

➤ Библиотеки

[] ↴ Скрыто 5 ячеек.

▼ Работа с train датасетом

```
from google.colab import drive
import shutil
```

```
drive.mount('/content/drive/', force_remount=True)
```

```
dataset = load_dataset(
    'csv',
    data_files=f'/content/train.csv',
    column_names=['id', 'keyword', 'location', 'text', 'labels'],
)
```

Mounted at /content/drive/
Generating train split: 7614/0 [00:00<00:00, 75608.83 examples/s]

dataset

```
DatasetDict({
  train: Dataset({
    features: ['id', 'keyword', 'location', 'text', 'labels'],
    num_rows: 7614
  })
})
```

Наш датасет очень, очень захламлён. Сам автор написал, что могут быть пустые поля в keyword и location. Давайте объективно глянем на датасет. Локаций тут тьма, кто-то пишет млечный путь, кто-то ничего, в общем это будет мешать модели установить закономерности. Также обратим внимание на ссылки, нам не важны сайты, но они начинаются с http:// и их нужно почистить, они не несут полезной информации. Также мусором можно назвать обращения, хэштеги, инородные символы. Хэштег не скажет точно ли катастрофа или нет. Нам с этим очень помогут регулярные выражения. Ну также удалим дубликаты, нам они будут мешать при обучении, как минимум, модель может немного переобучиться от одинаковых примеров. Тут возникла проблема. Я вместе с ненужными тегами удаляю такие теги, как #RIP и т.п. Ну всё же иногда теги занимают чуть ли не половину всего текста. Мы можем с одной стороны дать модели хэштеги и позволить ей самой определять нужные и ненужные, но датасет очень маленький (было бы тысяч 20 хотя бы, а тут 6 тысяч без дубликатов). Мы пойдём на компромисс и превратим хэштеги #god в слова -> god

```
def clean_tweet(text):
    text = re.sub(r'http\S+', '', text) # Удаление ссылок
    text = re.sub(r'@\w+', '', text)   # Удаление упоминаний
    text = re.sub(r'#', '', text)
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'^A-Za-z0-9\s.,;?!~\-', '', text) #удаление всего not-English алфавита
    text = text.strip()                    # Удаление пробелов в начале и конце
    return text
```

```
train_pd = pd.DataFrame(dataset['train'])
train_pd = train_pd.drop('id', axis=1)
train_pd = train_pd.drop('location', axis=1)
train_pd = train_pd.groupby('text', as_index=False).first()
train_pd['text'] = list(map(clean_tweet, train_pd['text']))
train_pd = train_pd.explode('text')
train_pd = train_pd.groupby('text', as_index=False).first()
train_pd = train_pd.drop_duplicates().reset_index(drop=True)
train_pd['labels'] = pd.to_numeric(train_pd['labels'], errors='coerce')
train_pd = train_pd[train_pd['labels'].isin([0, 1]).reset_index(drop=True)]
train_pd['labels'] = train_pd['labels'].astype(int)
train_pd
```



| | text | keyword | labels |
|------|---|-----------------|--------|
| 0 | ! Residents Return To Destroyed Homes As Washi... | wildfire | 1 |
| 1 | - Cleveland Heights Shaker Heights fight bligh... | blight | 0 |
| 2 | - Man gently dabbed the cotton rag across each... | wounds | 0 |
| 3 | - Pandemonium In Aba As Woman Delivers Baby Wi... | pandemonium | 0 |
| 4 | -- causing the entire sky around their battle ... | violent%20storm | 0 |
| ... | ... | ... | ... |
| 6903 | your turn ?? | electrocute | 0 |
| 6904 | youre correct it is coming from both sides. Ma... | inundated | 0 |
| 6905 | youre not injured anymore? ?? | injured | 0 |
| 6906 | youre the snowstorm Im purified. the darkest f... | snowstorm | 0 |
| 6907 | youre too busy finishing those weapon designs | weapon | 0 |

6908 rows × 3 columns

Далее:

[Посмотреть рекомендованные графики](#)[New interactive sheet](#)

```
import matplotlib.pyplot as plt
import seaborn as sns
label_counts = train_pd['labels'].value_counts()
```

