hugging face

# Hugging face (hub)

<https://huggingface.co/>

Платформа для хранения и презентации моделей

# Hugging face transformers

<https://github.com/huggingface/transformers>

библиотека Transformers предоставляет инструменты и интерфейсы для их простой загрузки и использования моделей с HF hub. Это позволяет вам экономить время и ресурсы, необходимые для обучения моделей с нуля.

## Code

### Формат датасета для NER

именно такой формат принят в библиотеке Hugging Face datasets для датасетов типа NER, и для удобной работы с Trainer из Transformers.

Вот официальные стандарты:

- список tokens (токенов)

- список ner\_tags (меток для этих токенов)

- номера меток — это числа, а их расшифровка — в словаре (label2id и id2label)

label2id = {

"O": 0,

"B-STREET\_TYPE": 1,

"B-STREET\_NAME": 2,

"B-BUILDING": 3

}

id2label = {v: k for k, v in label2id.items()}

data = [

{ "tokens": ["Санкт-Петербург", "г", ",", "ул", "Полоцкая", ",", "15"],

"ner\_tags": [0, 0, 0, 1, 2, 0, 3] # например, 0 — O, 1 — B-STREET\_TYPE, 2 — B-STREET\_NAME, 3 — B-BUILDING }

}

]

### Форматирование датасета

import pandas as pd

import numpy as np

import pandas as pd

from flair.data import Corpus

from flair.datasets import ColumnCorpus

import re, math

columns = ['Building\_Name', 'Building\_Number', 'City', 'Recipient',

'Street\_Name', 'Zip\_Code', 'State', 'Country']

count=0

labels = set()

def generate\_data(file\_name):

data = pd.read\_csv(file\_name)

def get\_value(x):

global count

results = []

address = x["Address"]

previous = ""

\_temp = x.to\_dict()

for each in address.split():

match = False

for col in columns:

if each.strip(" ,.-") in str(\_temp[col]):

if previous == col:

results.append((count, each, "I-"+col))

labels.add("I-"+col)

else:

results.append((count, each, "B-"+col))

labels.add("B-"+col)

previous = col

match = True

break

if not match:

results.append((count, each, "O"))

labels.add("O")

match = False

results.append(("", "",""))

count += 1

return results

values = data[:].apply(lambda x:get\_value(x), axis=1)

flat\_list = [item for sublist in values for item in sublist]

train\_df = pd.DataFrame(flat\_list, columns=["sentence\_id", "words", "labels"])

return train\_df

Weights&Biases (W&B)

Платформа для удобного логирования обучения

Transformers NLP

# Инфо

Токенайзер и модель ДОЛЖНЫ БЫТЬ одинаковыми (ну или из одного семейства), тк индексы эмбеддингов у них могут различаться и все будет шиворот на выворот.

## Начало

Рассмотрим как подойти к решению NLP задачи:

Нам нужно например сделать так чтобы нейросеть из естественного текста выделяла словосочитания, относящиеся к полям - улица, литера, дом.

Более точно: задача классификации токенов на классы - улица, литера, дом

## Определенный датафрейм

Нужно каким либо образом разметить каждое слово из адреса на класс вручную. Обычно датасеты структурированы и можно это автоматизировать с помощью парсинга строк:

1. Сначала я разбил строку по запятым ( + проставил дополнительные запятые там где нужно)
2. и каждый фрагмент разделенный запятыми обрабатывал:
   1. разбивал на слова ( по пробелу )
   2. и каждому слову давал свой класс в зависимости от того какой это фрагмент (все фрагменты идут друг за другом в одинаковом порядке для всех адресов)

## Разметка слов поля на классы

### Ассоциированные слова

1. Важно чтобы нейросеть обращала внимание на нужные слова и игнорировала ненужные. Нужно правильно разметить символы, чтобы
2. Очень важно понимать, чтобы нейросеть ассоциировала словосочетания как ОДНУ улицу

ПРИМЕР:  
1 - класс улица:

1 1  
ул. Губкина

В таком случае нейросеть скажет, что в тексте ДВЕ УЛИЦЫ.

