Raport Badawczy - Analiza Emocji W Recenzjach Filmowych

---

1. Wstęp

1.1 Cel Projektu

Celem projektu jest zbadanie, który model inteligencji obliczeniowej jest najbardziej wydajny w zadaniu oceny emocji na podstawie tekstowych recenzji filmów. Analizie poddane zostaną różne podejścia do klasyfikacji sentymentu, aby określić, które z nich najlepiej sprawdzają się w praktyce.

1.2 Kontekst i znaczenia analizy emocji

Analiza emocji to jedno z kluczowych zagadnień w dziedzinie przetwarzania języka naturalnego. Jej celem jest automatyczne rozpoznawanie i klasyfikacja treści zawartych w tekstach, takich jak opinie, recenzje czy komentarze. W dobie mediów społecznościowych, forów internetowych oraz platform takich jak IMDb czy Rotten Tomatoes, użytkownicy generują ogromne ilości danych zawierających emocje i opinie.

Zastosowanie analizy emocji znajduje szerokie zastosowanie w marketingu, badaniach rynku, analizie reputacji marek, a także w systemach rekomendacyjnych. W kontekście recenzji filmowych, klasyfikacja sentymentu pozwala na automatyczne ocenianie odbioru filmu przez widzów, co może wspierać zarówno użytkowników, jak i twórców treści w podejmowaniu decyzji.

Rozwój modeli sztucznej inteligencji, w tym uczenia maszynowego i głębokiego uczenia, umożliwia coraz skuteczniejsze rozwiązywanie tego typu problemów. Celem niniejszego projektu jest sprawdzenie, które z dostępnych modeli najskuteczniej radzą sobie z analizą emocji w kontekście recenzji filmowych.

1.3 Zakres pracy i pytania badawcze

Zakres projektu obejmuje pełny cykl badawczy, począwszy od pozyskania i przygotowanie danych, przez eksploracji i implementacji różnym modeli kwalifikujących, kończąc na ocenie ich skuteczności i porównania wyników. Projekt skupia się na binarnej kwalifikacji danych (postivie vs negative) w recenzjach filmowych.

W ramach pracy sformułowano następujące pytania badawcze:

- Które modele inteligencji obliczeniowej osiągają najlepsze wyniki w zadaniu analizy emocji?

- Czy nowoczesne modele głębokiego uczenia, takie jak BERT, przewyższają klasyczne podejścia (np. Naive Bayes, SVM)?

- Jakie znaczenie mają metody reprezentacji tekstu (np. TF-IDF vs. embeddingi kontekstowe) dla skuteczności klasyfikatorów?

Odpowiedzi na powyższe pytania zostaną udzielone w oparciu o wyniki eksperymentów przeprowadzonych na publicznie dostępnym zbiorze danych z recenzjami filmów.

---

2. Dane

2.1 Źródła danych

Do realizacji projektu wykorzystałem zbiór danych **IMD Large Movie Review Dataset**, opracowany przez **Andrew Massa i współpracowników** z Uniwersytetu Stanforda. Zbiór ten jest szeroko stosowany w badaniach nad analizą sentymentu i stanowi standardowy benchmark dla porównywania modeli klasyfikacji emocji.

Zbiór zawiera 50 tyś recenzji filmowych podzielone równo na dane testowe i treningowe (po 25 tyś). Każda recenzja została oznaczona jako pozytywna lub negatywna, co umożliwia binarną klasyfikację sentymentu. Zbiór danych został zbalansowany, co oznacza, że liczba przykładów z każdej klasy jest równa, co eliminuje problem niezbalansowanych danych przy trenowaniu modeli.

2.2 Struktura i zawartość zbioru

Zbiór danych IMDb zawiera 50 tyś recenzji filmowych, podzielony na część treningową oraz testową, po 25 tyś każda. Na dodatek, każda z tych części zawiera odpowiednio 12,5 tyś recenzji pozytywnych oraz tyle samo negatywnych. Co zapewnia pełną równowagę klas.

Dane są udostępnione w postaci plików .txt, zorganizowanych w strukturze folderów:

---

aclIMDb/

|---- train/

| |---- pos/ <- recenzje pozytywne (w jednym pliku txt jest jedna recenzja)

| |---- neg/ <- recenzje negatywne

|---- test/

| |---- pos/

| |---- neg/

Stworzyłem plik kompresor.py który ma za zadanie skopresować tą bazę do plików .csv. Jeden plik dla danych testowych drugi dla treningowych. Dane w tym pliku wygladają następująco:

| Recenzja | Emocja Recenzji |

gdzie:

- Recenzja - jest zapisana w cudzysłowiu
- Emocja Recenzji - jest zapisana w postaci 1 gdy opinia jest pozytywna lub 0 gdy negatywna

2.3 Analiza Kodu Kompresora

Plik kompresor.py wygląda następująco:

```python
import os
import pandas as pd

def load_imdb_split(data_dir, split):
    data = {"review": [], "sentiment": []}

    for sentiment in ["pos", "neg"]:
        dir_path = os.path.join(data_dir, split, sentiment)
        label = 1 if sentiment == "pos" else 0

        for filename in os.listdir(dir_path):
            file_path = os.path.join(dir_path, filename)
            with open(file_path, encoding="utf-8") as file:
                review = file.read().strip()
                data["review"].append(review)
                data["sentiment"].append(label)

    return pd.DataFrame(data)

data_path = "aclImdb"

# Wczytaj i zapisz dane treningowe
df_train = load_imdb_split(data_path, "train")
df_train.to_csv("imdb_train.csv", index=False, encoding="utf-8")
print("Zapisano imdb_train.csv:", df_train.shape)

# Wczytaj i zapisz dane testowe
df_test = load_imdb_split(data_path, "test")
df_test.to_csv("imdb_test.csv", index=False, encoding="utf-8")
print("Zapisano imdb_test.csv:", df_test.shape)
```

---

3. Analiza Modeli

3.1 Model 1 - Klasyfikator Naive Bayes

3.1.1 Analiza Kod

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import numpy as np

# Wczytywanie danych
train_df = pd.read_csv("imdb_train.csv")
test_df = pd.read_csv("imdb_test.csv")

X_train = train_df['review']
y_train = train_df['sentiment']
X_test = test_df['review']
y_test = test_df['sentiment']

# Funkcja trenująca i ewaluująca model
def train_and_evaluate(vectorizer, model, model_name):
    print(f"\n=== {model_name} ===")

