lab7_short

December 5, 2022

1.Get acquainted with the data of the Polish Cyberbullying detection dataset. Pay special attention to the distribution of the positive and negative examples in the first task as well as distribution of the classes in the second task.

[ ]:
```
!pip install datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting datasets
  Downloading datasets-2.7.1-py3-none-any.whl (451 kB)
     |                 | 451 kB 4.9 MB/s
Collecting xxhash
  Downloading
xxhash-3.1.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (212 kB)
     |                 | 212 kB 66.6 MB/s
Requirement already satisfied: pyarrow>=6.0.0 in
/usr/local/lib/python3.8/dist-packages (from datasets) (9.0.0)
Collecting responses<0.19
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Collecting multiprocess
  Downloading multiprocess-0.70.14-py38-none-any.whl (132 kB)
     |                 | 132 kB 72.1 MB/s
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.8/dist-packages (from datasets) (2.23.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.8/dist-packages
(from datasets) (3.8.3)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-
packages (from datasets) (6.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-
packages (from datasets) (1.21.6)
Requirement already satisfied: dill<0.3.7 in /usr/local/lib/python3.8/dist-
packages (from datasets) (0.3.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-
packages (from datasets) (21.3)
Collecting huggingface-hub<1.0.0,>=0.2.0
  Downloading huggingface_hub-0.11.1-py3-none-any.whl (182 kB)
     |                 | 182 kB 68.4 MB/s
Requirement already satisfied: fsspec[http]>=2021.11.1 in
```

```
/usr/local/lib/python3.8/dist-packages (from datasets) (2022.11.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.8/dist-
packages (from datasets) (4.64.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages
(from datasets) (1.3.5)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->datasets) (1.3.3)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->datasets) (6.0.2)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->datasets) (4.0.2)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.8/dist-
packages (from aiohttp->datasets) (22.1.0)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in
/usr/local/lib/python3.8/dist-packages (from aiohttp->datasets) (2.1.1)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.8/dist-
packages (from aiohttp->datasets) (1.8.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.8/dist-packages (from huggingface-
hub<1.0.0,>=0.2.0->datasets) (4.1.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-
packages (from huggingface-hub<1.0.0,>=0.2.0->datasets) (3.8.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.8/dist-packages (from packaging->datasets) (3.0.9)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-
packages (from requests>=2.19.0->datasets) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->datasets)
(2022.9.24)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->datasets)
(1.24.3)
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
     |                 | 127 kB 77.2 MB/s
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.8/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-
packages (from pandas->datasets) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-
packages (from python-dateutil>=2.7.3->pandas->datasets) (1.15.0)
Installing collected packages: urllib3, xxhash, responses, multiprocess,
huggingface-hub, datasets
  Attempting uninstall: urllib3
```

```
    Found existing installation: urllib3 1.24.3
    Uninstalling urllib3-1.24.3:
      Successfully uninstalled urllib3-1.24.3
Successfully installed datasets-2.7.1 huggingface-hub-0.11.1
multiprocess-0.70.14 responses-0.18.0 urllib3-1.25.11 xxhash-3.1.0
```

[ ]:
```python
from datasets import load_dataset

dataset_1 = load_dataset("poleval2019_cyberbullying", "task01")
dataset_2 = load_dataset("poleval2019_cyberbullying", "task02")
```

[ ]:
```python
import pandas as pd
```

[ ]:
```python
dataset_1['train'][:10]
```

[ ]:
```python
pd1 = pd.DataFrame.from_dict(dataset_1['train'])
pd2 = pd.DataFrame.from_dict(dataset_2['train'])
```

[ ]:
```python
pd1.loc[pd1['label'] > 0] # normal/non-harmful tweets (class: 0) any kind of␣
↪harmful information (class: 1)
```

