```
train_history = []
valid_history = []
```

```
N_EPOCHS = 3
CLIP = 1

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(bert_clf, train_loader, optimizer, criterion, CLIP, train_hi
    valid_loss = evaluate(bert_clf, valid_loader, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(bert_clf.state_dict(), 'best-val-model.pt')

    train_history.append(train_loss)
    valid_history.append(valid_loss)
    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}
```

Train loss

```
a = torch.tensor([1, 2, 3, 4, 5])
print(np.array((a>3).to(torch.int)))
```

```
[0 0 0 1 1]
```

```
best_model = BertClassifier(model, 768).to(device)
best_model.load_state_dict(torch.load('best-val-model.pt'))

pred_labels = []
true_labels = []

best_model.eval()
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_loader)):
        inputs = batch["inputs"].to(device)
        mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        true_labels.append(labels.cpu().numpy())

        output = best_model(inputs, mask).reshape(-1)
        pred_labels.append(np.array((output>0.5).to(torch.int).cpu()))
```

```
0it [00:00, ?it/s]
```

```
from sklearn.metrics import accuracy_score

true_labels = np.concatenate(true_labels, axis=0)
pred_labels = np.concatenate(pred_labels, axis=0)
accuracy_score(true_labels, pred_labels)
```

```
0.865606936416185
```

```
assert accuracy_score(true_labels, pred_labels) >= 0.86
```

Finetuned model from **HUGGING FACE**

BertForSequenceClassification

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# we have the same tokenizer
# new_tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased-finetuned
new_model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-un
```

```
Downloading:   0%|          | 0.00/629 [00:00<?, ?B/s]
Downloading:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

```
pred_labels = []
```

```
true_labels = []

new_model.eval()
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_loader)):
        inputs = batch["inputs"].to(device)
        mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        true_labels.append(labels.cpu().numpy())

        output = new_model(inputs, mask).logits
        pred_labels.append(np.array(torch.argmax(output, dim=-1).cpu()))


true_labels = np.concatenate(true_labels, axis=0)
pred_labels = np.concatenate(pred_labels, axis=0)
accuracy_score(true_labels, pred_labels)
```

```
0it [00:00, ?it/s]
0.9841040462427746
```

```
model_structure(new_model)
```

```
                 dropout
                 lin1
                 lin2

            output_layer_norm
        2:
            attention:
                      dropout
                      q_lin
                      k_lin
                      v_lin
                      out_lin
            sa_layer_norm
            ffn:
                 dropout
                 lin1
                 lin2
            output_layer_norm
        3:
            attention:
                      dropout
                      q_lin
                      k_lin
                      v_lin
                      out_lin
            sa_layer_norm
            ffn:
                 dropout
                 lin1
                 lin2
            output_layer_norm
        4:
            attention:
                      dropout
```

```
                q_lin
                k_lin
                v_lin
                out_lin
        sa_layer_norm
        ffn:
            dropout
            lin1
            lin2
        output_layer_norm
    5:
        attention:
                dropout
                q_lin
                k_lin
                v_lin
                out_lin
        sa_layer_norm
        ffn:
            dropout
            lin1
            lin2
        output_layer_norm
pre_classifier
classifier
dropout
```

Напишите вывод о своих результатах. В выводы включите ваши гиперпараметры.

Качество с помощью Fine-Tuning должно достигать 0.86.

```
На классификации с логистической регресии точность вышла 0.82
На Fine-Tuning - 0.86. Параметры: max_len=80
На FineTuned - 0.984
```