    # Wektoryzacja
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # Trening
    model.fit(X_train_vec, y_train)

    # Predykcja
    y_pred = model.predict(X_test_vec)

    # Metryki
    acc = accuracy_score(y_test, y_pred)
    print(f"Dokładność: {acc:.4f}")

    # Macierz błędu
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negatywne",
"Pozytywne"], yticklabels=["Negatywne", "Pozytywne"])
    plt.title(f"Macierz błędu - {model_name}")
    plt.xlabel("Predykcja")
    plt.ylabel("Rzeczywistość")
    plt.tight_layout()
    plt.savefig(f"conf_matrix_{model_name}.png")
    plt.show()

    # Raport klasyfikacji
    report = classification_report(y_test, y_pred, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    report_df.to_csv(f"classification_report_{model_name}.csv")
```

```
    # Wykres F1-score
    plt.figure(figsize=(5,4))
    report_df.loc[['0', '1']]['f1-score'].plot(kind='bar')
    plt.title(f"F1-score - {model_name}")
    plt.ylabel("F1-score")
    plt.xlabel("Klasa")
    plt.xticks([0,1], ["Negatywne", "Pozytywne"], rotation=0)
    plt.tight_layout()
    plt.savefig(f"f1_score_{model_name}.png")
    plt.show()

    return {
        "model": model_name,
        "accuracy": acc,
        "f1_neg": report['0']['f1-score'],
        "f1_pos": report['1']['f1-score']
    }

# Lista modeli do porównania
models_to_test = [
    ("NB_BOW_unigram", CountVectorizer(stop_words='english'), MultinomialNB()),
    ("NB_BOW_bigram", CountVectorizer(ngram_range=(1,2), stop_words='english'),
MultinomialNB()),
    ("NB_TFIDF", TfidfVectorizer(stop_words='english'), MultinomialNB())
]

# Uruchomienie i zbieranie wyników
results = []
for name, vec, model in models_to_test:
    res = train_and_evaluate(vec, model, name)
    results.append(res)

# Tabela porównawcza
results_df = pd.DataFrame(results)
print("\n=== Porównanie modeli ===")
print(results_df)

# Zapis do CSV
results_df.to_csv("naive_bayes_model_comparison.csv", index=False)

# Wykres porównawczy dokładności
plt.figure(figsize=(6,4))
sns.barplot(data=results_df, x="model", y="accuracy", palette="viridis")
plt.title("Porównanie dokładności modeli Naive Bayes")
```

