Automatyczna Klasyfikacja Tematyczna Polskich Artykulow Newsowych

Maciej Biegan Kamil Dziedzic Jan Bobak

Eksploracja Danych Tekstowych – Grudzien 2025

Spis treści

1 Wstep

1.1 Opis Problemu

Automatyczne tagowanie artykulow (klasyfikacja tematyczna) to zadanie przypisania dokumentu tekstowego do jednej z predefiniowanych kategorii na podstawie jego tresci. Problem nalezy do domeny klasyfikacji wieloklasowej tekstu w przetwarzaniu jezyka naturalnego (NLP).

Formalnie: Dla dokumentu d i zbioru kategorii $C = \{c_1, c_2, ..., c_k\}$ szukamy funkcji $f : D \to C$ maksymalizujacej prawdopodobienstwo poprawnej klasyfikacji.

W niniejszym projekcie $k = 6$ kategorii:

- **Polska** – polityka krajowa, sprawy wewnetrzne
- **Swiat** – wydarzenia miedzynarodowe
- **Biznes** – gospodarka, finanse, rynki
- **Technologie** – IT, innowacje, startupy
- **Moto** – motoryzacja, samochody
- **Sport** – wydarzenia sportowe, wyniki

1.2 Wyzwania dla Jezyka Polskiego

Klasyfikacja polskich tekstów jest trudniejsza niż angielskich ze względu na kilka czynników. Po pierwsze, skomplokowana budowa języka polskiego, z siedmioma przypadkami, co utrudnia proces lematyzacji. Po drugie, polski ma swobodny szyk zdania, gdzie kolejność słów nie zawsze determinuje ich funkcję, co utrudnia analizę składniową. Po trzecie, brak gotowych danych do trenowania modeli.

1.3 Cel Projektu

Celem projektu było opracowanie systemu pozwalającego na:

1. Automatyczny scraping artykulow z kanalow RSS
2. Pipeline preprocessingu NLP (normalizacja, lematyzacja, wektoryzacja)
3. Trening i ewaluacja trzech architektur modeli
4. Aplikacja webowa do predykcji i trenowania

2 Metody Przetwarzania Jezyka Naturalnego

2.1 Pipeline Przetwarzania

Tekst przechodzi sekwencje transformacji:

$$\text{RAW} \xrightarrow{\text{normalize}} \text{CLEAN} \xrightarrow{\text{tokenize}} \text{TOKENS} \xrightarrow{\text{lemmatize}} \text{LEMMAS} \xrightarrow{\text{vectorize}} \mathbb{R}^n$$

2.2 Normalizacja Tekstu

Funkcja `normalize_text()` w `Scrapper/scrapper.py` wykonuje:

Listing 1: Implementacja normalizacji tekstu

```python
import re

RE_URL = re.compile(r'https?://\S+|www\.\S+')
RE_NON_LETTER = re.compile(r'[^a-zA-ZacelnoszzACELNOSZZ\s-]')
RE_MULTI_WS = re.compile(r'\s+')

def normalize_text(text: str) -> str:
    if not isinstance(text, str):
        return ''
    text = text.strip()
    text = RE_URL.sub(' ', text)       # usuwa URLs
    text = text.replace('\xa0', ' ')   # non-breaking space
    text = RE_NON_LETTER.sub(' ', text)  # tylko litery
    text = RE_MULTI_WS.sub(' ', text)    # normalizacja spacji
    text = text.lower()
    return text
```

Operacje:

- `RE_URL.sub(' ', text)` – usuwa linki HTTP/HTTPS (brak informacji semantycznej)
- `RE_NON_LETTER.sub(' ', text)` – zachowuje tylko litery ASCII i polskie diakrytyki
- `text.lower()` – unifikacja wielkosci liter

2.3 Lematyzacja z spaCy

Lematyzacja redukuje formy fleksyjne do formy podstawowej. Model `pl_core_news_sm` (15MB) trenowany na NKJP.