[ ]:
```
                                                    text  label
9      @anonymized_account @anonymized_account @anony…      1
21     @anonymized_account @anonymized_account No to …      1
39     #Woronicza 17 poseł Halicki oburzony za Bolka…      1
44     @anonymized_account @anonymized_account @anony…      1
53     Nikt nigdy nie rozsiewał takiego smrodu jak @a…      1
…                                                    …    …
10012  RT @anonymized_account Premier @anonymized_acc…      1
10013  Proponuję pozbawić obywatelstwa polskiego i ob…      1
10027  @anonymized_account Zwycięstwa kogo?, czego? B…      1
10029  @anonymized_account @anonymized_account Tobie …      1
10030  @anonymized_account @anonymized_account Mental…      1

[851 rows x 2 columns]
```

[ ]:
```python
pd2.loc[pd2['label'] > 0] #0 (non-harmful), 1 (cyberbullying), 2 (hate-speech)
```

[ ]:
```
                                                    text  label
9      @anonymized_account @anonymized_account @anony…      2
21     @anonymized_account @anonymized_account No to …      2
39     #Woronicza 17 poseł Halicki oburzony za Bolka…      2
44     @anonymized_account @anonymized_account @anony…      1
53     Nikt nigdy nie rozsiewał takiego smrodu jak @a…      1
…                                                    …    …
10012  RT @anonymized_account Premier @anonymized_acc…      2
10013  Proponuję pozbawić obywatelstwa polskiego i ob…      2
```

```
10027  @anonymized_account Zwycięstwa kogo?, czego? B…      2
10029  @anonymized_account @anonymized_account Tobie …      2
10030  @anonymized_account @anonymized_account Mental…      1

[851 rows x 2 columns]
```

2. Train the following classifiers on the training sets (for the task 1 and the task 2):

[ ]:
```
dataset1_train = dataset_1['train']
dataset1_test = dataset_1['test']
dataset2_train = dataset_2['train']
dataset2_test = dataset_2['test']
```

i Bayesian classifier with TF * IDF weighting.

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
```

[ ]:
```
def Bayesian_classifier(dataset_train):
    x_train, y_train = dataset_train['text'], dataset_train['label']
    vectorizer = TfidfVectorizer()
    x_train_tfidf = vectorizer.fit_transform(x_train)

    classifier = GaussianNB()
    #classifier = MultinomialNB()
    classifier.fit(x_train_tfidf.toarray(), y_train)

    return classifier, vectorizer
```

[ ]:
```
classifier_Bayesian1, vectorizer_Bayesian1 = Bayesian_classifier(dataset1_train)
```

[ ]:
```
classifier_Bayesian2, vectorizer_Bayesian2 = Bayesian_classifier(dataset2_train)
```

ii Fasttext text classifier

[ ]:
```
!pip install fasttext
```

[ ]:
```
import fasttext
```

[ ]:
```
def convert_to_fasttext(dataset):
    with open('fasttext.txt', "w") as f:
        for label, text in zip(dataset['label'], dataset['text']):
            f.write(f"__label__{label} {text}\n")
```

[ ]:
```
convert_to_fasttext(dataset1_train)
model_fasttext = fasttext.train_supervised('fasttext.txt')
```

```
[ ]: convert_to_fasttext(dataset2_train)
     model2_fasttext = fasttext.train_supervised('fasttext.txt')
```

iii Transformer classifier (take into account that a number of experiments should be performed for this model).

Tutaj użyłem 3 transformerów Berta, Roberty oraz Polberta.

```
[ ]: import numpy as np
```

```
[ ]: !pip install transformers
```

```
[ ]: from transformers import AutoTokenizer, AutoModelForSequenceClassification,␣
      ↪TrainingArguments, Trainer, DataCollatorWithPadding
```

```
[ ]: def Fine_tuning_with_Trainer(model_name, dataset):
         tokenizer = AutoTokenizer.from_pretrained(model_name)

         dataset_tokenized = dataset.map(lambda x: tokenizer(x["text"],␣
      ↪padding=True, truncation=True, max_length=512))
         training_args = TrainingArguments(
             output_dir='./results',
             per_device_train_batch_size=16,
             per_device_eval_batch_size=64,
             num_train_epochs=3,
             weight_decay=0.01,
         )

         model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
      ↪num_labels=len(set((dataset['train']['label']))))

         trainer = Trainer(
             model=model,
             args=training_args,
             train_dataset=dataset_tokenized["train"],
             eval_dataset=dataset_tokenized["test"],
             tokenizer=tokenizer,
             #compute_metrics=compute_metrics