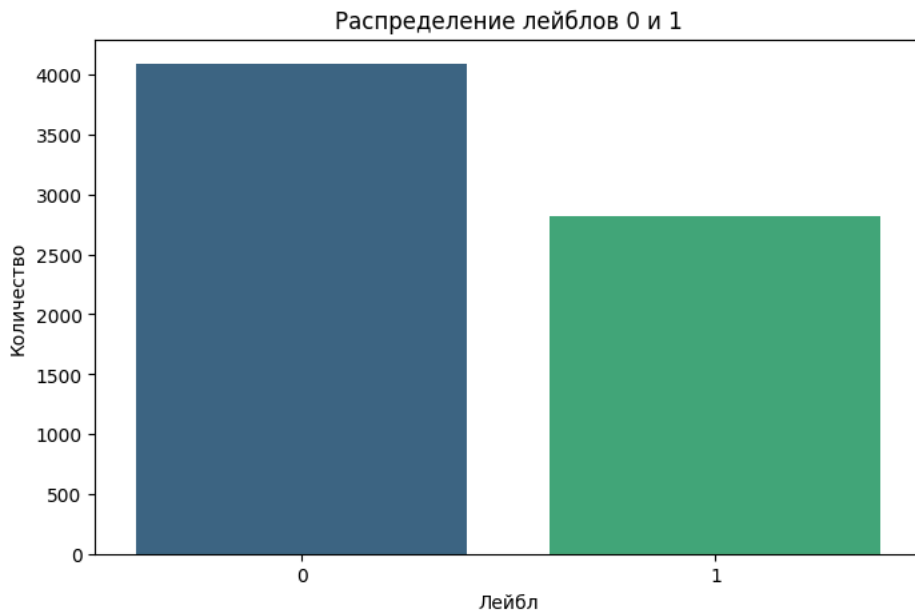
```
# Построение столбчатой диаграммы
plt.figure(figsize=(8, 5))
sns.barplot(x=label_counts.index, y=label_counts.values, palette='viridis')
plt.title('Распределение лейблов 0 и 1')
plt.xlabel('Лейбл')
plt.ylabel('Количество')
plt.xticks(ticks=[0, 1], labels=['0', '1'])
plt.show()
```



<ipython-input-10-a8f9c9680dd1>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

```
sns.barplot(x=label_counts.index, y=label_counts.values, palette='viridis')
```



```
train = Dataset.from_pandas(train_pd)
train
```



```
Dataset({
  features: ['text', 'keyword', 'labels'],
  num_rows: 6908
})
```

Ну это выглядит поприятнее. Далее нам нужно выбрать модель и обучить. Я помню мне говорили взять какой-то простой датасет и обучить на нём две модельки и сравнить. Ну моделей очень много и я бы хотел использовать трансформеры здесь (RNN, LSTM и прочие дают результаты хуже и с ними много мучаться). Поэтому я обучу одну модель с замороженным backbone и размороженным,

сравню их и сделаю выводы. Будет две модели, но я надеюсь, что это не повлияет на оценку. Я планирую в следующем семестре написать что-то мощное, чтобы сдать Елене Владимировне, это проект просто для зачёта. Я разобрал устройство трансформера в виде класса из статьи Attention is all you need и думаю, что будет проще загрузить модель из hugging face. Остаётся только выбрать нужную. Можно вообще взять какую-нибудь базу, типа Bert

Я зашёл на hugging face и взял RoBERT, который обучен на постах из facebook. Bert и RoBERT очень похожи и их реализация практически идентична, но RoBERT немного лучше. К тому же для классификации постов с твиттера будет полезна модель, которая обучена на постах в facebook

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```

/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as :
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%          48.0/48.0 [00:00<00:00, 3.09kB/s]
vocab.txt: 100%                    232k/232k [00:00<00:00, 950kB/s]
tokenizer.json: 100%               466k/466k [00:00<00:00, 1.02MB/s]
config.json: 100%                  570/570 [00:00<00:00, 27.1kB/s]
```

Существует несколько способов токенизации, когда у нас несколько колонок. И самый простой - объединить два столбца и разделить их служебным токеном. Самый простой не только для меня, но и для модели. Хотя я только новичок. Хотя механизм attention вроде как параллельно всё обрабатывает, я не знаю, как будет действовать с несколькими колонками.