Listing 2: Funkcja lematyzacji i tokenizacji

```python
import spacy
from stop_words import get_stop_words

def get_polish_stopwords() -> Set[str]:
    nlp = spacy.load('pl_core_news_sm')
    spacy_stopwords = set(nlp.Defaults.stop_words)
    base_stopwords = set(get_stop_words('polish'))
    extra_stops = {'z', 'na', 'i', 'w', 'o', 'ze', 'sie',
                   'roku', 'r', 'godz', 'fot', 'foto'}
    return base_stopwords | spacy_stopwords | extra_stops

def lemmatize_and_tokenize(text, nlp_model, stopwords,
                           min_token_len=3):
    text_norm = normalize_text(text)
    doc = nlp_model(text_norm)
    tokens = []
    for tok in doc:
        if tok.is_space or tok.is_punct:
            continue
```

```
        lemma = tok.lemma_.lower().strip()
        if len(lemma) < min_token_len:
            continue
        if lemma in stopwords or lemma.isdigit():
            continue
        tokens.append(lemma)
    return tokens
```

Przyklad transformacji:

```
Input:  "Prezydent podpisal ustawe o reformie szpitali"
Output: ["prezydent", "podpisac", "ustawa", "reformia", "szpital"]
```

Redukcja: 7 slow $\rightarrow$ 5 lemmatow (28% kompresji).

2.4 Wektoryzacja TF-IDF

Term Frequency-Inverse Document Frequency przeksztalca tekst w wektor numeryczny.

Definicja:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D) \tag{1}$$

Listing 3: Konfiguracja TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(
    max_features=5000,       # limit wymiarowosci
    ngram_range=(1, 2),      # unigramy + bigramy
    sublinear_tf=True,       # logarytmiczne TF
    min_df=2,                # min 2 dokumenty
    max_df=0.95              # max 95% dokumentow
)
X = vectorizer.fit_transform(df['text']).toarray()
```

N-gramy: Bigramy chwytaja frazy: „liga_mistrzow”, „premier_polski”, „gielda_papierow”.

2.5 Embeddingi Kontekstowe (Transformery)

W przeciwienstwie do TF-IDF (bag-of-words), transformery generuja embeddingi zalezne od kontekstu.

Self-Attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

gdzie Q, K, V to projekcje liniowe wejscia, d_k – wymiar klucza.

Tokenizacja WordPiece:

Listing 4: Tokenizacja dla HerBERT

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(
    'allegro/herbert-base-cased'
)

```

```
def tokenize_fn(examples):
    return tokenizer(
        examples["text"],
        padding="max_length",
        truncation=True,
        max_length=256
    )
```

WordPiece dzieli nieznane slowa na subwordy:

„niesamowity” → [„nie”, „##sam”, „##owity”]

3 Zrodla Danych

3.1 Kanaly RSS

Dane pochodza z 26 kanalow RSS polskich portali informacyjnych:

Tabela 1: Zrodla RSS per kategoria

Kategoria	Portal	Domena
Polska	Polsat News, TVN24, WP, Fakt, Do Rzeczy, Wprost	6 zrodel
Swiat	Polsat News, TVN24, RMF24, Newsweek	4 zrodla
Biznes	Polsat News, TVN24 Biznes, Puls Biznesu	3 zrodla
Technologie	Polsat News, TVN24, Computerworld, Spider's Web	4 zrodla
Moto	Polsat News, TVN24 Moto	2 zrodla
Sport	Polsat News, TVN24 Sport, Onet Sport	3 zrodla

3.2 Architektura Scrapera

Implementacja w `Scrapper/scrapper.py`:

Listing 5: Slownik zrodel RSS

```
FEEDS = {
    'Polska': [
        'https://www.polsatnews.pl/rss/polska.xml',
        'https://tvn24.pl/polska.xml',
        'https://wiadomosci.wp.pl/rss.xml',
        'https://www.fakt.pl/rss',
        'https://dorzeczy.pl/rss',
        'https://www.wprost.pl/rss'
    ],
    'Swiat': [
        'https://www.polsatnews.pl/rss/swiat.xml',
        'https://tvn24.pl/swiat.xml',
        'https://www.rmf24.pl/feed/',
        'https://www.newsweek.pl/rss'
    ],
    # ... pozostale kategorie
}
```

3.3 Ekstrakcja Pelnej Tresci

RSS zawiera tylko tytul i streszczenie. Pelna tresc wymaga pobrania HTML:

Listing 6: Ekstrakcja tekstu z URL

```
from newspaper import Article
from bs4 import BeautifulSoup
import requests

def extract_full_text(url: str, timeout: int = 10) -> str:
    headers = {'User-Agent': 'Mozilla/5.0'}

    # Metoda 1: newspaper3k
    try:
        article = Article(url)
        article.download()
        article.parse()
        if article.text and len(article.text) > 100:
            return article.text
    except Exception:
        pass

    # Metoda 2: BeautifulSoup fallback
    try:
        resp = requests.get(url, headers=headers, timeout=timeout)
        soup = BeautifulSoup(resp.content, 'lxml')

        selectors = ["article", "div[class*='article']",
                     "div[class*='content']"]
        for sel in selectors:
            el = soup.select_one(sel)
            if el:
                ps = el.find_all('p')
                text = '\n'.join([p.get_text(strip=True)
                                  for p in ps])
                if len(text) > 100:
                    return text
    except Exception:
        pass

    return None
```

3.4 Filtracja Jakosciowa

Listing 7: Kryteria filtracji w funkcji fetch_articles_from_feeds

```
def fetch_articles_from_feeds(feeds, min_length=200):
    # ...
    for entry in category_entries:
        text = extract_full_text(link)
        text = clean_text(text)

        # Filtr 1: minimalna dlugosc
        if len(text) < min_length:
            continue

        # Filtr 2: wykrywanie jezyka
```

```
        if detect(text) != 'pl':
            continue

        records.append({...})

    # Filtr 3: deduplikacja
    df.drop_duplicates(subset=['url', 'text'], inplace=True)
    return df
```

3.5 Charakterystyka Datasetu

Wynikowy plik: `Scrapper/polsatnews_articles_clean.csv`

Tabela 2: Rozklad artykulow per kategoria

Kategoria	Liczba	Udzial [%]	Zrodla
Polska	934	26.8	6
Sport	762	21.9	3
Swiat	708	20.3	4
Biznes	463	13.3	3
Technologie	389	11.2	4
Moto	231	6.6	2
Razem	**3487**	**100**	**26**

Niezbalansowanie: Polska (26.8%) vs Moto (6.6%) – stosunek 4:1. Zastosowano `class_weight='balanced'`.

4 Modele

W projekcie zaimplementowano i porownano trzy rozne architektury modeli uczenia maszynowego do klasyfikacji tekstu. Kazdy z modeli wykorzystuje inna reprezentacje tekstu oraz inny mechanizm uczenia. Model MLP operuje na wektorach TF-IDF i stanowi klasyczne podejscie oparte na plytkim uczeniu. Modele transformerowe HerBERT i BERT wykorzystuja mechanizm self-attention i pretrenowane reprezentacje jezykowe, co pozwala na lepsze zrozumienie kontekstu i semantyki tekstu. Wszystkie modele zostaly zaimplementowane w PyTorch z wykorzystaniem biblioteki Hugging Face Transformers dla modeli BERT.

4.1 MLP

Model MLP jest wielowarstwowa siecia neuronowa typu feed-forward, ktora przyjmuje na wejsciu wektor TF-IDF o wymiarze 5000 i klasyfikuje go do jednej z 6 kategorii. Architektura sieci jest w pelni konfigurowalna i pozwala na zmiane liczby warstw ukrytych, rozmiaru warstw oraz wspolczynnika dropout.

