Natural Language Processing

Ilya Shymialevich

21.01.2025

1 Preproccesing

1.1 Wprowadzenie

Kod ten analizuje zbiór danych dotyczacy klauzul abuzynnych w jezyku polskim, zawierajacy trzy podzbiory: `train`, `validation` oraz `test`. Celem tego kodu jest wizualizacja rozkładu klas w zbiorach, ocena statystyk dotyczacych liczby słów w próbkach oraz generowanie wykresów słupkowych i statystyk słów dla każdej z klas. Zbiór danych pochodzi z repozytorium `laugustyniak/abusive-clauses-pl`.

1.2 Importowanie bibliotek

Na poczatku kod importuje niezbedne biblioteki:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from datasets import load_dataset
```

1.3 Ładowanie zbioru danych

Zbiór danych jest ładowany za pomoca funkcji `load_dataset` z biblioteki `datasets`. Zbiór danych `laugustyniak/abusive-clauses-pl` jest pobierany i przypisywany do zmiennej `ds`.

```
ds = load_dataset("laugustyniak/abusive-clauses-pl")
```

1.4 Mapa etykiet

Aby ułatwić interpretacje etykiet w zbiorze danych, tworzona jest mapa etykiet, która mapuje wartości 0 i 1 na nazwy klas:

```
label_map = {0: 'KLAUZULA_ABUZYWNA', 1: '
    BEZPIECZNE_POSTANOWIENIE_UMOWNE'}
```

1.5 Funkcja `show_class_distribution`

Funkcja ta wypisuje rozkład klas dla każdego z wybranych podzbiorów danych (np. `train`, `validation`, `test`). Etykiety 0 i 1 sa zamieniane na odpowiadajace im nazwy klas. Funkcja ta używa metody `value_counts()` z biblioteki `pandas` do zliczania liczby próbek w każdej klasie.

```
def show_class_distribution(dataset, subsets):
    for subset in subsets:
        if subset in dataset:
            data = dataset[subset].to_pandas()
            data['label'] = data['label'].map(label_map)
            print(f"Rozk ad klas w zbiorze {subset}:")
            print(data['label'].value_counts(), "\n")
```

1.6 Funkcja `plot_class_distribution`

Funkcja ta generuje wykresy słupkowe, które przedstawiaja rozkład klas w zbiorach `train`, `validation` i `test`. Dodatkowo, funkcja oblicza średnia liczbe słów w każdej z klas i generuje wykres porównujacy te średnie. Wykresy sa tworzone za pomoca `seaborn` i `matplotlib`.

```
def plot_class_distribution(dataset, subsets):
    for subset in subsets:
        if subset in dataset:
            data = dataset[subset].to_pandas()
            data['label'] = data['label'].map(label_map)

            # Tworzenie wykresu dla rozk adu klas
            plt.figure(figsize=(12, 6))
            plt.subplot(1, 2, 1)
            sns.countplot(data=data, x='label', hue='label',
                palette='Set2', alpha=0.8)
            plt.title(f"Rozk ad klas w zbiorze {subset}")
            plt.xlabel("Klasy")
            plt.ylabel("Liczba pr bek")

            # Obliczenia dla redniej liczby s w w ka dej
                klasie
            data['word_count'] = data['text'].apply(lambda x: len(x
                .split()))
            avg_words_by_class = data.groupby('label')['word_count'
                ].mean()

            # Wykres dla redniej liczby s w
            plt.subplot(1, 2, 2)
            sns.barplot(x=avg_words_by_class.index, y=
                avg_words_by_class.values, palette='Set2')
            plt.title(f" rednia liczba s w w ka dej klasie w
                zbiorze {subset}")
            plt.xlabel("Klasa")
            plt.ylabel(" rednia liczba s w ")

            # Wy wietlenie wykresu
            plt.tight_layout()
```

```
            plt.show()
```

1.7 Funkcja show_word_statistics_and_plots

Funkcja ta oblicza ogólne statystyki słów w zbiorze, takie jak całkowita liczba słów, średnia liczba słów na próbke oraz średnia liczba słów w każdej klasie. Statystyki sa nastepnie wypisywane, a wykresy słupkowe pokazujace średnia liczbe słów w każdej klasie sa generowane.