         )
         trainer.train()
         return model
```

```
[ ]: bert= Fine_tuning_with_Trainer('bert-base-multilingual-cased', dataset_1)
```

```
[ ]: roberta = Fine_tuning_with_Trainer('xlm-roberta-base', dataset_1)
```

[ ]:
```
dkleczek = Fine_tuning_with_Trainer('dkleczek/bert-base-polish-uncased-v1',␣
↪dataset_1)
```

[ ]:
```
bert_2 = Fine_tuning_with_Trainer('bert-base-multilingual-cased', dataset_2)
roberta_2 = Fine_tuning_with_Trainer('xlm-roberta-base', dataset_2)
dkleczek_2 = Fine_tuning_with_Trainer('dkleczek/bert-base-polish-uncased-v1',␣
↪dataset_2)
```

3. Compare the results of classification on the test set. Select the appropriate measures (from accuracy, F1, macro/micro F1, MCC) to compare the results.

Na podstawie uzyskanych wyników jasno widać że najlepiej wypada Transformer(Polbert), następnie fasttext i na końcu Bayesian. Transformery Bert oraz Roberta ustawiają wszystkie predykcje na 1 klasę a i tak uzyskują lepsze accuracy od Bayesiana przez, to że jest znacznie więcej komentarzy neutralnych niż tych negatywnych

[ ]:
```
def get_score(y_true, y_pred):
  acc = accuracy_score(y_true, y_pred)
  f1_macro = f1_score(y_true, y_pred, average='macro')
  f1_micro = f1_score(y_true, y_pred, average='micro')
  mcc = matthews_corrcoef(y_true, y_pred)
  print(f'acc = {acc}, f1_macro = {f1_macro}, f1_micro = {f1_micro}, mcc =␣
↪{mcc}')
  return [acc, f1_macro, f1_micro, mcc]
```

[ ]:
```
res1, res2 = [], []
```

Bayesian

[ ]:
```
from sklearn.metrics import  f1_score, accuracy_score, matthews_corrcoef
```

[ ]:
```
def predict(dataset_test, classifier, vectorizer):
  x_test, y_true = dataset_test['text'], dataset_test['label']
  x_test_tfidf = vectorizer.transform(x_test)
  y_pred = classifier.predict(x_test_tfidf.toarray())

  return get_score(y_true, y_pred)
```

[ ]:
```
res1.append(predict(dataset1_test, classifier_Bayesian1, vectorizer_Bayesian1))
```

```
acc = 0.782, f1_macro = 0.5701858847467252, f1_micro = 0.782, mcc =
0.1428942557422714
```

[ ]:
```
[0.782, 0.5701858847467252, 0.782, 0.1428942557422714]
```

[ ]:
```
res2.append(predict(dataset2_test, classifier_Bayesian2, vectorizer_Bayesian2))
```

```
acc = 0.787, f1_macro = 0.3968305029876156, f1_micro = 0.787, mcc =
0.1282543759318036
```

[ ]: [0.787, 0.3968305029876156, 0.787, 0.1282543759318036]

fasttext

```
[ ]: y_pred1, _ = model_fasttext.predict(dataset1_test['text'])
     y_pred1 = [int(label.split("__label__")[1]) for (label,) in y_pred1]
```

```
[ ]: y_pred2, _ = model2_fasttext.predict(dataset2_test['text'])
     y_pred2 = [int(label.split("__label__")[1]) for (label,) in y_pred2]
```

```
[ ]: res1.append(get_score(dataset1_test['label'], y_pred1))
     res2.append(get_score(dataset2_test['label'], y_pred2))
```

acc = 0.873, f1_macro = 0.5939365453911798, f1_micro = 0.8729999999999999, mcc = 0.2650301059500807
acc = 0.868, f1_macro = 0.36843539780455736, f1_micro = 0.868, mcc = 0.16001981125515372

transformers