4.1.1 Architektura

Klasa TextClassifier dziedziczy po nn.Module i buduje siec sekwencyjnie. Pierwsza warstwa liniowa redukuje wymiarowosc z 5000 cech TF-IDF do rozmiaru warstwy ukrytej (domyslnie 256). Nastepnie stosowana jest normalizacja LayerNorm, funkcja aktywacji GELU oraz regularyzacja Dropout. Ten blok powtarzany jest dla kazdej warstwy ukrytej. Ostatnia warstwa liniowa mapuje reprezentacje na 6 klas wyjsciowych.

Implementacja w `models/model.py`:

Listing 8: Klasa TextClassifier

```
import torch
import torch.nn as nn

class TextClassifier(nn.Module):
    def __init__(self, input_size: int, hidden_size: int,
                 num_classes: int, num_layers: int = 3,
                 dropout: float = 0.4):
        super().__init__()

        layers = [
            nn.Linear(input_size, hidden_size),
            nn.LayerNorm(hidden_size),
            nn.GELU(),
            nn.Dropout(dropout),
        ]

        for _ in range(num_layers - 1):
            layers.extend([
                nn.Linear(hidden_size, hidden_size),
                nn.LayerNorm(hidden_size),
                nn.GELU(),
                nn.Dropout(dropout),
            ])

        layers.append(nn.Linear(hidden_size, num_classes))
        self.network = nn.Sequential(*layers)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.network(x)
```

4.1.2 Komponenty

Architektura MLP sklada sie z kilku kluczowych komponentow, ktore wspolpracuja w celu efektywnego przetwarzania i klasyfikacji tekstu.

LayerNorm normalizuje aktywacje po wymiarze cech, a nie po wymiarze batcha jak BatchNorm. Dla kazdego przykladuw batchu obliczana jest srednia i odchylenie standardowe po wszystkich cechach, a nastepnie aktywacje sa standaryzowane. Parametry gamma i beta sa uczone podczas treningu i pozwalaja sieci na skalowanie i przesuwanie znormalizowanych wartosci. LayerNorm jest szczegolnie przydatna w NLP, poniewaz dziala stabilnie niezaleznie od rozmiaru batcha.

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sigma + \epsilon} + \beta \tag{3}$$

GELU jest funkcja aktywacji, ktora mnozy wejscie przez wartosc dystrybuanty rozkladu normalnego. W przeciwienstwie do ReLU, ktora ostro obcina wartosci ujemne do

zera, GELU zapewnia gladkie przejscie w okolicy zera. Dzieki temu gradienty sa lepiej propagowane przez siec, co jest szczegolnie korzystne w zadaniach NLP. GELU jest standardowa funkcja aktywacji w modelach BERT i GPT.

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{4}$$

Dropout jest technika regularyzacji, ktora podczas treningu losowo zeruje czesc neuronow z prawdopodobienstwem p. Zapobiega to nadmiernemu dopasowaniu modelu do danych treningowych (overfitting), poniewaz siec nie moze polegac na zadnym pojedynczym neuronie. Podczas inferencji dropout jest wylaczany, a wagi sa skalowane przez (1-p) aby zachowac oczekiwana wartosc aktywacji.

$$\text{Dropout}(x) = \begin{cases} 0 & \text{z prawdop. } p \\ \frac{x}{1-p} & \text{wpp.} \end{cases} \tag{5}$$

4.1.3 Parametry Konfiguracyjne

Definicja w `models/config.py`:

Listing 9: MLPConfig dataclass

```python
@dataclass
class MLPConfig:
    epochs: int = 20
    batch_size: int = 32
    learning_rate: float = 0.001
    max_features: int = 5000
    hidden_size: int = 256
    num_layers: int = 3
    dropout: float = 0.4
    early_stopping: bool = False
    early_stopping_patience: int = 3
```