```
def show_word_statistics_and_plots(dataset, subsets):
    for subset in subsets:
        if subset in dataset:
            data = dataset[subset].to_pandas()
            data['label'] = data['label'].map(label_map)

            # Policz liczb  s   w w ka dej pr bce
            data['word_count'] = data['text'].apply(lambda x: len(x
                .split()))

            # Og lne statystyki
            total_words = data['word_count'].sum()  # Ca kowita
                liczba s   w
            avg_words_per_sample = data['word_count'].mean()  #
                 rednia  liczba s   w na pr bk
            avg_words_by_class = data.groupby('label')['word_count'
                ].mean()  #  rednia  liczba s   w w ka dej klasie

            print(f"Statystyki␣dla␣zbioru␣{subset}:")
            print(f"Ca kowita␣liczba␣s   w :␣{total_words}")
            print(f" rednia ␣liczba␣s   w␣na␣pr bk :␣{
                avg_words_per_sample:.2f}")
            print(f" rednia ␣liczba␣s   w␣dla␣ka dej␣klasy:")
            print(avg_words_by_class, "\n")
```

1.8 Funkcja czyszczenia tekstu

Na końcu, teksty w zbiorze beda czyszczone za pomoca funkcji clean_text. Funkcja ta przekształca tekst na małe litery, usuwa interpunkcje, cyfry oraz wielokrotne spacje. Celem tej operacji jest poprawienie jakości danych wejściowych przed dalsza analiza lub treningiem modelu.

```
# Funkcja do czyszczenia tekstu
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^\w\s]", "", text)  # Usuni cie interpunkcji
    text = re.sub(r"\d+", "", text)      # Usuni cie cyfr
    text = re.sub(r"\s+", "␣", text)     # Usuni cie wielokrotnych
        spacji
    return text.strip()
```

1.9 Podsumonie rozdziału I

W tym rozdziale przedstawione zostana operacje wstepnego przetwarzania tekstu, które maja na celu przygotowanie danych wejściowych do dalszej analizy. Preprocessing obejmuje takie działania jak czyszczenie tekstu, tokenizacja, usuwanie zbednych znaków, itp.

```
Rozkład klas w zbiorze train:
label
BEZPIECZNE_POSTANOWIENIE_UMOWNE    2338
KLAUZULA_ABUZYWNA                  1946
Name: count, dtype: int64

Rozkład klas w zbiorze validation:
label
KLAUZULA_ABUZYWNA                  1063
BEZPIECZNE_POSTANOWIENIE_UMOWNE     456
Name: count, dtype: int64

Rozkład klas w zbiorze test:
label
BEZPIECZNE_POSTANOWIENIE_UMOWNE    2333
KLAUZULA_ABUZYWNA                  1120
Name: count, dtype: int64
```

Figure 1: Rozkład klass

```
Statystyki dla zbioru train:
Całkowita liczba słów: 105206
Średnia liczba słów na próbkę: 24.56
Średnia liczba słów dla każdej klasy:
label
BEZPIECZNE_POSTANOWIENIE_UMOWNE    21.942686
KLAUZULA_ABUZYWNA                  27.699897
Name: word_count, dtype: float64

Statystyki dla zbioru validation:
Całkowita liczba słów: 51675
Średnia liczba słów na próbkę: 34.02
Średnia liczba słów dla każdej klasy:
label
BEZPIECZNE_POSTANOWIENIE_UMOWNE    37.767544
KLAUZULA_ABUZYWNA                  32.411101
Name: word_count, dtype: float64