```

def tokenize(dataset: Dataset, tokenizer: BertTokenizer):
    combined_text = [f"{text} [KEYWORD] {keyword}" for text, keyword in zip(dataset["text"], dataset["keyword"])]

    tokenized_text = tokenizer(
        combined_text,
        truncation=True,
        padding='max_length',
        max_length=512
    )

    return tokenized_text
```

```

train = train.map(tokenize, batched=True, fn_kwargs={"tokenizer": tokenizer})
train = train.remove_columns(["keyword"])
train
```

```

Map: 100%          6908/6908 [00:05<00:00, 1392.33 examples/s]
Dataset({
  features: ['text', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
  num_rows: 6908
})
```

```

train = train.train_test_split(test_size=0.10).shuffle(seed=42)
train
```

```

DatasetDict({
  train: Dataset({
    features: ['text', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
    num_rows: 6217
  })
  test: Dataset({
    features: ['text', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
    num_rows: 691
  })
})
```

Вот и выполнен Препроцессинг данных. Мы избавились от мусора, как смогли, и токенизировали текст.

✓ Модель

```

class TransformerClassificationModel(nn.Module):
    def __init__(self, base_transformer_model = ('bert-base-uncased'), num_labels: int = 2):
        super().__init__()
        self.num_labels = num_labels
```

```

self.backbone = BertModel.from_pretrained(base_transformer_model)
self.classifier = nn.Linear(self.backbone.config.hidden_size, num_labels)


def forward(
    self,
    input_ids=None,
    attention_mask=None,
    token_type_ids=None,
    position_ids=None,
    head_mask=None,
    inputs_embeds=None,
    labels=None,
    output_attentions=None,
    output_hidden_states=None,
    return_dict=None,
):
    outputs = self.backbone(input_ids, attention_mask)
    embeddings = outputs.last_hidden_state[:, 0, :]
    logits = self.classifier(embeddings)

    loss = None
    if labels is not None:
        loss_fn = nn.CrossEntropyLoss()
        loss = loss_fn(logits.view(-1, self.num_labels), labels.view(-1))

    return SequenceClassifierOutput(
        loss=loss,
        logits=logits,
        hidden_states=outputs.hidden_states,
        attentions=outputs.attentions,
    )

from transformers import BertForSequenceClassification

bert = BertForSequenceClassification.from_pretrained('bert-base-uncased')

 model.safetensors: 100% 440M/440M [00:04<00:00, 102MB/s]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
<
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)

    accuracy = accuracy_score(labels, preds)
    precision = precision_score(labels, preds, average='weighted')
    recall = recall_score(labels, preds, average='weighted')
    f1 = f1_score(labels, preds, average='weighted')

    return {
        'eval_accuracy': accuracy,
        'eval_precision': precision,
        'eval_recall': recall,
        'eval_f1': f1
    }

def freeze_backbone_function(model: TransformerClassificationModel):
    for param in model.backbone.parameters():
        param.requires_grad = False
    return model

```

✓ Обучение с замороженным backbone

```

bert = TransformerClassificationModel()

class MetricsLogger(TrainerCallback):
    def __init__(self):
        self.f1_scores = []
        self.steps = []

    def on_evaluate(self, args, state, control, metrics=None, **kwargs):
        if metrics is not None and 'eval_f1' in metrics:
            self.f1_scores.append(metrics['eval_f1'])
            self.steps.append(state.global_step)

```

```
metrics_logger = MetricsLogger()

import shutil
dir = '/content/models'
shutil.rmtree(f'{dir}/Bert', ignore_errors=True)
model = copy.copy(bert)
model = freeze_backbone_function(model).to(device)

output_dir=f'{dir}/Bert'
batch_size = 256
num_epochs = 15
learning_rate = 1e-1
weight_decay = 0.001
evaluation_strategy = "epoch"
save_total_limit = 3
remove_unused_columns = True
report_to = "none"
padding = True
logging_steps = 1
metric_for_best_model = "eval_f1" # или любое другое метрика
greater_is_better = True # зависит от метрики

training_args = TrainingArguments(
    output_dir=output_dir,
    evaluation_strategy=evaluation_strategy,
    learning_rate=learning_rate,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=weight_decay,
    save_total_limit=save_total_limit,
    num_train_epochs=num_epochs,
    remove_unused_columns=remove_unused_columns,
    report_to=report_to,
    logging_steps=logging_steps,
    load_best_model_at_end=True,
    metric_for_best_model=metric_for_best_model,
    save_strategy="epoch"
)