4.1.4 Trening

Listing 10: Petla treningowa MLP (fragment _train_mlp)

```python
from sklearn.utils.class_weight import compute_class_weight

# Wagi klas dla niezbalansowanego datasetu
class_weights = torch.tensor(
    compute_class_weight('balanced',
                         classes=np.unique(y_train),
                         y=y_train),
    dtype=torch.float32
).to(DEVICE)

criterion = nn.CrossEntropyLoss(weight=class_weights)
optimizer = torch.optim.AdamW(model.parameters(),
                              lr=config.learning_rate)

for epoch in range(config.epochs):
    model.train()
    for X_batch, y_batch in train_loader:
```

```
        optimizer.zero_grad()
        outputs = model(X_batch.to(DEVICE))
        loss = criterion(outputs, y_batch.to(DEVICE))
        loss.backward()
        optimizer.step()
```

4.2 HerBERT

Polski transformer BERT pretrenowany przez Allegro.pl.

Tabela 3: Specyfikacja HerBERT

Parametr	Wartosc
Model	allegro/herbert-base-cased
Warstwy Transformer	12
Wymiar ukryty	768
Attention heads	12
Parametry	110M
Korpus treningowy	14GB polskiego tekstu
Tokenizer	SentencePiece (BPE)
Vocab size	50,000

4.3 BERT Multilingual

Model `bert-base-multilingual-cased` trenowany na 104 jezykach:

- Vocab size: 119,547 tokenow
- Polski: ok. 3% korpusu
- Gorsze wyniki niz HerBERT (brak specjalizacji)

4.4 Konfiguracja Transformerow

Listing 11: TransformerConfig dataclass

```
@dataclass
class TransformerConfig:
    epochs: int = 10
    batch_size: int = 16
    eval_batch_size: int = 64
    warmup_steps: int = 500
    weight_decay: float = 0.01
    learning_rate: float = 2e-5
    max_length: int = 256
    early_stopping: bool = False
    early_stopping_patience: int = 3
```

4.5 Fine-tuning

Listing 12: Fine-tuning transformera (fragment _train_transformer)

```
from transformers import (AutoModelForSequenceClassification,
                          Trainer, TrainingArguments)

model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAMES[model_type],
    num_labels=len(label_encoder.classes_)
).to(DEVICE)

training_args = TrainingArguments(
    output_dir=str(PATHS.results_dir),
    num_train_epochs=config.epochs,
    per_device_train_batch_size=config.batch_size,
    learning_rate=config.learning_rate,
    warmup_steps=config.warmup_steps,
    weight_decay=config.weight_decay,
    eval_strategy='epoch',
    save_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='accuracy',
    fp16=torch.cuda.is_available(),
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
)
trainer.train()
```

5 Aplikacja Webowa

System klasyfikacji zostal wdrozony jako aplikacja webowa, ktora umozliwia uzytkownikom klasyfikowanie artykulow newsowych w czasie rzeczywistym oraz trenowanie wlasnych modeli. Aplikacja zostala zbudowana z wykorzystaniem nowoczesnych technologii webowych i zaprojektowana z mysla o skalowalnosci i latwosci uzytkowania.

5.1 Architektura

Aplikacja zostala zaimplementowana w jezyku Python z wykorzystaniem frameworka Flask, ktory jest lekkim frameworkiem webowym opartym na bibliotece Werkzeug i silniku szablonow Jinja2. Architektura aplikacji opiera sie na wzorcu MVC, gdzie modele uczenia maszynowego stanowia warstwe Model, szablony HTML stanowia warstwe View, a funkcje widokow Flask stanowia warstwe Controller.

Glowny plik aplikacji `app.py` definiuje endpointy HTTP i zarzadza logika biznesowa. Modele ML sa enkapsulowane w klasie ModelManager, ktora zapewnia thread-safe dostep do modeli z wielu watkow. Szablony HTML wykorzystuja Bootstrap do stylowania i JavaScript do dynamicznej aktualizacji interfejsu.