Statystyki dla zbioru test:
Całkowita liczba słów: 84373
Średnia liczba słów na próbkę: 24.43
Średnia liczba słów dla każdej klasy:
label
BEZPIECZNE_POSTANOWIENIE_UMOWNE    21.763823
KLAUZULA_ABUZYWNA                  29.998214
Name: word_count, dtype: float64
```

Figure 2: Rozkład średnich

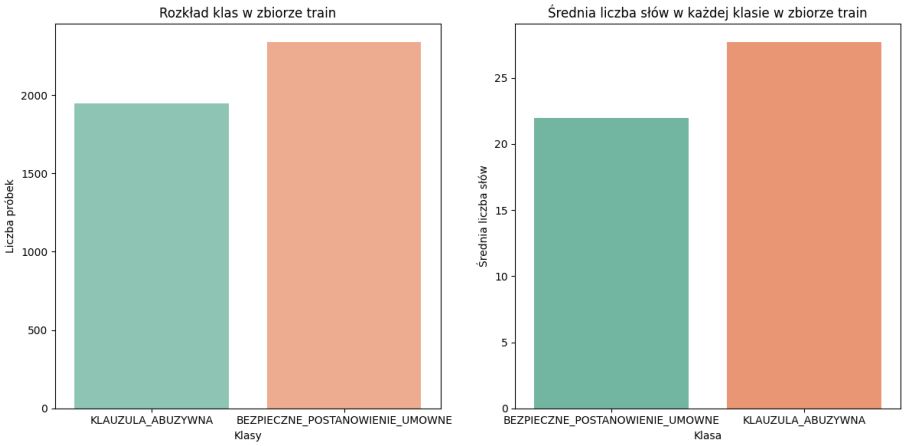

Figure 3: Rozkład dla kategorii train

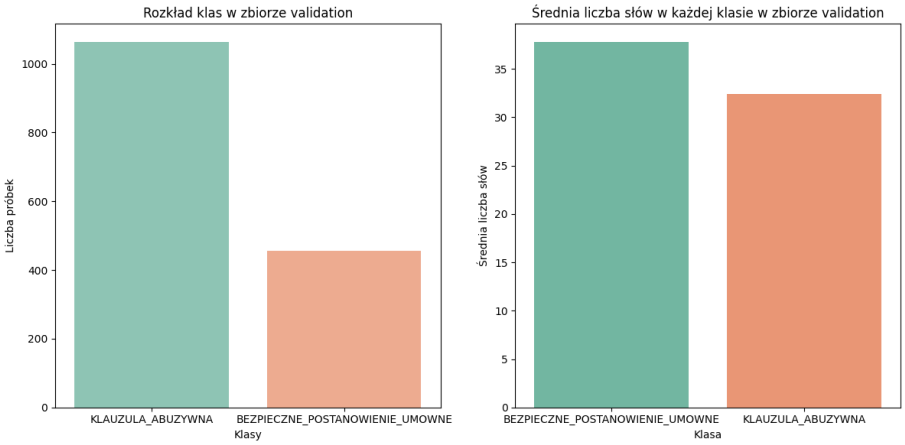

Figure 4: Rozkład dla kategorii validate

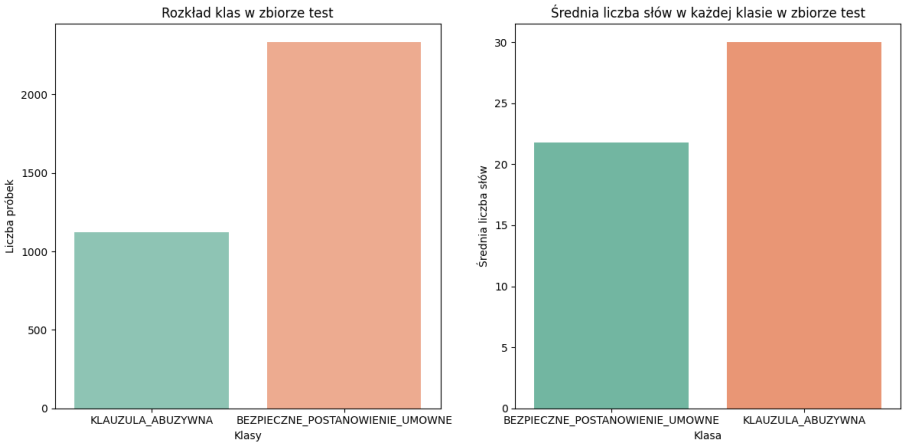

Figure 5: Rozkład dla kategorii test

2 Model treningowy i optymalizacja hiperparametrów

W tej sekcji kodu, model oparty na architekturze BERT jest trenowany do klasyfikacji klauzul abuzynnych w jezyku polskim. Zastosowane sa różne techniki, w tym optymalizacja hiperparametrów, użycie tokenizerów oraz generowanie wykresów oceny modelu.

2.1 Importowanie niezbednych bibliotek

Kod importuje nastepujace biblioteki:

- `PyTorch` do tworzenia i trenowania modelu, w tym `torch.nn` i `torch.optim` do budowania sieci i optymalizacji.
- `Transformers` z biblioteki Hugging Face, która umożliwia użycie pretrenowanych modeli, takich jak BERT i TinyBERT, do zadań klasyfikacyjnych.
- `Scikit-learn` do obliczania miar oceny, takich jak F1-score, oraz do generowania macierzy konfuzji i wykresu ROC.
- `Matplotlib` i `Seaborn` do wizualizacji wyników.

2.2 Ładowanie danych

Zbiór danych jest ładowany z repozytorium `laugustyniak/abusive-clauses-pl` przy użyciu funkcji `load_dataset` z biblioteki `datasets`. Zawiera on dane podzielone na zbiory treningowy, walidacyjny i testowy.