```
[ ]: def compute_metrics(p):
         pred, labels = p
         pred = np.argmax(pred, axis=1)
         sc = get_score(labels, pred)
         t = ['acc', 'f1_macro', 'f1_micro', 'mcc']
         return {t[i]:sc[i] for i in range(len(t))}
```

```
[ ]: def test_transformer(model, model_name, dataset):

       tokenizer = AutoTokenizer.from_pretrained(model_name)
       tokenized_dt = dataset.map(lambda x: tokenizer(x["text"], truncation=True),␣
      ↪batched=True)

       trainer = Trainer(model=model,
                         eval_dataset=tokenized_dt,
                         tokenizer=tokenizer,
                         compute_metrics=compute_metrics)
       ev = trainer.evaluate()
       return [v for k,v in ev.items() if k in␣
      ↪['eval_acc','eval_f1_macro','eval_f1_micro','eval_mcc']]
```

```
[ ]: b1 = test_transformer(bert, 'bert-base-multilingual-cased', dataset1_test)
     r1 = test_transformer(roberta, 'xlm-roberta-base', dataset1_test)
     d1 = test_transformer(dkleczek, 'dkleczek/bert-base-polish-uncased-v1',␣
      ↪dataset1_test)
```

```
[ ]: b2 = test_transformer(bert_2, 'bert-base-multilingual-cased', dataset2_test)
     r2 = test_transformer(roberta_2, 'xlm-roberta-base', dataset2_test)
```

```
d2 = test_transformer(dkleczek_2, 'dkleczek/bert-base-polish-uncased-v1',␣
↪dataset2_test)
```

[ ]:
```
res1.append(b1)
res1.append(r1)
res1.append(d1)

res2.append(b2)
res2.append(r2)
res2.append(d2)
```

[ ]:
```
df1 = pd.DataFrame(res1, columns= ['model', 'accuracy', 'F1 macro', 'F1 micro',␣
↪'MCC'])
df2 = pd.DataFrame(res2, columns= ['model', 'accuracy', 'F1 macro', 'F1 micro',␣
↪'MCC'])
```

[ ]:
```
df1
```

[ ]:

	model	accuracy	F1 macro	F1 micro	MCC
0	Bayesian	0.782	0.570186	0.782	0.142894
1	Fasttext	0.873	0.593937	0.873	0.265030
2	Bert	0.866	0.464094	0.866	0.000000
3	Roberta	0.866	0.464094	0.866	0.000000
4	Polbert	0.902	0.741463	0.902	0.509538

[ ]:
```
df2
```

[ ]:

	model	accuracy	F1 macro	F1 micro	MCC
0	Bayesian	0.787	0.396831	0.787	0.128254
1	Fasttext	0.868	0.368435	0.868	0.160020
2	Bert	0.866	0.309396	0.866	0.000000
3	Roberta	0.866	0.309396	0.866	0.000000
4	Polbert	0.891	0.541534	0.891	0.447937

4. Select 1 TP, 1 TN, 1 FP and 1 FN from your predictions (for the best classifier) and compare the decisions of each classifier on these examples using LIME.

best classifier = Polbert

[ ]:
```
def make_predictions(model, dataset):
    tokenizer = AutoTokenizer.from_pretrained("dkleczek/
↪bert-base-polish-uncased-v1")
    tokenized_data = [tokenizer(x, truncation=True) for x in dataset]
    trainer = Trainer(
        model=model,
        tokenizer=tokenizer)
    return trainer.predict(tokenized_data).predictions
```

[ ]:
```
type(dataset1_test['text'])
```

[ ]: list

[ ]:
```
predictions = make_predictions(dkleczek, dataset1_test['text'])
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--dkleczek--bert-base-polish-
uncased-v1/snapshots/62be9821055981deafb23f217b68cc41f38cdb76/config.json
Model config BertConfig {
  "_name_or_path": "dkleczek/bert-base-polish-uncased-v1",
  "architectures": [
    "BertForMaskedLM",
    "BertForPreTraining"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.25.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 60000
}