Listing 13: Klasa ModelManager

```
class ModelManager:
    """Thread-safe manager dla modeli ML"""

    def __init__(self, default_model_type="herbert"):
        self.model_type = default_model_type
        self.model = None
        self.tokenizer_or_vectorizer = None
        self.label_encoder = None
        self._lock = threading.Lock()
        self.load()

    def load(self, model_type=None):
        with self._lock:
            if model_type:
                self.model_type = model_type
            (self.model,
             self.tokenizer_or_vectorizer,
             self.label_encoder) = load_model(self.model_type)

    def predict(self, text: str):
        if self.model is None:
            raise RuntimeError("Model not loaded")
        with self._lock:
            return predict_category(
                text, self.model,
                self.tokenizer_or_vectorizer,
                self.label_encoder,
                self.model_type
            )
```

5.2 Endpointy

Tabela 4: REST API

Endpoint	Metody	Opis
/	GET, POST	Predykcja kategorii
/train	GET, POST	Formularz/start treningu
/training_status	GET	Status treningu (JSON)
/training_result	GET	Wyniki treningu (JSON)

5.3 Obsluga Wejscia

Trzy zrodla tekstu: bezposredni input, upload pliku, URL artykulu.

Listing 14: Ekstrakcja tekstu z requestu

```
def extract_text_from_request(form, files):
    # Opcja 1: tekst bezposredni
    if form.get("text", "").strip():
        return form["text"].strip()

    # Opcja 2: upload pliku .txt
```

```
    if "file" in files and files["file"].filename:
        return files["file"].read().decode("utf-8")

    # Opcja 3: URL artykulu
    if form.get("url", "").strip():
        return extract_text_from_url(form["url"].strip())

    return None
```

5.4 Asynchroniczne Trenowanie

Trening w osobnym watku, aby nie blokowac UI:

Listing 15: Trenowanie w tle

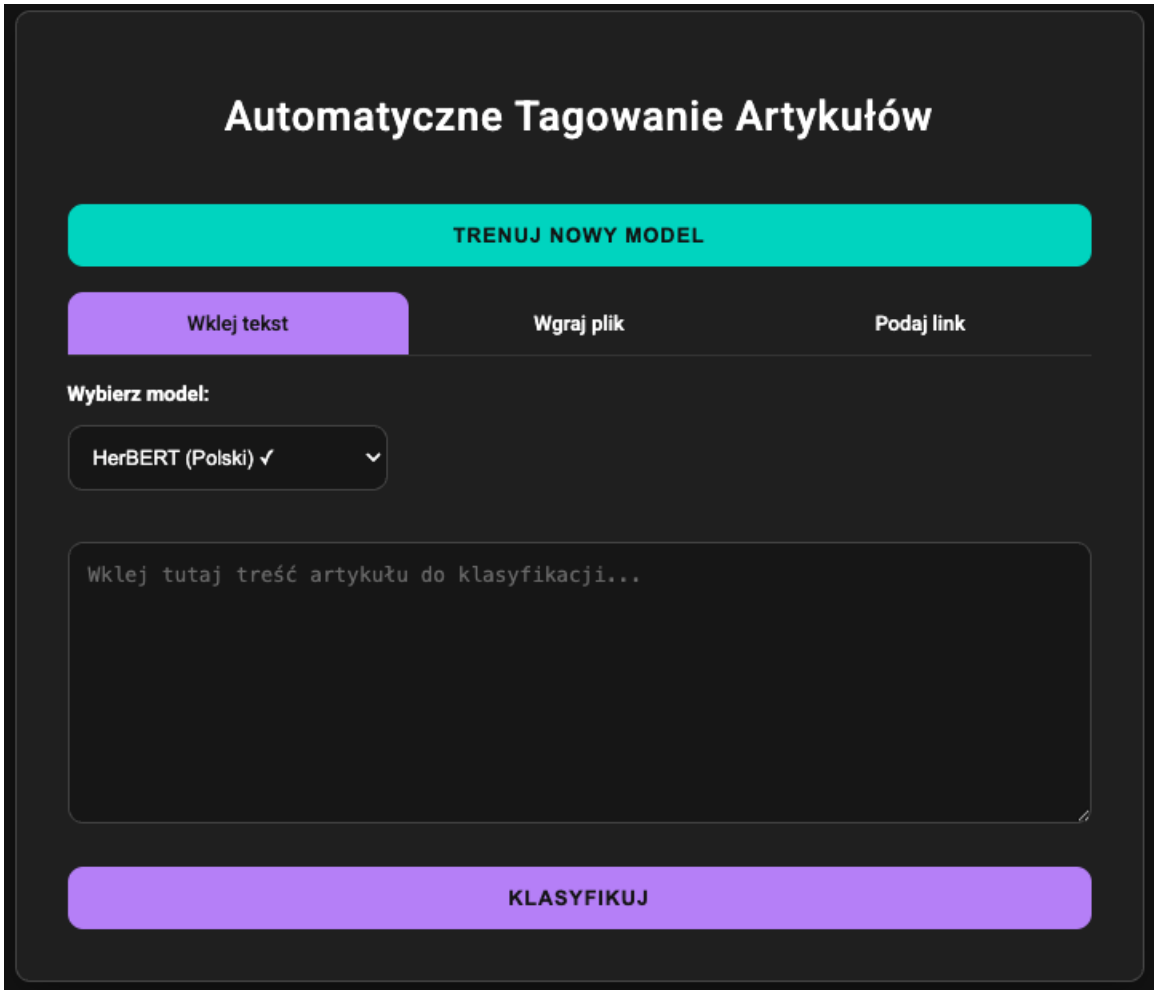
```
def _background_train(params):
    try:
        train_model_with_params(params)
        model_manager.load(params.get("model_type"))

        with model_module._status_lock:
            model_module.training_status["message"] = "completed"
    except Exception as e:
        with model_module._status_lock:
            model_module.training_status.update(
                {"running": False, "error": str(e)}
            )
    finally:
        with model_module._status_lock:
            model_module.training_status["running"] = False

@app.route('/train', methods=['POST'])
def train():
    # ... parsowanie parametrow ...
    thread = threading.Thread(
        target=_background_train,
        args=(params,),
        daemon=True
    )
    thread.start()
    return render_template('train.html', training=True)
```