```
dataset = load_dataset('laugustyniak/abusive-clauses-pl')
```

2.3 Model i tokenizer

Model BERT dla klasyfikacji sekwencji jest ładowany z pretrenowanego modelu `huawei-noah/TinyBERT_General_4L_312D`. Tokenizer BERT jest również załadowany, aby przekształcić teksty w odpowiedni format wejściowy.

```
model_name = 'huawei-noah/TinyBERT_General_4L_312D'
model = BertForSequenceClassification.from_pretrained(model_name,
    num_labels=2)
tokenizer = BertTokenizer.from_pretrained(model_name)
```

2.4 Funkcja czyszczenia tekstu

Teksty w zbiorze danych sa czyszczone za pomoca funkcji `clean_text`, która konwertuje tekst na małe litery, usuwa interpunkcje, cyfry oraz nadmiarowe spacje. Bedzie zrobiony test, używajac funkcji do czyszczenia oraz bez funkcji czyszczenia i pokazane wyniki

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^\w\s]", "", text)  # Usuni cie interpunkcji
    text = re.sub(r"\d+", "", text)      # Usuni cie cyfr
    text = re.sub(r"\s+", "␣", text)     # Usuni cie wielokrotnych
        spacji
    return text.strip()
```

2.5 Funkcja tokenizacji

Funkcja `tokenize_function` wykonuje tokenizacje tekstu, a także przetwarza teksty za pomoca funkcji czyszczenia. W wyniku działania tej funkcji, teksty sa konwertowane na tokeny, które moga zostać wykorzystane przez model.

```
def tokenize_function(examples):
    examples['text'] = [clean_text(text) for text in examples['text
        ']]
    return tokenizer(examples['text'], padding="max_length",
        truncation=True, max_length=128)
```

2.6 Tokenizacja danych

Dane sa tokenizowane, a nastepnie zapisane w zmiennej `tokenized_datasets`. Tokenizacja jest wykonywana na całym zbiorze danych za pomoca funkcji `map`.

```
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

2.7 Urzadzenie do obliczeń

Model jest uruchamiany na odpowiednim urzadzeniu (GPU lub CPU), zależnie od dostepności. Jeśli dostepne jest GPU, model zostanie przeniesiony na to urzadzenie.

```
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
print(f"Using␣device:␣{device}")
```

2.8 Funkcja do obliczania metryk

Funkcja `compute_metrics` oblicza metryki na podstawie wyników modelu, w tym F1-score, który jest używany jako główna miara jakości modelu.

```
def compute_metrics(p):
    preds, labels = p
    preds = preds.argmax(axis=1)
    return {'f1': f1_score(labels, preds, average='weighted')}
```

2.9 Hiperparametry do przeszukania

Hiperparametry modelu sa definiowane w słowniku `param_grid`. Sa to: szybkość uczenia (`learning_rate`), rozmiar partii treningowej (`per_device_train_batch_size`) oraz liczba kroków akumulacji gradientu (`gradient_accumulation_steps`).

```
param_grid = {
    'learning_rate': [1e-5,5e-5,1e-4],
    'per_device_train_batch_size': [16,24,32],
    'gradient_accumulation_steps': [2,4]
}
```

2.10 Optymalizacja hiperparametrów

Funkcja `train_with_params` trenuje model z różnymi kombinacjami hiperparametrów. Różne kombinacje sa generowane za pomoca `ParameterGrid`, a wynik F1-score dla każdej kombinacji jest zapisywany.