```
plt.ylabel("Dokładność")
plt.xlabel("Model")
plt.xticks(rotation=15)
plt.tight_layout()
plt.savefig("naive_bayes_accuracy_comparison.png")
plt.show()
```

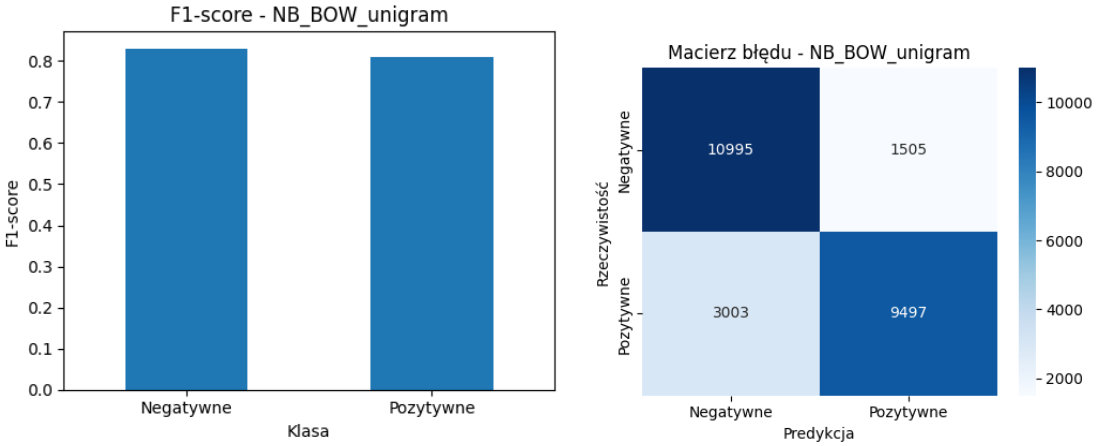
3.1.2 Analiza Wyników

Do tego modelu wykorzystałem klasyfikator Naive Bayes. I przyrównałem trzy sposoby prezentacji tekstu:

- Bag of Words (unigramy)
- Bag of Words (bigramy)
- TF-IDF

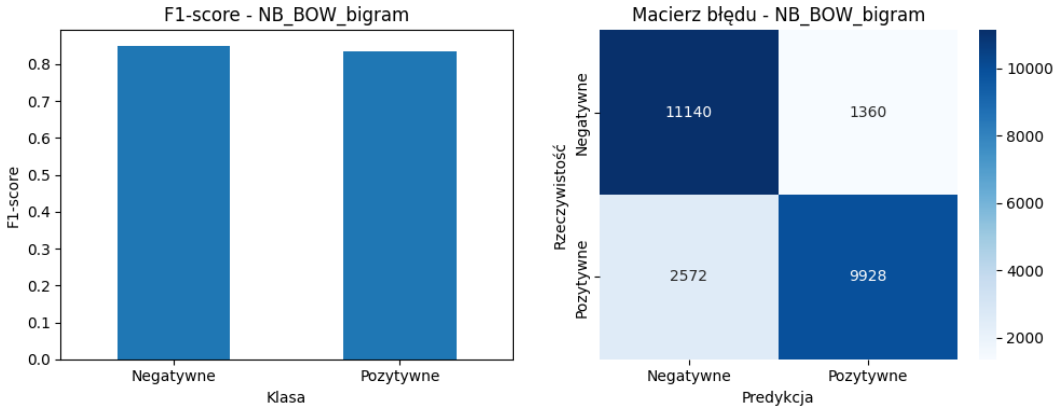
Te sposoby różnią się od siebie sposobem prezentacji tekstu. Bag of words unigram - Liczy tylko pojedyncze słowa. Bag of words bigram - Liczy pojedyncze słowa oraz pary. TF-IDF - Liczy wagę słów zależne od ich częstości występowania w tekście.

3.2.1.1 Naive Bayes Bag of Words (unigramy)



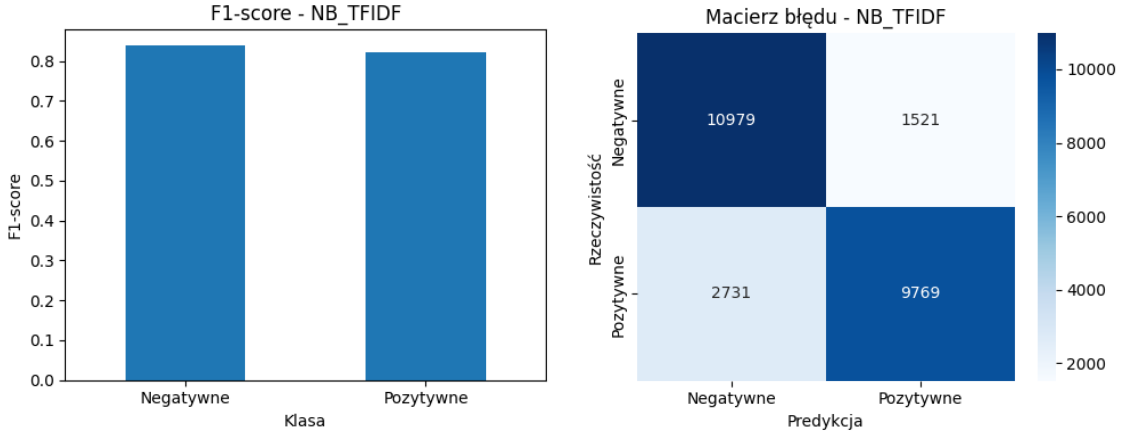
Powyżej przedstawione są dwa wykresy po lewej jest przedstawiony f1 score. Który pokazuje nam skuteczność tego modelu na poziomie 83% przy recenzjach negatywnych a 81% przy pozytywnych. Ogólna skuteczność modelu wynosi 81.97%. Po prawej mamy macierz błędów.

3.2.1.2 Naive Bayes Bag of Words (bigramy)



Powyżej mamy identyczne wykresy f1- score dla recenzji pozytywnych wyniósł 85%, a dla pozytywnych 83,5%. Ogólna skuteczność modelu wyniosła 84.27%.

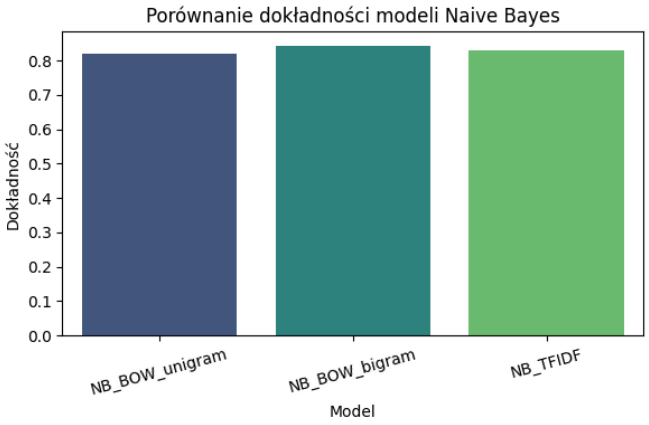
3.2.1.3 Naive Bayes TF-IDF



F1 - score dla tego modelu, dla recenzji negatywnych wyszedł 84% a dla pozytywnych 82 %. Ogólna skuteczność wyszła 82,99%.

3.2.1.4 Podsumowanie

Podsumowując najlepiej sprawdził się model Bag Of Words (bigramy). Łączna nauka tych modeli nie przekroczyła 2 minut. Co jest dobrym wynikiem procentowym patrząc na ilość czasu nauki.



3.2 Model 2 - Klasyfikator Logistic Regression z TF - IDF

3.2.1 Analiza Kodu

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re

# Czyszczenie tekstu
def preprocess_text(text):
   text = text.lower()
   text = re.sub(r'<.*?>', '', text)
   text = re.sub(r'[^a-zA-Z ]', '', text)
   return text

# Wczytanie danych
train_df = pd.read_csv("imdb_train.csv")
test_df = pd.read_csv("imdb_test.csv")

# Czyszczenie
train_df['review'] = train_df['review'].apply(preprocess_text)
test_df['review'] = test_df['review'].apply(preprocess_text)

X_train = train_df['review']
y_train = train_df['sentiment']
X_test = test_df['review']
y_test = test_df['sentiment']
```

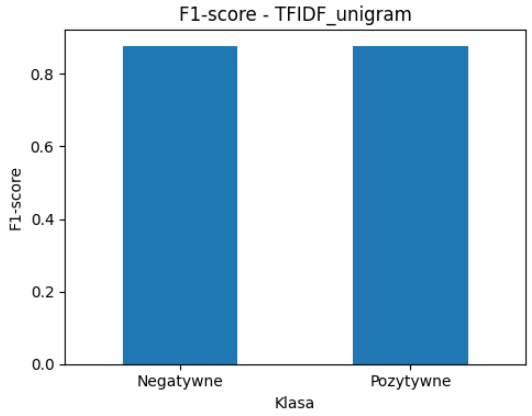
```
# Funkcja porównawcza
def train_logreg_model(vectorizer, model_name):
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_vec, y_train)

    y_pred = model.predict(X_test_vec)
    acc = accuracy_score(y_test, y_pred)

    print(f"\n=== {model_name} ===")
    print(f"Dokładność: {acc:.4f}")

    # Macierz błędu
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negatywne",
"Pozytywne"], yticklabels=["Negatywne", "Pozytywne"])
    plt.title(f"Macierz błędu - {model_name}")
    plt.xlabel("Predykcja")
    plt.ylabel("Rzeczywistość")
    plt.tight_layout()
    plt.savefig(f"conf_matrix_{model_name}.png")
    plt.show()

    # Raport
    report = classification_report(y_test, y_pred, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    report_df.to_csv(f"classification_report_{model_name}.csv")

    # Wykres F1
    plt.figure(figsize=(5,4))
    report_df.loc[['0', '1']]['f1-score'].plot(kind='bar')
    plt.title(f"F1-score - {model_name}")
    plt.ylabel("F1-score")
    plt.xlabel("Klasa")
    plt.xticks([0,1], ["Negatywne", "Pozytywne"], rotation=0)
    plt.tight_layout()
    plt.savefig(f"f1_score_{model_name}.png")
    plt.show()

    return {
        "model": model_name,
        "accuracy": acc,
```