loading file vocab.txt from cache at /root/.cache/huggingface/hub/models--
dkleczek--bert-base-polish-
uncased-v1/snapshots/62be9821055981deafb23f217b68cc41f38cdb76/vocab.txt
loading file tokenizer.json from cache at None
loading file added_tokens.json from cache at None
loading file special_tokens_map.json from cache at
/root/.cache/huggingface/hub/models--dkleczek--bert-base-polish-uncased-v1/snaps
hots/62be9821055981deafb23f217b68cc41f38cdb76/special_tokens_map.json
loading file tokenizer_config.json from cache at
/root/.cache/huggingface/hub/models--dkleczek--bert-base-polish-uncased-v1/snaps
hots/62be9821055981deafb23f217b68cc41f38cdb76/tokenizer_config.json
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--dkleczek--bert-base-polish-
uncased-v1/snapshots/62be9821055981deafb23f217b68cc41f38cdb76/config.json
Model config BertConfig {
  "_name_or_path": "dkleczek/bert-base-polish-uncased-v1",
  "architectures": [
    "BertForMaskedLM",
    "BertForPreTraining"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.25.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 60000
}

loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--dkleczek--bert-base-polish-
uncased-v1/snapshots/62be9821055981deafb23f217b68cc41f38cdb76/config.json
Model config BertConfig {
  "_name_or_path": "dkleczek/bert-base-polish-uncased-v1",
  "architectures": [
    "BertForMaskedLM",
    "BertForPreTraining"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
```

```
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.25.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 60000
}

Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.
No `TrainingArguments` passed, using `output_dir=tmp_trainer`.
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5
(from all installed integrations to none). In v5, you will need to use
`--report_to all` to get the same behavior as now. You should start updating
your code and make this info disappear :-).
***** Running Prediction *****
  Num examples = 1000
  Batch size = 8
You're using a BertTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.

<IPython.core.display.HTML object>
```

[ ]:
```
predicted_labels = np.argmax(predictions.predictions, axis=1)
```

[ ]:
```
predicted_labels
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

[ ]:
```
predictions
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

[ ]:
```
dataset1_test['label']
```

[ ]:
```
TP, FP, TN, FN = None, None, None, None
for i in range(len(dataset1_test['label'])):
  if TP == None and (predicted_labels[i] == dataset1_test['label'][i] == 1):
    TP = dataset1_test['text'][i]
  elif FP == None and (predicted_labels[i] == 1 and dataset1_test['label'][i] !
↪= predicted_labels[i]):
    FP = dataset1_test['text'][i]
  elif TN == None and (predicted_labels[i] == dataset1_test['label'][i] == 0):
    TN = dataset1_test['text'][i]
  elif FN == None and (predicted_labels[i] == 0 and dataset1_test['label'][i] !
↪= predicted_labels[i]):
```

```
    FN = dataset1_test['text'][i]
  if (TP!= None and FP!= None and TN != None and FN != None):
    break
```

[ ]:
```
print(f'TP: {TP}\nFP:{FP}\nTN:{TN}\nFN:{FN}')
```

```
TP: @anonymized_account Dokładnie, pisdzielstwo nie ma prawa rozpierdalać
systemu,  sądownictwa nie mając większości
FP:Prowadzący mówi ze nikt mu nie wysłał szkiców projektów jak nie jak ja ci
wysłałam imbecylu
TN:@anonymized_account Spoko, jak im Duda z Morawieckim zamówią po pięć piw to
wszystko będzie ok.
FN:@anonymized_account Tej szmaty się nie komentuje
```

[ ]:
```
!pip install lime
```

[ ]:
```
from lime.lime_text import LimeTextExplainer
```

[ ]:
```
def line_explain(case, model):
    class_names = ['negative', 'positive']
    explainer = LimeTextExplainer(class_names=class_names)
    return explainer.explain_instance(case, lambda x: make_predictions(model,␣
 ↪x))
```

[ ]:
```
TP, FP, TN, FN = line_explain(TP,dkleczek), line_explain(FP,dkleczek),␣
 ↪line_explain(TN,dkleczek), line_explain(FN,dkleczek)
```

[ ]:
```
TP.as_list()
```

[ ]:
```
[('pisdzielstwo', 4.761457610300215),
 ('sądownictwa', -0.9155334900697341),
 ('Dokładnie', 0.2682979416139585),
 ('nie', 0.2596520263391427),
 ('rozpierdalać', -0.16415968842601067),
 ('systemu', -0.09225198884084469),
 ('ma', -0.08550343793313588),
 ('anonymized_account', 0.056259481067662036),
 ('mając', -0.0467418121913276),
 ('prawa', 0.04438254328448664)]
```

[ ]:
```
FP.as_list()
```

[ ]:
```
[('imbecylu', 5.957318841029351),
 ('ci', 0.31368803978449805),
 ('mu', 0.28681843296548765),
 ('Prowadzący', -0.24521424219166457),
 ('szkiców', -0.2134299913395874),
```