```
def train_with_params(params):
    print(f"\nTrening z parametrami: {params}")
    training_args = TrainingArguments(
        output_dir=f"./results_lr{params['learning_rate']}_bs{
            params['per_device_train_batch_size']}_ga{params['
            gradient_accumulation_steps']}",
        eval_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
        per_device_train_batch_size=params['
            per_device_train_batch_size'],
        per_device_eval_batch_size=24,
        num_train_epochs=1,
        learning_rate=params['learning_rate'],
        weight_decay=0.01,
        logging_dir="./logs",
        logging_steps=50,
        metric_for_best_model="f1",
        greater_is_better=True,
        fp16=True,
        gradient_accumulation_steps=params['
            gradient_accumulation_steps'],
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_datasets['train'],
        eval_dataset=tokenized_datasets['validation'],
        compute_metrics=compute_metrics
    )

    trainer.train()
    eval_results = trainer.evaluate(tokenized_datasets['validation'
        ])
    f1_score = eval_results['eval_f1']
    print(f"F1 score dla parametr w {params}: {f1_score:.4f}")
```

```
    return f1_score, params
```

2.11 Wyszukiwanie najlepszych hiperparametrów

Funkcja ta wykonuje przeszukiwanie przestrzeni hiperparametrów. Dla każdej kombinacji hiperparametrów, model jest trenowany, a wynik F1-score jest zapisany. Najlepsze wartości hiperparametrów sa zapisywane do późniejszego użycia.

```
best_f1 = 0
best_params = None
results = []
for params in grid:
    f1, tested_params = train_with_params(params)
    results.append((f1, tested_params))
    if f1 > best_f1:
        best_f1 = f1
        best_params = tested_params

print(f"\nNajlepszy F1: {best_f1:.4f} uzyskany dla parametr w: {
    best_params}")
```

2.12 Testowanie na najlepszych hiperparametrach

Po znalezieniu najlepszych hiperparametrów, model jest trenowany ponownie na pełnym zbiorze treningowym, a nastepnie testowany na zbiorze testowym, aby ocenić jego ostateczna wydajność.

