```
!pip install datasets
```

Показать скрытые выходные данные

```
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

from datasets import load_dataset
from transformers import pipeline
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Formality detection approaches comparison

Formality detection can be treated as both a binary classification task and a regression problem. In this work, I approach this task as a binary classification with formal and informal classes to better align with real-world use cases, where clear distinctions in formality assessment are often required.

Evaluation Metrics

I will evaluate model performance using standard binary classification metrics: Accuracy, Precision, Recall and F1-Score. To address potential class imbalance and detect model bias, I will compare macro-averaged and weighted variants of these metrics.

Datasets

I'm using the Pavlick and Tetreault (2016) formality scores dataset. Unlike larger datasets like GYAFC and XFORMAL, which require special permission to access, this dataset is freely available and allows straightforward testing of formality detection models without time-consuming approval steps. I chose this dataset to prioritize time efficiency to streamline the evaluation process.

Key characteristics of this dataset:

- 9,270 sentences for training and 2,000 for testing
- Formality scores from -3 (very informal) to +3 (very formal)
- Text types: Answers, news articles, blog posts and emails
- Human-labeled scores reflecting human perception of text formality

Data Preprocessing

To adapt this dataset for binary classification, I will add a `label` field that maps formality scores to binary classes:

- **1 (Formal)**
- **0 (Informal)**

```
dataset = load_dataset('osyvokon/pavlick-formality-scores', split='test')
dataset = dataset.to_pandas()
dataset['label'] = (dataset['avg_score'] >= 0).astype(int)
```

```
dataset.head()
```

	domain	avg_score	sentence	label
0	news	1.00	Saleh said the detainees told interrogators th...	1
1	answers	-2.25	i own this board, now.	0
2	answers	-2.00	will lead you into blind alleys.	0
3	email	2.20	If you have any questions or wish to speak fur...	1
4	answers	0.60	However, your case may be different.	1

```
dataset.groupby('domain')['sentence'].count()
```

```
         sentence
 domain
 answers      894
    blog      345
   email      273
    news      488

dtype: int64
```

```
dataset.groupby('label')['sentence'].count()
```

```
       sentence
 label
     0      876
     1     1124

dtype: int64
```

```
dataset.groupby(['domain', 'label'])['sentence'].count()
```

```
                  sentence
 domain  label
 answers     0         595
             1         299
    blog     0         112
             1         233
   email     0          85
             1         188
    news     0          84
             1         404

dtype: int64
```

```
texts = dataset["sentence"].to_list()
true_labels = dataset["label"].to_list()
```

Comparison of Approaches

I evaluated two distinct approaches for formality detection:

1. Simple Baseline Approach

- Bag-of-words with Logistic Regression: a traditional machine learning method using word frequency features.

2. Transformer-Based Approaches
I tested two pre-trained transformer models:

1. `s-nlp/deberta-large-formality-ranker`
2. `s-nlp/mdistilbert-base-formality-ranker`

1. Bag-of-words with Logistic regression

No pre-trained models using the bag-of-words method exist for formality detection. So, I trained a logistic regression model from scratch using the Pavlick dataset.

First, I will load the Pavlick dataset's training data and preprocess it:

```
train_dataset = load_dataset('osyvokon/pavlick-formality-scores', split='train')
train_dataset = train_dataset.to_pandas()
train_dataset['label'] = (train_dataset['avg_score'] >= 0).astype(int)

train_texts = train_dataset["sentence"]
train_true_labels = train_dataset["label"]
```

Then, I will extract features for the bag-of-words approach:

```
vectorizer = CountVectorizer(
    max_features=1000,
    stop_words='english',
    ngram_range=(1, 2)
)
X_train = vectorizer.fit_transform(train_texts).toarray()
```

And then I will train a Logistic Regression model on them:

```
model = LogisticRegression(class_weight="balanced")
model.fit(X_train, train_true_labels)
```

Показать скрытые выходные данные

Then I will make predictions on the test data:

```
X_test = vectorizer.transform(texts).toarray()
pred_binary = model.predict(X_test)
```

And evaluate the results:

```
print("Classification Report:")
print(classification_report(true_labels, pred_binary))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.80      0.71       876
           1       0.80      0.64      0.71      1124

    accuracy                           0.71      2000
   macro avg       0.72      0.72      0.71      2000
weighted avg       0.73      0.71      0.71      2000
```

We see a good F1-score for both classes (0.71), and also a quite decent accuracy of 0.71.

Also, we can see that this model has a high recall for the informal class (0.80), so it can be useful for applications where it's important to catch all informal sentences.

Despite the test data being imbalanced, looking at the macro-averaged F1-score and the weighted average F1-score, we can see that the model is not biased.

2. deberta-large-formality-ranker

Next, I want to test a transformer-based approach, "deberta-large-formality-ranker". This model was pre-trained on a formal detection dataset and is publicly available on Hugging Face, which is why I can use it directly.