data_collator = DataCollatorWithPadding(
    tokenizer=tokenizer,
    padding=padding
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train["train"],
    eval_dataset=train["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[metrics_logger]
)

trainer.train()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated and v
warnings.warn(
<ipython-input-23-521224e7f720>:44: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__ini
trainer = Trainer(
```

[34/375 04:06 < 43:51, 0.13 it/s, Epoch 1.32/15]

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|-------|---------------|-----------------|----------|-----------|----------|----------|
| 1 | 0.545600 | 0.808743 | 0.784370 | 0.800995 | 0.784370 | 0.771676 |

[375/375 49:31, Epoch 15/15]

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|-------|---------------|-----------------|----------|-----------|----------|----------|
| 1 | 0.545600 | 0.808743 | 0.784370 | 0.800995 | 0.784370 | 0.771676 |
| 2 | 1.119500 | 1.223617 | 0.730825 | 0.790264 | 0.730825 | 0.693374 |
| 3 | 0.617000 | 0.772295 | 0.696093 | 0.758237 | 0.696093 | 0.695451 |
| 4 | 0.949600 | 1.159930 | 0.703329 | 0.774946 | 0.703329 | 0.652064 |
| 5 | 0.528400 | 0.499053 | 0.817656 | 0.816602 | 0.817656 | 0.816095 |
| 6 | 0.564600 | 0.504178 | 0.811867 | 0.810901 | 0.811867 | 0.809770 |
| 7 | 0.969100 | 1.426244 | 0.568741 | 0.751769 | 0.568741 | 0.530959 |
| 8 | 1.232100 | 0.608037 | 0.768452 | 0.783738 | 0.768452 | 0.770703 |
| 9 | 0.706100 | 0.508716 | 0.807525 | 0.809095 | 0.807525 | 0.808103 |
| 10 | 0.726400 | 0.452463 | 0.810420 | 0.811646 | 0.810420 | 0.810893 |
| 11 | 0.504000 | 0.516380 | 0.765557 | 0.783283 | 0.765557 | 0.767877 |
| 12 | 0.500700 | 0.457056 | 0.811867 | 0.812102 | 0.811867 | 0.811977 |
| 13 | 0.449800 | 0.450179 | 0.810420 | 0.812323 | 0.810420 | 0.811081 |
| 14 | 0.533700 | 0.428337 | 0.819103 | 0.818540 | 0.819103 | 0.816840 |
| 15 | 0.463500 | 0.431114 | 0.820550 | 0.823601 | 0.820550 | 0.816144 |

TrainOutput(global_step=375, training_loss=0.7441137425899506, metrics={'train_runtime': 2980.6104, 'train_samples_per_second':

И тут я столкнулся с ужасной проблемой, а именно с Wi-Fi СУНЦа, который не поддерживает обучение. Я пытался около 13 (уже несколько дней) часов обучать, но всё сбрасывается под конец... Пока я делал эти попытки, я потратил мощности на всех трёх (уже четырёх) google аккаунтах и... У меня остался сри, который почему-то не поддерживает обучение трансформера

[250/250 1:47:40, Epoch 10/10]

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|-------|---------------|-----------------|----------|-----------|----------|----------|
| 1 | 9.932000 | 11.460972 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 2 | 13.091500 | 10.624026 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 3 | 1.567700 | 5.492534 | 0.400868 | 0.460146 | 0.400868 | 0.231948 |
| 4 | 1.406600 | 2.125702 | 0.622287 | 0.768336 | 0.622287 | 0.499319 |
| 5 | 5.466400 | 1.190585 | 0.573082 | 0.671804 | 0.573082 | 0.557895 |
| 6 | 0.778000 | 1.030077 | 0.662808 | 0.674330 | 0.662808 | 0.665784 |
| 7 | 1.846500 | 1.357109 | 0.494935 | 0.620542 | 0.494935 | 0.450190 |
| 8 | 4.180400 | 3.959067 | 0.597685 | 0.358611 | 0.597685 | 0.448263 |