```
trainer.train()
test_results = trainer.evaluate(tokenized_datasets['test'])
print("Test results with best params:", test_results)
```

2.13 Macierz konfuzji i wykres ROC

Na końcu, generowana jest macierz konfuzji oraz wykres ROC, aby wizualnie ocenić działanie modelu na zbiorze testowym. Wyniki sa zapisywane do plików CSV.

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Illegal', 'legal'], yticklabels=['Illegal', 'legal'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

2.14 Sprawdzenie funkcji czyszczenia

F1	Params
0.455	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 16}
0.474	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 24}
0.489	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 32}
0.612	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 16}
0.613	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 24}
0.611	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 32}
0.640	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 16}
0.641	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 24}
0.652	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 32}
0.651	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 16}
0.651	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 24}
0.650	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 32}
0.658	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 16}
0.651	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 24}
0.655	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 32}
0.647	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 16}
0.653	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 24}
0.650	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 32}

Table 1: Tabela z wynikami F1 i parametrami (zaokraglone do 3 miejsc po przecinku) z funkcja clean text

F1	Params
0.573	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 16}
0.575	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 24}
0.581	{'gradient_accumulation_steps': 2, 'learning_rate': 1e-05, 'per_device_train_batch_size': 32}
0.595	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 16}
0.603	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 24}
0.608	{'gradient_accumulation_steps': 2, 'learning_rate': 5e-05, 'per_device_train_batch_size': 32}
0.627	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 16}
0.632	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 24}
0.633	{'gradient_accumulation_steps': 2, 'learning_rate': 0.0001, 'per_device_train_batch_size': 32}
0.633	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 16}
0.637	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 24}
0.635	{'gradient_accumulation_steps': 4, 'learning_rate': 1e-05, 'per_device_train_batch_size': 32}
0.636	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 16}
0.644	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 24}
0.648	{'gradient_accumulation_steps': 4, 'learning_rate': 5e-05, 'per_device_train_batch_size': 32}
0.632	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 16}
0.648	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 24}
0.650	{'gradient_accumulation_steps': 4, 'learning_rate': 0.0001, 'per_device_train_batch_size': 32}

Table 2: Tabela z wynikami F1 i parametrami (zaokraglone do 3 miejsc po przecinku) bez funkcji clean text

- Testy na danych walidacyjnych, a model był trenowany przez jedna epoke.
- Testy zostały przeprowadzone na danych walidacyjnych, co pozwala na ocene wydajności modelu w kontekście generalizacji.
- Maksymalne wartości F1 w obu tabelach osiagaja 0.658 (Tabela 1) i 0.650 (Tabela 2),.
- Parametry `learning_rate` o wartościach 0.0001 i `per_device_train_batch_size` o rozmiarze 32 sa optymalne, co prowadzi do najwyższych wyników F1.