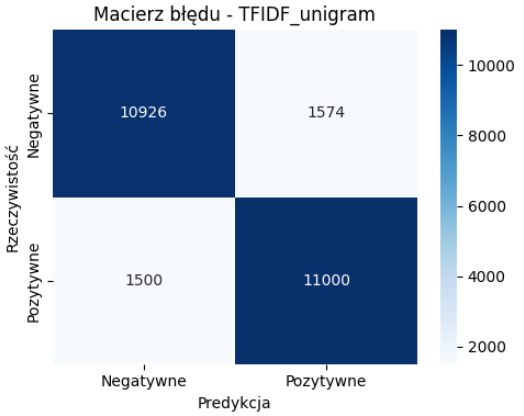
```
        "f1_neg": report['0']['f1-score'],
        "f1_pos": report['1']['f1-score']
    }

# Testowane wersje
vectorizer_configs = [
    ("TFIDF_unigram", TfidfVectorizer(stop_words='english', max_features=10000)),
    ("TFIDF_bigram", TfidfVectorizer(ngram_range=(1,2), stop_words='english',
max_features=10000)),
    ("BOW_unigram", CountVectorizer(stop_words='english', max_features=10000))
]

# Porównanie
results = []
for name, vec in vectorizer_configs:
    res = train_logreg_model(vec, name)
    results.append(res)

# Podsumowanie
results_df = pd.DataFrame(results)
print("\n=== Podsumowanie modeli Logistic Regression ===")
print(results_df)
results_df.to_csv("logreg_model_comparison.csv", index=False)

# Wykres porównawczy
plt.figure(figsize=(6,4))
sns.barplot(data=results_df, x="model", y="accuracy", palette="viridis")
plt.title("Porównanie dokładności modeli Logistic Regression")
plt.ylabel("Dokładność")
plt.xlabel("Model")
plt.xticks(rotation=15)
plt.tight_layout()
plt.savefig("logreg_accuracy_comparison.png")
plt.show()
```

3.2.2 Analiza Wyników

Kolejnym modelem jest klasyfikator Logistic Regression, jest to model liniowy, który przewiduje prawdopodobieństwo przynależności próbki do klasy. Do tego klasyfikatora porównałem znowu trzy sposoby przekształcania tekstu.

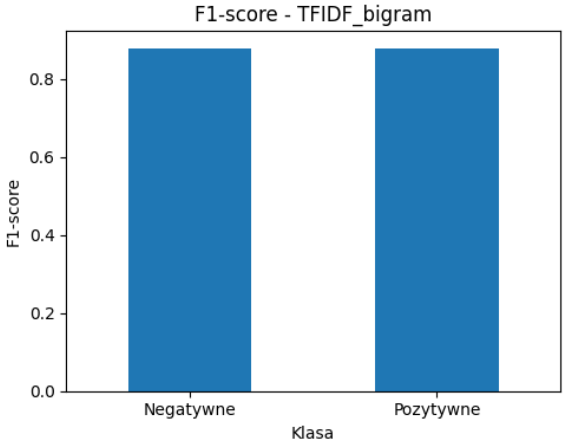
3.2.2.1 TF-IDF (unigramy)

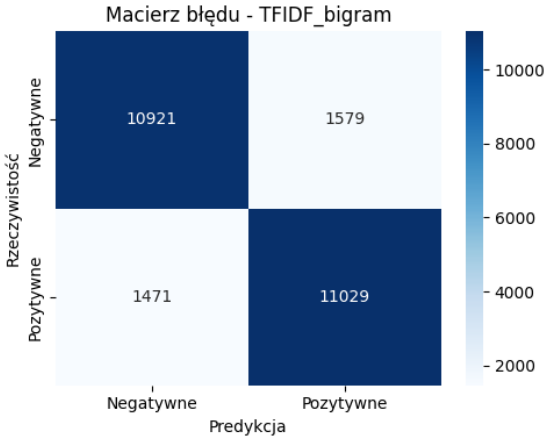




Sposób ten dzieli tekst na pojedyncze słowa. Liczy wagę na podstawie częstości występowania w tej recenzji i w całym zbiorze. Wyniki f1-score dla tego modelu dla recenzji negatywnych wyniósł 87,6% a dla pozytywnych 87,7%. Ogólny wynik wyszedł 87,77%.

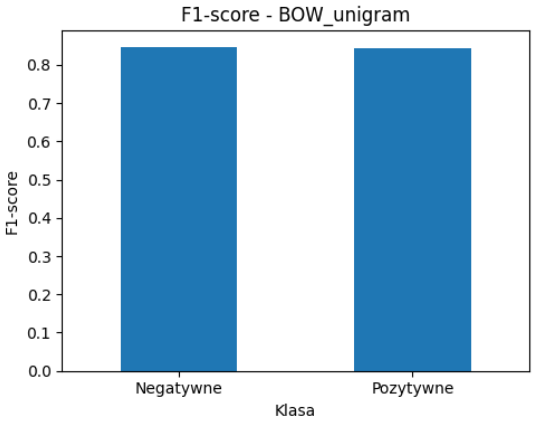
3.2.2.2 TF-IDF (bigramy)

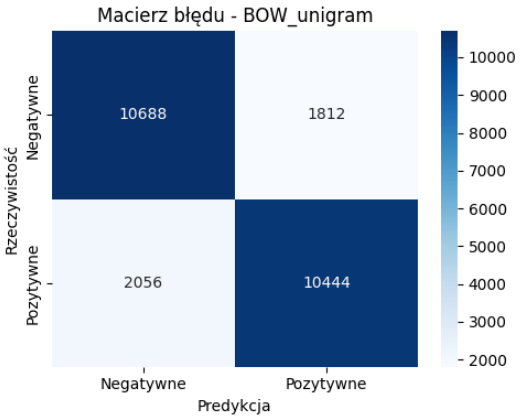




Sposób ten dzieli dzieli tekst na pojedyncze słowa oraz na pary słów. Wykrywa proste zależności kontekstowe. Wynik f1-score dla recenzji negatywnych to 87,7% a dla pozytywnych to 87,8%. Ogólna ocena skuteczności tego modelu wyniosła 87,8%