| 9 | 0.740800 | 0.653962 | 0.712012 | 0.757586 | 0.712012 | 0.672369 |
| 10 | 0.782100 | 0.516926 | 0.768452 | 0.766931 | 0.768452 | 0.764061 |

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|-------|---------------|-----------------|----------|-----------|----------|----------|
| 1 | 10.305300 | 11.951322 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 2 | 12.592400 | 9.992604 | 0.597685 | 0.358611 | 0.597685 | 0.448263 |
| 3 | 1.116600 | 4.659231 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 4 | 3.453400 | 4.669896 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 5 | 4.580100 | 2.594554 | 0.567294 | 0.534905 | 0.567294 | 0.532632 |
| 6 | 1.395000 | 1.193070 | 0.638205 | 0.774423 | 0.638205 | 0.531393 |
| 7 | 1.090600 | 1.218599 | 0.622287 | 0.683664 | 0.622287 | 0.511768 |
| 8 | 0.947700 | 0.778547 | 0.645441 | 0.687812 | 0.645441 | 0.647033 |
| 9 | 5.324200 | 2.181677 | 0.461650 | 0.634588 | 0.461650 | 0.377971 |
| 10 | 4.149100 | 1.446784 | 0.617945 | 0.691145 | 0.617945 | 0.498954 |
| 11 | 3.923300 | 4.556591 | 0.599132 | 0.358959 | 0.599132 | 0.448942 |
| 12 | 2.251700 | 1.343438 | 0.636758 | 0.747581 | 0.636758 | 0.532364 |
| 13 | 0.718900 | 0.570683 | 0.764110 | 0.762805 | 0.764110 | 0.758995 |
| 14 | 3.109300 | 1.528565 | 0.664255 | 0.725856 | 0.664255 | 0.594413 |
| 15 | 0.873000 | 0.760654 | 0.623734 | 0.696693 | 0.623734 | 0.619303 |
| 16 | 0.930400 | 0.663370 | 0.735166 | 0.770106 | 0.735166 | 0.706645 |
| 17 | 0.885000 | 0.565336 | 0.690304 | 0.728593 | 0.690304 | 0.692339 |
| 18 | 0.784900 | 0.619988 | 0.707670 | 0.764794 | 0.707670 | 0.662498 |
| 19 | 0.734900 | 0.498731 | 0.756874 | 0.754939 | 0.756874 | 0.755378 |
| 20 | 0.848300 | 0.528310 | 0.752533 | 0.754402 | 0.752533 | 0.753276 |

Вот финальный результат

[375/375 49:31, Epoch 15/15]

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|-------|---------------|-----------------|----------|-----------|----------|----------|
| 1 | 0.545600 | 0.808743 | 0.784370 | 0.800995 | 0.784370 | 0.771676 |
| 2 | 1.119500 | 1.223617 | 0.730825 | 0.790264 | 0.730825 | 0.693374 |
| 3 | 0.617000 | 0.772295 | 0.696093 | 0.758237 | 0.696093 | 0.695451 |
| 4 | 0.949600 | 1.159930 | 0.703329 | 0.774946 | 0.703329 | 0.652064 |
| 5 | 0.528400 | 0.499053 | 0.817656 | 0.816602 | 0.817656 | 0.816095 |
| 6 | 0.564600 | 0.504178 | 0.811867 | 0.810901 | 0.811867 | 0.809770 |
| 7 | 0.969100 | 1.426244 | 0.568741 | 0.751769 | 0.568741 | 0.530959 |
| 8 | 1.232100 | 0.608037 | 0.768452 | 0.783738 | 0.768452 | 0.770703 |
| 9 | 0.706100 | 0.508716 | 0.807525 | 0.809095 | 0.807525 | 0.808103 |
| 10 | 0.726400 | 0.452463 | 0.810420 | 0.811646 | 0.810420 | 0.810893 |
| 11 | 0.504000 | 0.516380 | 0.765557 | 0.783283 | 0.765557 | 0.767877 |
| 12 | 0.500700 | 0.457056 | 0.811867 | 0.812102 | 0.811867 | 0.811977 |
| 13 | 0.449800 | 0.450179 | 0.810420 | 0.812323 | 0.810420 | 0.811081 |
| 14 | 0.533700 | 0.428337 | 0.819103 | 0.818540 | 0.819103 | 0.816840 |
| 15 | 0.463500 | 0.431114 | 0.820550 | 0.823601 | 0.820550 | 0.816144 |

✓ Препроцессинг теста

```
drive.mount('/content/drive/', force_remount=True)
```

```
datasett = load_dataset(
    'csv',
    data_files=f'/content/test.csv',
    column_names=['id','keyword','location', 'text'],
)
```

Mounted at /content/drive/