```
('projektów', -0.19685394828550745),
 ('wysłałam', -0.19111956329541396),
 ('nie', 0.11517597257485347),
 ('mówi', -0.07844022853691499),
 ('wysłał', 0.029852514006461866)]
```

```
[ ]: TN.as_list()
```

```
[ ]: [('Morawieckim', 0.5687288732813915),
 ('ok', -0.4007464393300194),
 ('zamówią', -0.3315767799765059),
 ('im', 0.27285676101939016),
 ('piw', -0.2198110735148882),
 ('Spoko', -0.20702717757499584),
 ('będzie', -0.1823719198751095),
 ('anonymized_account', 0.18025344420893719),
 ('pięć', -0.17849411202776405),
 ('jak', 0.17105732183289163)]
```

```
[ ]: FN.as_list()
```

```
[ ]: [('szmaty', 2.3277266706372144),
 ('anonymized_account', 1.9470428215376496),
 ('komentuje', -1.8635513699709219),
 ('się', -0.3696219309039097),
 ('Tej', -0.21509534493446633),
 ('nie', 0.1726428629375659)]
```

1Answer the following questions:

Which of the classifiers works the best for the task 1 and the task 2.

Dla obu zbiorów Najlepiej poradził sobie Polbert co widać w tabelkach. Następnie fasttext. Zbiór danych jest jednak mocno niezbalansowany. Oznacza to, że accuracy może tutaj nie być najelpszą metryką do oecniania. Np Bert oraz Roberta ustawiając w predykcjach wszystkie komentarze jako neutralne uzyskali lepszy wynik accuracy od klasyfikatora Bayesowskiego

Did you achieve results comparable with the results of PolEval Task?

Dla zbioru 1 accuracy bardzo podobne oraz f1 większe. Dla 2 zbioru F1 było niższe

Did you achieve results comparable with the Klej leaderboard?

Strona nie działa

Describe strengths and weaknesses of each of the compared algorithms.

Najlepsze wyniki osiąga transformer Polbert jednak jego minusem jest znaczący czas trenowania modelu. Gorsze wyniki osiąga fasttext oraz klasyfikator Bayesowski jednak czas potrzebny na ich uczenie jest znacznie mniejszy. Podsumowując jeśli zależy nam jedynie na wyniku a nie na czasie oraz zasobach należy użyć transformerów. Dwóch pozostałych modeli można używać gdy nie mamy zasobów do uczenia lub czasu i nie zależy nam na jak najelpszym wyniku.

Do you think comparison of raw performance values on a single task is enough to assess the value of a given algorithm/model?

Nie, ponieważ dany model może być dostosowany lepiej do konkretnych zbiorów danych lub jak w tym przypadku zbiór może być niezbalansowany i modele które zawsze obstawiają jedną klasę(Bert, Roberta) będą uzyskiwać lepsze accuracy od innych.

Did LIME show that the models use valuable features/words when performing their decision?

Tak, słowa które mogą być używane w nękaniu mają wysoką wartość jak np. 'pisdzielstwo', 'imbecylu', 'szmaty' Część z nich zależy od kontekstu jednak w komentarzach zazwyczaj są obelgami