3.2.2.3 Bag of Words

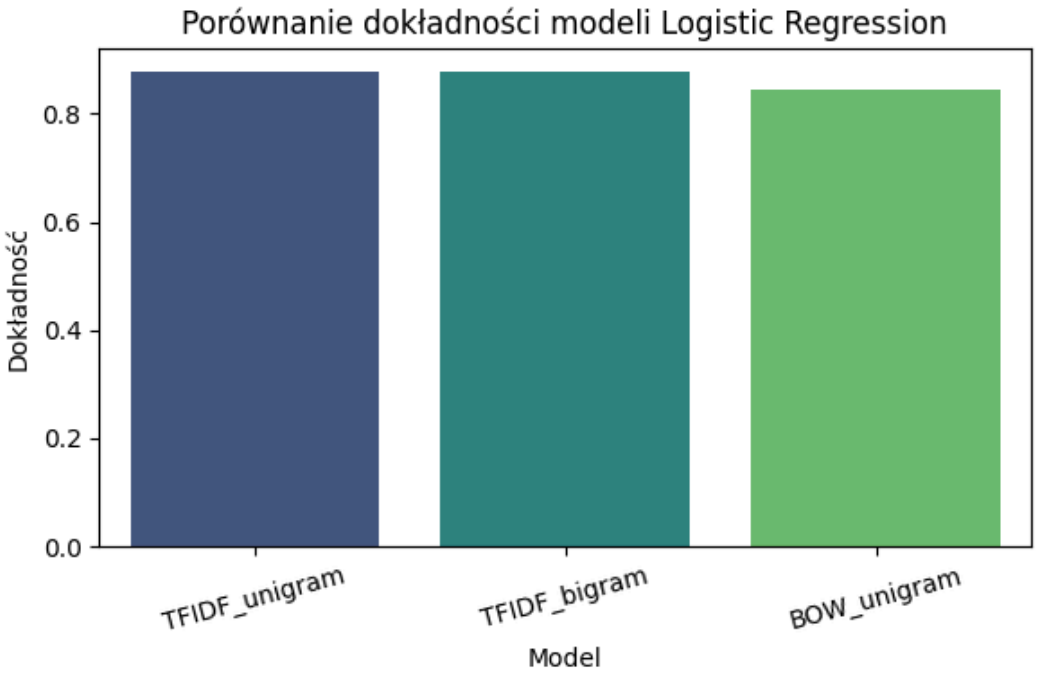




Skuteczność modelu dla recenzji pozytywnych wyniosła 84,3% a dla negatywnych 84,6%. Ogólna skuteczność była na poziomie 84,52%.

3.2.2.4 Podsumowanie

Podsumowując wyniki trzech różnych sposobów przygotowania tekstu do klasyfikatora Logistic Regression, najlepiej do niego pasuje sposób TF-IDF (bingram). Chociaż sposób TF-IDF (unigram) nie odstawał aż tak bo różnica między nimi wyniosła 0,1 %.



3.3 Model 3 - Recurrent Neural Network (LSTM)

3.3.1 Analiza Kodu

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np
import re
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tqdm import tqdm

def preprocess_text(text):
   text = text.lower()
   text = re.sub(r'<.*?>', '', text)
   text = re.sub(r'[^a-zA-Z ]', '', text)
   return text

# Wczytywanie danych
train_df = pd.read_csv("imdb_train.csv")
test_df = pd.read_csv("imdb_test.csv")

train_df['review'] = train_df['review'].apply(preprocess_text)
test_df['review'] = test_df['review'].apply(preprocess_text)

X_train = train_df['review'].values
y_train = train_df['sentiment'].values

X_test = test_df['review'].values
y_test = test_df['sentiment'].values

# Tokenizacja
max_words = 10000
max_len = 200
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)

X_train_seq = pad_sequences(tokenizer.texts_to_sequences(X_train), maxlen=max_len)
X_test_seq = pad_sequences(tokenizer.texts_to_sequences(X_test), maxlen=max_len)
```

```
# Dataset torch
class IMDBDataset(Dataset):
   def __init__(self, sequences, labels):
       self.sequences = sequences
       self.labels = labels

   def __len__(self):
       return len(self.sequences)

   def __getitem__(self, idx):
       return torch.tensor(self.sequences[idx], dtype=torch.long),
torch.tensor(self.labels[idx], dtype=torch.float)

train_dataset = IMDBDataset(X_train_seq, y_train)
test_dataset = IMDBDataset(X_test_seq, y_test)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)

# Model LSTM(S)
class SentimentLSTM(nn.Module):
   def __init__(self, vocab_size, embedding_dim, hidden_dim):
       super(SentimentLSTM, self).__init__()
       self.embedding = nn.Embedding(vocab_size, embedding_dim)
       self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
       self.fc = nn.Linear(hidden_dim, 1)
       self.sigmoid = nn.Sigmoid()

   def forward(self, x):
       x = self.embedding(x)
       _, (hidden, _) = self.lstm(x)
       out = self.fc(hidden[-1])
       return self.sigmoid(out)

# Inicjalizacja
vocab_size = min(len(tokenizer.word_index) + 1, max_words)
model = SentimentLSTM(vocab_size, embedding_dim=128, hidden_dim=64)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Trenowanie modelu
model.train()
```

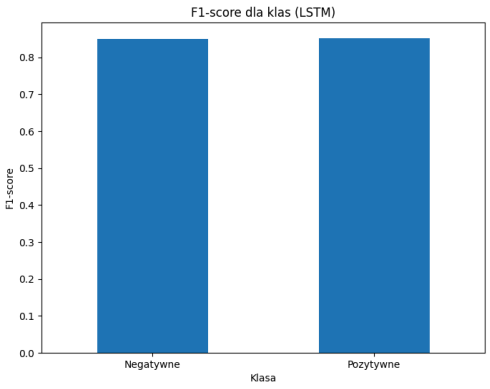
```
for epoch in range(15):
    total_loss = 0
    for inputs, labels in tqdm(train_loader):
        inputs, labels = inputs.to(device), labels.to(device).unsqueeze(1)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoka {epoch+1}, Strata: {total_loss/len(train_loader):.4f}")

# Ewaluacja
model.eval()
y_pred = []
y_true = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        outputs = model(inputs).cpu().numpy()
        preds = (outputs > 0.5).astype(int).flatten()
        y_pred.extend(preds)
        y_true.extend(labels.numpy())

acc = accuracy_score(y_true, y_pred)
print(f"Dokładność: {acc:.4f}")

cm = confusion_matrix(y_true, y_pred)

# Macierz błędu
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negatywne",
"Pozytywne"], yticklabels=["Negatywne", "Pozytywne"])
plt.title("Macierz pomyłek - LSTM")
plt.xlabel("Predykcja")
plt.ylabel("Rzeczywistość")
plt.savefig("confusion_matrix_lstm.png")
plt.show()

# Raport
report = classification_report(y_true, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df.to_csv("classification_report_lstm.csv")