#### BIO

Поэтому делают общепринятое правило BIO

label2id = {

"O": 0,

"B-улица": 1,

"I-улица": 2,

"B-дом": 3,

"I-дом": 4,

"B-корпус": 5,

"I-корпус": 6,

"B-литера": 7,

"I-литера": 8,

"B-квартира": 9,

"I-квартира": 10,

}

id2label = {v: k for k, v in label2id.items()}

1 - класс улица:

1 2 1 2  
ул. Губкина, ул. Мешавина

Теперь нейросеть сможет определить, что в предложении 2 независимые улицы

### Паттерны

Разметка BIO будет зависеть от того **какой паттерн** мы хотим выучить

ул -> 1 или ул**.** -> 1

Мешавина -> 11

Если люди часто пишут улицу как ул. , то нужно чтобы нейросеть замечала именно ул. дальнейшие слова как продолжения ул.

## Токенизация

Первый слой трансформера принимает эмбеддинги, поэтому все слова датасета нужно преобразовать в токены - токенизировать.

Хотя вроде большинство моделей автоматически токенизируют вход, наверно зависит от того как модель загрузить:

from transformers import **AutoModelForTokenClassification**

model = AutoModelForTokenClassification.from\_pretrained(

"DeepPavlov/rubert-base-cased",

num\_labels=len(label2id),

id2label=id2label,

label2id=label2id

).to(device)

### Тэги и токены

Поскольку токены могут разбить слово на части, то и тэги этого слова также разбиваются (дублируются на каждый токен).

Так слово с тэгом 1 может разбиться на два и модель посчитает, что   
 ули -> 1

##ца -> 1

Маркова -> 11

Это две улицы:

1. “ули”
2. “##ца Маркова”

Поэтому ОЧЕНЬ ВАЖНО при токенизации использовать **is\_split\_into\_words=True**:

tokenized\_inputs = tokenizer(

examples["tokens"],

**is\_split\_into\_words=True, # так как уже токены**

truncation=True,

padding="max\_length",

max\_length=128

)

Это сделает так, чтобы токен ул. -> 1 разбитый на 2 токена воспринимался как одно слово с одним тэгом

## HF Обучение

1. GPU автоматически определяется и вычисления будут на gpu просто если перевести модель to.(device)
2. В зависимости от того какую модель загрузить алгоритм сам определит как нужно обучать сеть (from transformers import **AutoModelForTokenClassification**)
3. По умолчанию loss для классификатора стоит CrossEntropyLoss

Но можно и свой поставить

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=test\_dataset,

tokenizer=tokenizer,

loss\_fn=BCEWithLogitsLoss() # изменяем лосс

)

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Пайплайн

## Стандартная загрузка данных

import pandas as pd

data1 = pd.read\_csv("/content/myDrive/MyDrive/life/EXplore/мероприятия/Марафон \"Цифровой Петербург\"/FuzzySearch/данные/fias\_dict.csv")

data2 = pd.read\_csv("/content/myDrive/MyDrive/life/EXplore/мероприятия/Марафон \"Цифровой Петербург\"/FuzzySearch/данные/ojf\_dict2.csv")

data = pd.concat([data1, data2], axis=0, ignore\_index=True)

print(len(data1))

print(len(data2))

print(len(data))

В итоге мы имеем сырые данные даже без целевой переменной, тк она зависит от того, какая у нас задача

id address (цельная строка)

0 4812 Санкт-Петербург г, ул Полоцкая, 15 к. 2 литера...

1 4813 Санкт-Петербург г, ул Полоцкая, 15 к. 2 литера...

## Создание датафрейм с тэгами

Разделение данных по полям

import pandas as pd

data\_parsed = []

for i in range(len(data.iloc[:10000])):

row\_data = []

#for j in range(len(data.columns)):

value = str(data.iloc[i, 1])

if "к." in value:

pos = value.find("к.")

value = value[:pos-1] + "," + value[pos-1:]

pos = value.find("литера")

value = value[:pos-1] + "," + value[pos-1:]

# Разбиваем по запятым

parts = value.split(",")

# Убираем лишние пробелы и пустые строки

parts = [part.strip() for part in parts if part.strip() != ""]

result = []

for i, part in enumerate(parts):

result.append(part)

if i != len(parts) - 1:

result.append(",")

row\_data.extend(result)

data\_parsed.append(row\_data)

# Преобразуем в DataFrame

data\_parsed\_df = pd.DataFrame(data\_parsed)

x = 1000

data.iloc[x:x+20]

data\_parsed\_df.iloc[x:x+10]