```
datasett
```

```
DatasetDict({
  train: Dataset({
    features: ['id', 'keyword', 'location', 'text'],
    num_rows: 3264
  })
})
```

```
test_pd = pd.DataFrame(datasett['train'])
test_pd = pd.DataFrame(datasett['train'])
test_pd = test_pd.drop('id', axis=1)
test_pd = test_pd.drop('location', axis=1)
test_pd['text'] = list(map(clean_tweet, test_pd['text']))
```

test_pd

| | keyword | text |
|------|---------|---|
| 0 | keyword | text |
| 1 | None | Just happened a terrible car crash |
| 2 | None | Heard about earthquake is different cities, st... |
| 3 | None | there is a forest fire at spot pond, geese are... |
| 4 | None | Apocalypse lighting. Spokane wildfires |
| ... | ... | ... |
| 3259 | None | EARTHQUAKE SAFETY LOS ANGELES SAFETY FASTENER... |
| 3260 | None | Storm in RI worse than last hurricane. My city... |
| 3261 | None | Green Line derailment in Chicago |
| 3262 | None | MEG issues Hazardous Weather Outlook HWO |
| 3263 | None | CityofCalgary has activated its Municipal Emer... |

3264 rows × 2 columns

Далее:

[Посмотреть рекомендованные графики](#)

[New interactive sheet](#)

```
test_pd = test_pd.drop(0)
```

test_pd

| | keyword | text |
|------|---------|---|
| 1 | None | Just happened a terrible car crash |
| 2 | None | Heard about earthquake is different cities, st... |
| 3 | None | there is a forest fire at spot pond, geese are... |
| 4 | None | Apocalypse lighting. Spokane wildfires |
| 5 | None | Typhoon Soudelor kills 28 in China and Taiwan |
| ... | ... | ... |
| 3259 | None | EARTHQUAKE SAFETY LOS ANGELES SAFETY FASTENER... |
| 3260 | None | Storm in RI worse than last hurricane. My city... |
| 3261 | None | Green Line derailment in Chicago |
| 3262 | None | MEG issues Hazardous Weather Outlook HWO |
| 3263 | None | CityofCalgary has activated its Municipal Emer... |

3263 rows × 2 columns

Далее:

[Посмотреть рекомендованные графики](#)

[New interactive sheet](#)

```
test = Dataset.from_pandas(test_pd)
```

test

```
Dataset({
  features: ['keyword', 'text'],
  num_rows: 3263
})
```

```
test = test.map(tokenize, batched=True, fn_kwargs={"tokenizer": tokenizer})
```

```
test = test.remove_columns(["keyword"])
```

test

```
Map: 100% 3263/3263 [00:03<00:00, 896.89 examples/s]
Dataset({
  features: ['text', 'input_ids', 'token_type_ids', 'attention_mask'],
  num_rows: 3263
})
```



```
predictions = trainer.predict(test)
```

```
print(predictions)
```

```
PredictionOutput(predictions=array([[ -0.0093503 ,  0.3915323 ],
 [ -0.9349357 ,  1.2569785 ],
 [ -0.38660774,  0.843007  ],
 ...,
 [ -0.92728686,  1.7744044 ],
 [ 0.24704064,  0.34554175],
 [ -0.28532642,  0.7368324 ]], dtype=float32), label_ids=None, metrics={'test_runtime': 91.161, 'test_samples_per_second': 35.7
```

```
predictions_array = predictions.predictions
```

```
print(predictions_array)
```

```
[[ -0.0093503  0.3915323 ]
 [ -0.9349357  1.2569785 ]
 [ -0.38660774  0.843007  ]
 ...
 [ -0.92728686  1.7744044 ]
 [ 0.24704064  0.34554175]
 [ -0.28532642  0.7368324 ]]
```

```
predicted_classes = np.argmax(predictions_array, axis=1)
```

```
class_counts = np.bincount(predicted_classes)
```

```
print("Количество элементов в классе 0:", class_counts[0])
```

```
print("Количество элементов в классе 1:", class_counts[1])
```

```
Количество элементов в классе 0: 1951
Количество элементов в классе 1: 1312
```

```
print("Предсказанные классы:", predicted_classes)
```

```
Предсказанные классы: [1 1 1 ... 1 1 1]
```