# F1 score
```

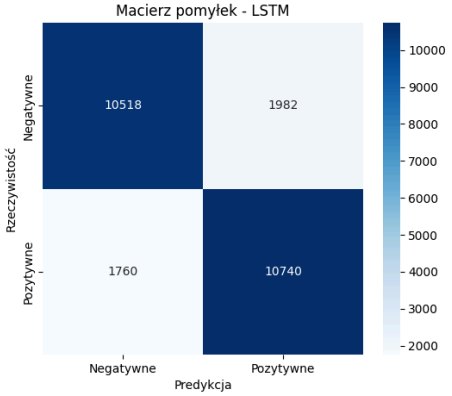
```
# Uniwersalne podejście: wykryj klasy liczbowe
class_labels = [label for label in report_df.index if label in ['0.0', '1.0', 0.0,
1.0, '0', '1', 0, 1]]

if class_labels:
   plt.figure(figsize=(8,6))
   report_df.loc[class_labels]["f1-score"].plot(kind='bar')
   plt.title("F1-score dla klas (LSTM)")
   plt.ylabel("F1-score")
   plt.xlabel("Klasa")
   plt.xticks(range(len(class_labels)), ["Negatywne", "Pozytywne"], rotation=0)
   plt.savefig("f1_score_per_class_lstm.png")
   plt.show()
else:
   print("Nie znaleziono etykiet klas 0/1 w classification_report.")
```

3.3.2 Analiza Wyników

Klasyfikator LSTM działa jako klasyfikator sekwencyjny, który “czyta” słowa jedno po drugim i uczy się, jak ich kolejność wpływa na znaczenie zdania. Na wejściu model otrzymuje sekwencję słów. Potem każde słowo jest zmieniane na wektor liczbowy. Następnie LSTM analizuje kolejność tych wektorów. Na wyjściu model generuje jedną wartość, w moim przypadku 0 lub 1.

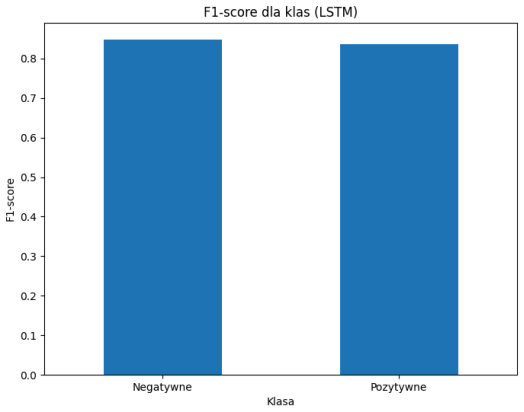
3.3.2.1 5-Epok

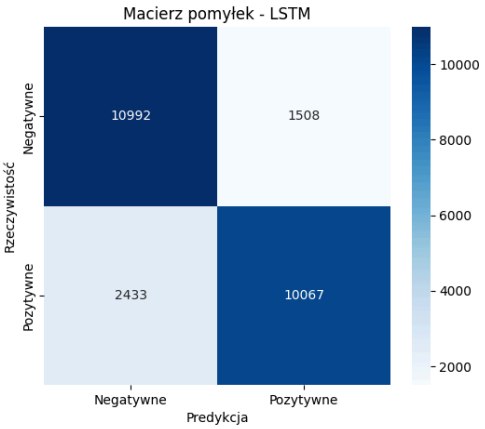




Model osiągnął skuteczność 85,03%. Wydaje się, że 5 epok to wystarczająco dużo, aby model nauczył się zależności w danych, a jednocześnie nie zdążył się przeuczyć.

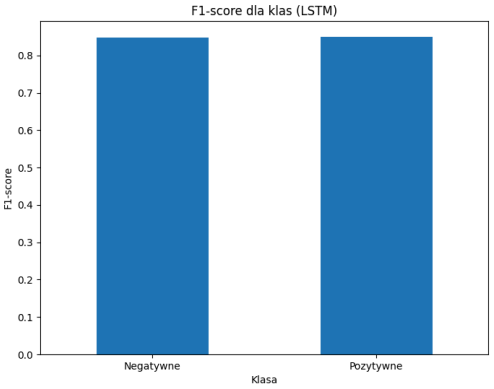
3.3.2.2 10-Epok

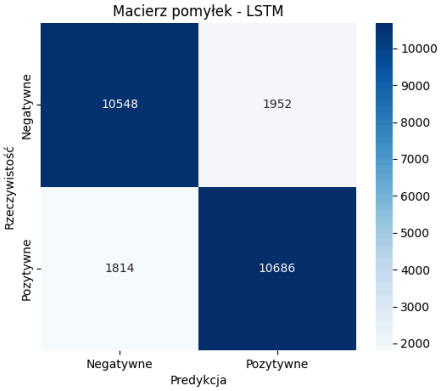




Model osiągnął wynik 84,24%. Spadek dokładności w porównaniu do 5 epok. Możliwe, że model zaczął się przeuczać, przez co gorzej generalizuje do danych testowych.

3.3.2.3 15-Epok





Model osiągnął wynik 84,94%. Lekka poprawa względem 10 epok, ale nadal wynik nie wrócił do poziomu z 5 epok. To sugeruje, że dalsze trenowanie nie wnosi już istotnej poprawy.

3.3.2.4 Podsumowanie

Liczba epok	Skuteczność	Czas Treningu	Uwagi
5	85,03%	~ 2 minuty	Najlepszy wynik
10	84,24%	~ 4 minut	Spadek - Możliwość przeuczenia
15	84,94%	~ 6 minut	Stabilizacja Wyników

Najllepszy wynik osiągnięto przy 5 epokach, dalsze trenowanie nie poprawi skuteczności, a jedynie skuteczność delikatnie spadała. Możliwe jest to że model zaczął się przyuczać od 10

epoki wzwyż. Podsumowując dla tego zbioru danych oraz dla tego modelu najlepszą ilością epok jeśli chodzi o czas nauki oraz efektywność to 5 epok.

3.4 Model 4 - Transfer Learning z BERT

3.4.1 Analiza Kodu