### Разметка полей

data\_words = []

data\_tags = []

for row in data\_parsed\_df.iloc[:10000].values:

words\_row = []

tags\_row = []

for field in row:

text = str(field).strip()

words\_field = text.split()

if not words\_field:

continue

tg = get\_label1(words\_field[0])

tags\_field = [tg]

for word in words\_field[1:]:

# если tg == 0 (O), то остаётся 0, иначе метка +1 (например, B → I)

tags\_field.append(tg + 1 if tg != 0 else 0)

words\_row.extend(words\_field)

tags\_row.extend(tags\_field)

data\_words.append(words\_row)

data\_tags.append(tags\_row)

#Итоговый размеченный датафрейм

**data\_words\_df = pd.DataFrame(data\_words)**

**data\_tags\_df = pd.DataFrame(data\_tags)**

x = 6600

data\_words\_df.iloc[x:x+5]

data\_tags\_df.iloc[x:x+5]

### Функция получения класса для слова

import re

def get\_label1(text):

text = str(text).lower().strip()

if 'санкт' in text:

return label2id["O"]

elif re.search(r',', text):

return label2id["O"]

elif re.search(r'(пр|проспект|пер|ул\.?|улица|б-р|бул\.?|пл\.?)', text):

return label2id["B-улица"]

elif re.fullmatch(r'\d+', text) or re.fullmatch(r'\d+-\d+', text) or 'д.' in text:

return label2id["B-дом"]

elif re.search(r'кв|квартира|помещ', text):

return label2id["B-квартира"]

elif re.search(r'к\.?', text):

return label2id["B-корпус"]

elif re.search(r'лит', text):

return label2id["B-литера"]

else:

return label2id["O"] # неизвестное или пустое

### Тэги в LONG

data\_tags\_df = data\_tags\_df.fillna(0).astype(int)

## Датасет HF

### Сбор текущих данных в принятую структуру для HF

data\_final = []

for i in range(len(data\_words\_df)):

tokens = data\_words\_df.iloc[i].dropna().tolist()

tags = data\_tags\_df.iloc[i].dropna().tolist()

data\_final.append({

"tokens": tokens,

"ner\_tags": tags

})

print(data\_final[:3])

### HF dataset

from datasets import Dataset

dataset = Dataset.from\_list(data\_final)

### Токенизация слов датасета

**Токинайзер**

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained("DeepPavlov/rubert-base-cased")

#tokenizer = AutoTokenizer.from\_pretrained("bert-base-multilingual-cased")

**функция преобразующая слова в индексы эмбедингов (inputs\_id)**

def tokenize\_and\_align\_labels(examples):

tokenized\_inputs = tokenizer(

examples["tokens"],

**is\_split\_into\_words=True, # так как уже токены**

truncation=True,

padding="max\_length",

max\_length=128

)

labels = []

for i, label in enumerate(examples["ner\_tags"]):

word\_ids = tokenized\_inputs.word\_ids(batch\_index=i) # соответствие токенов словам

label\_ids = []

previous\_word\_idx = None

for word\_idx in word\_ids:

if word\_idx is None:

label\_ids.append(-100) # паддинги игнорировать в loss'е

elif word\_idx != previous\_word\_idx:

label\_ids.append(label[word\_idx])

else:

label\_ids.append(label[word\_idx]) # можно вставить I-метку, если хочешь

previous\_word\_idx = word\_idx

labels.append(label\_ids)

tokenized\_inputs["labels"] = labels

return tokenized\_inputs

**Применение этой функции ко всему датасету**

tokenized\_dataset = dataset.map(tokenize\_and\_align\_labels, batched=True)

print(tokenized\_dataset)

## Модель

from transformers import AutoModelForTokenClassification

model = AutoModelForTokenClassification.from\_pretrained(

"DeepPavlov/rubert-base-cased",

num\_labels=len(label2id),

id2label=id2label,

label2id=label2id

).to(device)

**архитектура**

print(model)

**Проверка какие веса обучаются**

for name, param in model.named\_parameters():

print(f"{name}: {'обучается' if param.requires\_grad else 'заморожен'}")