- Zmienność wyników sugeruje, że odpowiednia optymalizacja tych parametrów znaczaco wpływa na uzyskiwana jakość modelu.

Najlespsze parametry użyjemy do przyszłych obliczeń,iż obliczenia były przeprowadzone dla 1 epochy, z powodu dużych kosztów obliczeniowych, nastepnie bedzie zmieniana liczba epoch.

2.15 Wyniki F1 Score dla różnych epok dla danych testowych

Dane przedstawiaja wyniki F1 Score dla modelu po różnych liczbach epok treningowych (1, 5, 10).Widać, że model poprawia swoje osiagi w miare upływu czasu treningu. Wzrost F1 Score z 0.7857 do 0.8410 wskazuje na poprawe jakości modelu w miare dalszego dopasowywania go do danych. Poniżej przedstawiam analize tych wyników:

Epoka	**F1 Score**
1	0.7857
5	0.8287
10	0.8410

Table 3: Wyniki F1 Score po różnych epokach treningu.

Choć model znaczaco poprawił swoje wyniki w pierwszych 5 epokach (z około 0.7857 do 0.8287), to w kolejnych 5 epokach (z 5 do 10) poprawa była mniejsza (z 0.8287 do 0.8410). Może to sugerować, że model zbliża sie do swojego maksimum i dalszy trening nie przynosi już tak dużych korzyści.

Macierz konfuzji to narzedzie oceny wydajności modelu klasyfikacyjnego, które porównuje rzeczywiste etykiety z przewidywaniami modelu. Zawiera cztery podstawowe elementy: True Positive (TP), True Negative (TN), False Positive (FP) oraz False Negative (FN), które pozwalaja na analize błedów klasyfikacji i skuteczności modelu.

Wykres ROC (Receiver Operating Characteristic) przedstawia zależność miedzy True Positive Rate (TPR) a False Positive Rate (FPR), pokazujac, jak zmienia sie skuteczność modelu przy różnych prógach decyzyjnych. AUC (Area Under the Curve) mierzy ogólna jakość modelu – im wyższe AUC, tym lepsza wydajność.

Wyniki macierzy konfuzji oraz wykresów ROC pozwalaja na dokładna ocene i optymalizacje modelu, szczególnie w przypadku nierównych zbiorów danych.

Macierz konfuzji

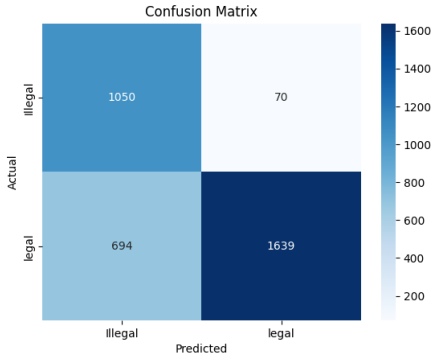

Figure 6: Macierz konfuzji - Epoka 1

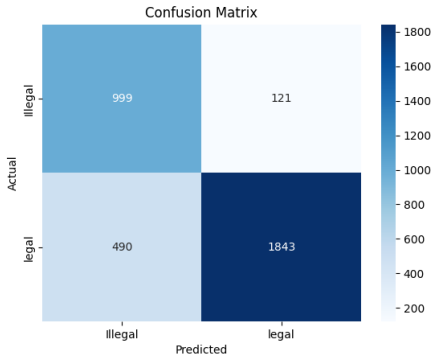

Figure 7: Macierz konfuzji - Epoka 5

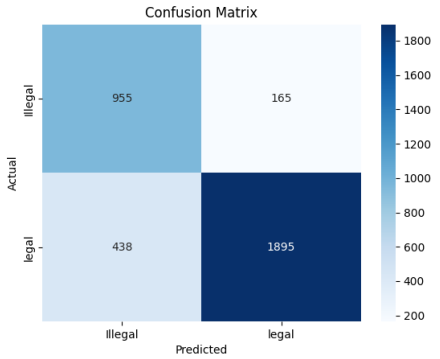

Figure 8: Macierz konfuzji - Epoka 10

Wykresy ROC

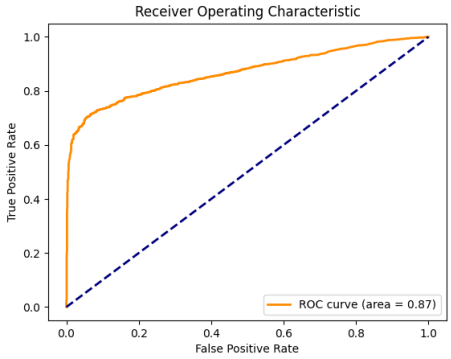

Figure 9: Macierz konfuzji - Epoka 1

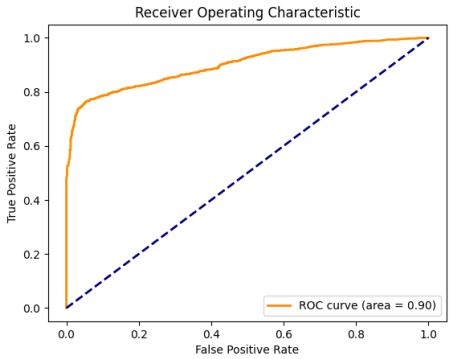

Figure 10: Macierz konfuzji - Epoka 5

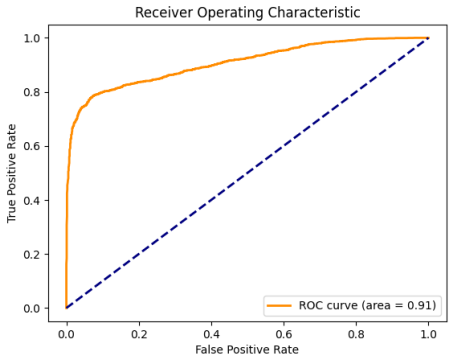

Figure 11: Macierz konfuzji - Epoka 10

2.16 Wnioski

Analiza wyników macierzy konfuzji wskazuje na nastepujace zależności:

- **Poprawa wraz z epokami**: Wartość AUC rośnie wraz z liczba epok, co wskazuje na lepsze dostosowanie modelu do danych i poprawe zdolności rozróżniania klas.
- **Stabilizacja po 5 epokach**: Wzrost AUC pomiedzy 5 a 10 epokami jest minimalny (zaledwie 0.01), co sugeruje, że model osiaga punkt nasycenia, a dalszy trening nie przynosi znaczacych korzyści.
- **Ogólna skuteczność**: AUC na poziomie 0.91 po 10 epokach świadczy o bardzo dobrej skuteczności modelu w separacji klas, co oznacza, że rzadko myli klasy w sposób systematyczny
- Liczba poprawnych klasyfikacji dla klasy **Illegal-Illegal (True Positives)** zmniejsza sie wraz z liczba epok. Wyniki te moga wynikać z bardziej konserwatywnego podejścia modelu, który stara sie minimalizować błedne oznaczenia klasy *Legal* jako *Illegal*.
- Liczba poprawnych klasyfikacji dla klasy **Legal-Legal (True Negatives)** rośnie wraz z liczba epok. Model zyskuje wieksza pewność w rozpoznawaniu tej klasy, co prowadzi do zmniejszenia liczby błednych klasyfikacji *Illegal* jako *Legal*.
- Wartości w macierzy stabilizuja sie po 5 epokach, co może sugerować, że dalszy trening (np. do 10 epok) nie przynosi znaczacej poprawy w klasyfikacji.