```
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

# Wczytywanie danych
data_train = pd.read_csv("imdb_train.csv")
data_test = pd.read_csv("imdb_test.csv")

# Dataset
class IMDBDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, index):
        text = str(self.texts[index])
        label = self.labels[index]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
```

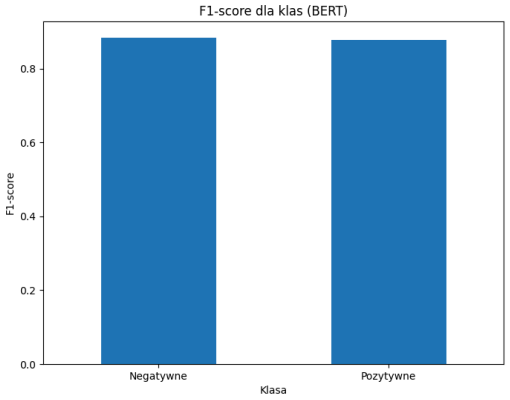
```
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Parametry
MAX_LEN = 128
BATCH_SIZE = 16
EPOCHS = 15
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

train_dataset = IMDBDataset(data_train['review'].values,
data_train['sentiment'].values, tokenizer, MAX_LEN)
test_dataset = IMDBDataset(data_test['review'].values,
data_test['sentiment'].values, tokenizer, MAX_LEN)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE)

# Model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=2)
model.to(device)

optimizer = AdamW(model.parameters(), lr=2e-5)

# Trening
model.train()
for epoch in range(EPOCHS):
   total_loss = 0
   for batch in tqdm(train_loader):
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask,
labels=labels)
        loss = outputs.loss
        loss.backward()
```

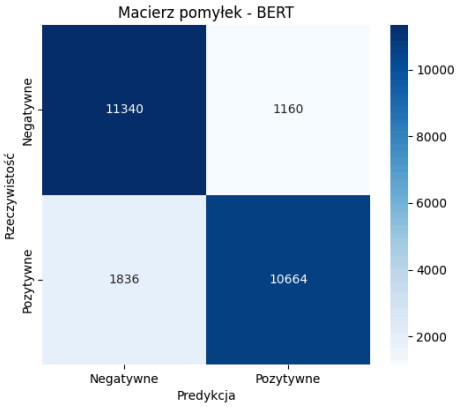
```
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoka {epoch+1}, Strata: {total_loss/len(train_loader):.4f}")

# Ewaluacja
model.eval()
y_pred = []
y_true = []
with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        preds = torch.argmax(outputs.logits, dim=1)
        y_pred.extend(preds.cpu().numpy())
        y_true.extend(labels.cpu().numpy())

# Metryki
acc = accuracy_score(y_true, y_pred)
print(f"Dokładność: {acc:.4f}")

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negatywne",
"Pozytywne"], yticklabels=["Negatywne", "Pozytywne"])
plt.title("Macierz pomyłek - BERT")
plt.xlabel("Predykcja")
plt.ylabel("Rzeczywistość")
plt.savefig("confusion_matrix_bert.png")
plt.show()

report = classification_report(y_true, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df.to_csv("classification_report_bert.csv")