**заморозка весов**

1. Заморозка всех базовых весов bert, кроме классификатора

for param in model.base\_model.parameters():

param.requires\_grad = False

1. заморозка/разморозка конкретных слоев

for param in model.base\_model.encoder.layer[11].parameters():

param.requires\_grad = True

for param in model.classifier.parameters():

param.requires\_grad = True

## HF Обучение

from transformers import TrainingArguments

from transformers import Trainer

training\_args = TrainingArguments(

output\_dir='./results', # куда сохранять результаты

learning\_rate=2e-5,

per\_device\_train\_batch\_size=8,

per\_device\_eval\_batch\_size=8,

num\_train\_epochs=3,

weight\_decay=0.01,

logging\_dir='./logs', # куда сохранять логи

)

# Шаг 7: Настройка Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=test\_dataset,

tokenizer=tokenizer,

)

trainer.train()

Deploy

# ONNX

## Инфо

## Обратная токенизация

tokens = tokenizer.convert\_ids\_to\_tokens(inputs\_ids)

tokens = [tokenizer.id\_to\_token(id) for id in ids]

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Экспорт

from transformers import AutoTokenizer, AutoModelForTokenClassification

import onnx

model = AutoModelForTokenClassification.from\_pretrained('/content/mydrive/MyDrive/life/EXplore/мероприятия/Марафон "Цифровой Петербург"/FuzzySearch/Модели/mid\_mybert\_model')

model.eval()

tokenizer = AutoTokenizer.from\_pretrained('/content/mydrive/MyDrive/life/EXplore/мероприятия/Марафон "Цифровой Петербург"/FuzzySearch/Модели/mid\_mybert\_model')

import torch

dummy\_input = tokenizer(

"ул. Маяковская, 48, литера А",

return\_tensors="pt",

padding="max\_length",

truncation=True,

max\_length=128

)

torch.onnx.export(

model,

(dummy\_input["input\_ids"], dummy\_input["attention\_mask"]),

"./onnxModel/midMybert.onnx",

input\_names=["input\_ids", "attention\_mask"],

output\_names=["logits"],

dynamic\_axes={

"input\_ids": {1: "seq\_len"},

"attention\_mask": {1: "seq\_len"},

"logits": {1: "seq\_len"}

},

opset\_version=14 # можно и 11, но 14/15 лучше для новых моделей

)

## 

## Загрузка токенизатора

import onnxruntime

import numpy as np

from tokenizers import Tokenizer

# Загрузка токенизатора из файла (формат .json)

tokenizer = Tokenizer.from\_file("/content/mydrive/MyDrive/life/EXplore/мероприятия/Марафон \"Цифровой Петербург\"/FuzzySearch/Модели/mid\_mybert\_model/tokenizer.json")

## Токенизация

# Текст

text = "ул. Маяковская, 48, литера А"

# Токенизация

encoded = tokenizer.encode(text)

# Преобразование в нужный формат

input\_ids = encoded.ids

attention\_mask = encoded.attention\_mask

# Приведение к фиксированной длине (например, 128)

max\_length = 128

pad\_id = tokenizer.token\_to\_id("[PAD]") # Убедись, что у тебя есть [PAD] в словаре

input\_ids += [pad\_id] \* (max\_length - len(input\_ids))

attention\_mask += [0] \* (max\_length - len(attention\_mask))

# Преобразуем в numpy массивы

input\_ids = np.array([input\_ids], dtype=np.int64)

attention\_mask = np.array([attention\_mask], dtype=np.int64)

## Загрузка и инференс

# Загрузка модели

session = onnxruntime.InferenceSession("./onnxModel/midMybert.onnx")

# Получение выходов

outputs = session.run(

None,

{

"input\_ids": input\_ids,

"attention\_mask": attention\_mask

}

)

# Результат

logits = outputs[0]

print(logits)

## Квантизация

pip install onnxruntime onnx onnxruntime-tools

from onnxruntime.quantization import quantize\_dynamic, QuantType

# Путь к исходной модели

model\_fp32 = "./onnxModel/midMybert.onnx"

# Путь к квантованной модели

model\_quant = "./onnxModel/midMybert\_quant.onnx"

# Квантизация

quantize\_dynamic(

model\_input=model\_fp32,

model\_output=model\_quant,

weight\_type=QuantType.QInt8 # Или QuantType.QUInt8

)

print("Модель успешно квантизована!")