```
pipe = pipeline("text-classification", model="s-nlp/deberta-large-formality-ranker")
predictions = pipe(texts)

pred_labels = [1 if pred["label"] == 'LABEL_0' else 0 for pred in predictions]
```

```
print("Test Results:")
print(classification_report(true_labels, pred_labels))
```

```
config.json: 100%                    790/790 [00:00<00:00, 91.1kB/s]
model.safetensors: 100%              1.62G/1.62G [00:06<00:00, 237MB/s]
tokenizer_config.json: 100%          1.40k/1.40k [00:00<00:00, 184kB/s]
vocab.json: 100%                     798k/798k [00:00<00:00, 14.2MB/s]
merges.txt: 100%                     456k/456k [00:00<00:00, 42.2MB/s]
tokenizer.json: 100%                 2.11M/2.11M [00:00<00:00, 41.9MB/s]
special_tokens_map.json: 100%        963/963 [00:00<00:00, 132kB/s]
Device set to use cuda:0
Test Results:
              precision    recall  f1-score   support

           0       0.87      0.48      0.62       876
           1       0.70      0.94      0.80      1124

    accuracy                           0.74      2000
   macro avg       0.78      0.71      0.71      2000
weighted avg       0.77      0.74      0.72      2000
```

We see a good F1-score for the formal class (0.80), and also a best accuracy of 0.74.

Also, we can see that this model has a high recall for the formal class (0.94), so it can be useful for applications where it's important to catch all formal sentences.

Despite the test data being imbalanced, looking at the macro-averaged F1-score and the weighted average F1-score, we can see that the model is not biased.

3. mdistilbert-base-formality-ranker

Next, I want to test a transformer-based approach, "mdistilbert-base-formality-ranker". This model was pre-trained on a formal detection dataset and is publicly available on Hugging Face, which is why I also can use it directly.

```
# s-nlp/mdistilbert-base-formality-ranker

pipe = pipeline("text-classification", model="s-nlp/mdistilbert-base-formality-ranker")
predictions = pipe(texts)

pred_labels = [1 if pred["label"] == 'LABEL_0' else 0 for pred in predictions]

print("Test Results:")
print(classification_report(true_labels, pred_labels))
```

```
config.json: 100%                    650/650 [00:00<00:00, 82.5kB/s]
model.safetensors: 100%              541M/541M [00:02<00:00, 266MB/s]
tokenizer_config.json: 100%          321/321 [00:00<00:00, 37.0kB/s]
vocab.txt: 100%                      996k/996k [00:00<00:00, 28.2MB/s]
tokenizer.json: 100%                 2.92M/2.92M [00:00<00:00, 45.8MB/s]
special_tokens_map.json: 100%        125/125 [00:00<00:00, 17.2kB/s]
Device set to use cuda:0
Test Results:
              precision    recall  f1-score   support

           0       0.87      0.40      0.55       876
           1       0.67      0.95      0.79      1124

    accuracy                           0.71      2000
   macro avg       0.77      0.68      0.67      2000
weighted avg       0.76      0.71      0.68      2000
```

We see a good F1-score for the formal class (0.79), and also a quite decent accuracy of 0.71.

Also, we can see that this model similar to previous model has a high recall for the formal class (0.95), so it can be useful for applications where it's important to catch all formal sentences.

Despite the test data being imbalanced, looking at the macro-averaged F1-score and the weighted average F1-score, we can see that the model is not biased.

Results:

Transformer models outperformed the bag-of-words baseline in formality detection, achieving higher F1-scores and accuracy. Specifically, `deberta-large-formality-ranker` excelled at identifying formal sentences, while the bag-of-words model showed a strong recall for informal content. Despite data imbalance, all models demonstrated minimal bias, indicating robust performance across both classes.