plt.figure(figsize=(8,6))
report_df.loc[['0','1']]["f1-score"].plot(kind='bar')
plt.title("F1-score dla klas (BERT)")
plt.ylabel("F1-score")
plt.xlabel("Klasa")
plt.xticks([0,1],["Negatywne", "Pozytywne"], rotation=0)
plt.savefig("f1_score_per_class_bert.png")
plt.show()
```

3.4.2 Analiza Wyników

BERT - (Bidirectional Encoder Representations from Transformers) jest to model językowy oparty na architekturze Transformer. Model ten Rozumie kontekst dwukierunkowo, analizuje jednocześnie słowa przed i po danym słowie w zdaniu. Został wytrenowany wcześniej na ogromnych zbiorach tekstów między innymi na Wikipedii. BERT pierwotnie był trenowany w dwóch zadaniach: Masked Language Modeling(MLM) oraz Next Sentence Prediction (NSP). Dzięki temu BERT rozumie strukturę języka, związki gramatyczne i znaczenie słów w kontekście. W trakcie właściwego transfer learningu. Wczytuje gotowy model który już zna angielski - bert-base-uncased. Później dodaj swoją warstwę klasyfikacyjną. Następnie model jest trenowany a finalnie przewiduje on emocje na podstawie kontekstu całych recenzji.

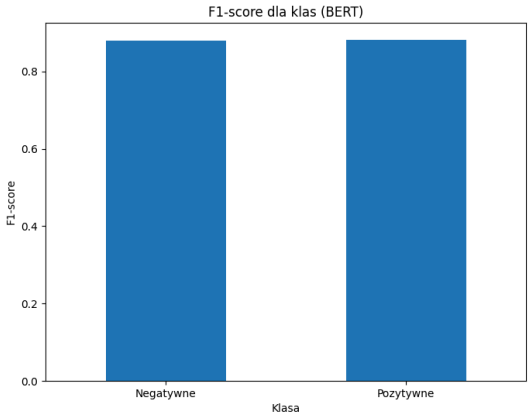
3.4.2.1 5-Epok

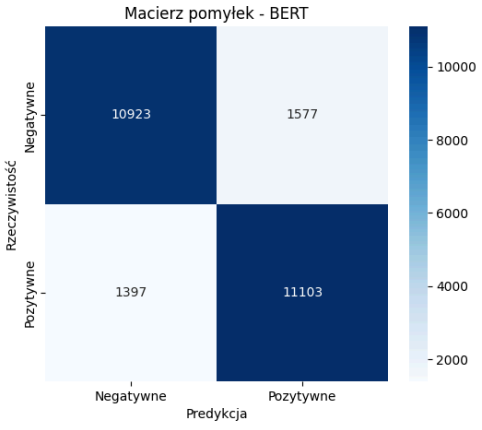




Już po 5 epokach model BERT osiągnął bardzo dobrą skuteczność - 88,02%. Model uczył się przez 5 godzin.

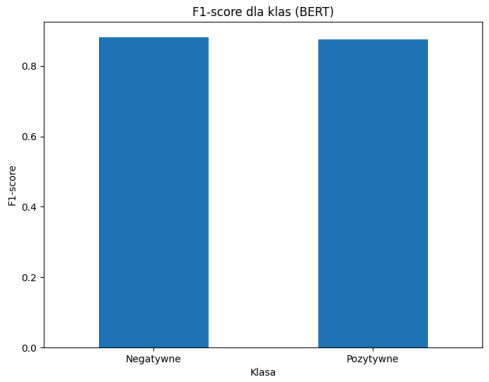
3.4.2.2 10-Epok

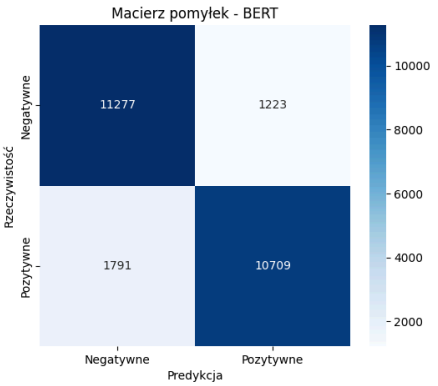




Po 10 epokach skuteczność modelu wyniosła 88,10%. Natomiast czas nauki nie zmienił się, model uczył się 10 godzin. Co pokazuje że model się stabilizuje ale za dużym kosztem.

3.4.2.3 15-Epok

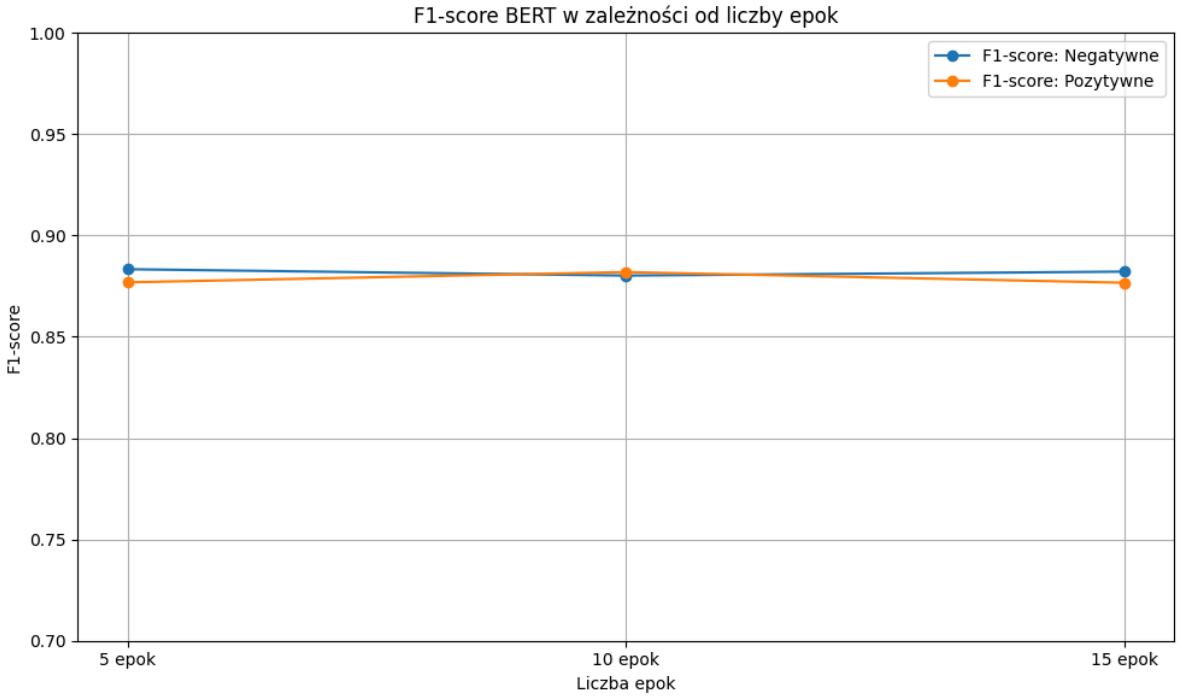




Po 15 epokach model obniżył znacząco swoją skuteczność gdyż wynik wyniósł 87.94%. Może to świadczyć o przeuczeniu modelu

3.4.2.4 Podsumowanie

Liczba epok	Skuteczność	Czas treningu	Uwagi
5	88.02%	~ 5 godzin	Wysoka skuteczność
10	88.10%	~ 10 godzin	Najlepszy wynik
15	87.94%	~ 15 godzin	Spadek - możliwość przeuczenia



4. Podsumowanie

Pytanie badawcze 1.
Które modele inteligencji obliczeniowej osiągają najlepsze wyniki w zadaniu analizy emocji?

W oparciu o przeprowadzone eksperymenty

Model	Reprezentacja	Najlepsza skuteczność
Naive Bayes	BOW (bigram)	84.27%
Logistic Regression	TF-IDF(bigram)	87.80%
LSTM	Word Embedding	85.03% (5 epok)
BERT (Transfer Learning)	Embedding kontekstowy	88.10% (10 epok)

Najlepsze wyniki uzyskał model BERT, przewyższając pozostałe klasyfikatory zarówno klasyczne, jak i oparte o RNN. Jednakże jeśli weźmiemy pod uwagę długość nauki modelu to najlepiej wychodzi Logistic Regression gdyż czas nauki zajął około minuty do dwóch, a BERT uczył się 12 godzin przy 10 epokach.

Pytanie badawcze 2.
Czy nowoczesne modele głębokiego uczenia, takie jak BERT, przewyższają klasyczne podejścia (np. Naive Bayes, SVM)?

Tak. BERT wyraźnie przewyższa klasyczne podejścia:
- O około 6% więcej niż Naive Bayes
- O około 0.3% więcej niż Logistic Regression, choć różnica jest niewielka, Logistic Regression okazało się bardzo konkurencyjne przy znacznie niższych kosztach obliczeniowych.

Jednak koszt czasowy i zasoby GPU potrzebne do trenowania BERT-a są nieporównywalnie wyższe niż przy klasycznych modelach.

Pytanie badawcze 3.
Jakie znaczenie mają metody reprezentacji tekstu (np. TF-IDF vs. embeddingi kontekstowe) dla skuteczności klasyfikatorów?

Reprezentacja tekstu ma kluczowe znaczenie:
- TF-IDF (unigram/bigram) okazał się bardzo skuteczny dla klasycznych modeli.
- BOW był mniej efektywny, szczególnie w wariancie unigramowym.
- Embeddingi (LSTM) nie przebił wyników TF-IDF, ale dały dobre rezultaty bez potrzeby inżynierii cech.
- Embeddingi kontekstowe (BERT) osiągnęły najwyższą skuteczność, ponieważ potrafią uchwycić znaczenie słów w kontekście, czego klasyczne podejścia nie robią

Podsumowując. BERT to najskuteczniejszy model, ale jego użycie wiąże się z dużym nakładem czasowym i sprzętowym. Logistic Regression natomiast w użyciu z TF-IDF bigram jest najlepszym podejściem kompromisowym (wysoka skuteczność przy niskich kosztach). Model Naive Bayes zaś jest najprostszym i najszybszym rozwiązaniem, ale wyraźnie słabszym jakościowo. Nie można zapomnieć też o tym, że reprezentacja danych tekstowych ma ogromny wpływ na efektywność modeli. Bardziej zaawansowane reprezentacje zwykle dają lepsze wyniki.