5.5 Interfejs Uzytkownika

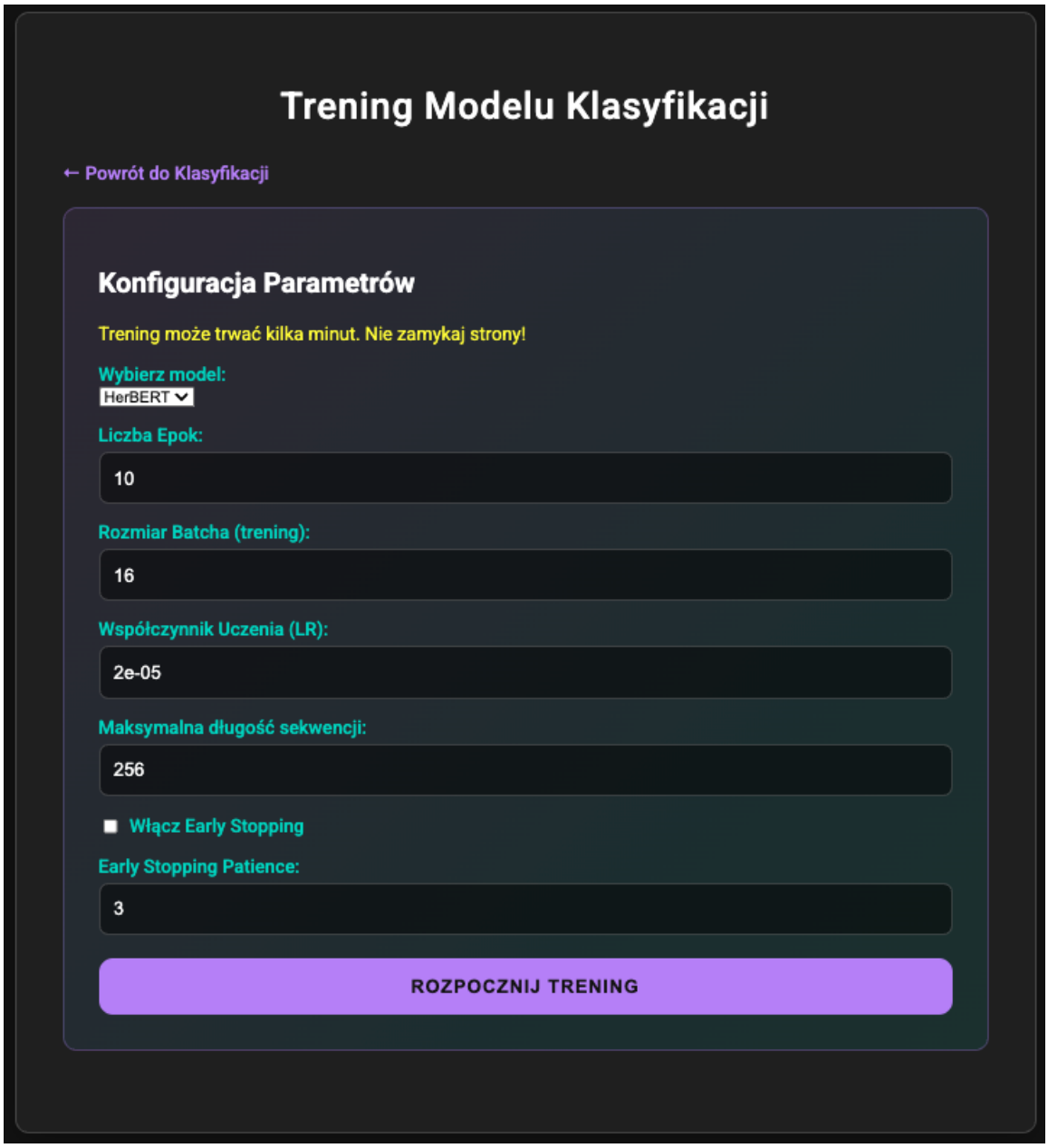
Interfejs uzytkownika zostal zaprojektowany z mysla o prostocie i intuicyjnosci. Aplikacja sklada sie z dwoch glownych widokow: strony glownej do klasyfikacji tekstu oraz panelu trenowania modeli.

Strona glowna umozliwia uzytkownikowi wprowadzenie tekstu do klasyfikacji na trzy sposoby. Pierwszy sposob to bezposrednie wpisanie lub wklejenie tekstu artykulu w pole tekstowe. Drugi sposob to upload pliku tekstowego w formacie UTF-8. Trzeci sposob to podanie URL artykulu, z ktorego tekst zostanie automatycznie wyodrebniony za pomoca biblioteki newspaper3k. Uzytkownik moze rowniez wybrac model do klasyfikacji sposrod dostepnych: HerBERT, BERT multilingual lub MLP. Po wyslaniu formularza wyswietlana jest przewidziana kategoria wraz z poziomem pewnosci modelu wyrażonym w procentach.

Panel trenowania pozwala na konfiguracje i uruchomienie treningu nowego modelu. Uzytkownik moze wybrac typ modelu oraz dostosowac hiperparametry takie jak liczba epok, learning rate, batch size, dropout oraz maksymalna liczba cech TF-IDF dla modelu MLP. Podczas treningu wyswietlany jest pasek postepu oraz biezace metryki: numer epoki, wartosc funkcji straty oraz dokladnosc na zbiorze walidacyjnym. Po zakonczeniu treningu wyswietlane sa wykresy krzywej uczenia.



Rysunek 1: Strona glowna aplikacji z formularzem do wprowadzania tekstu



Rysunek 2: Panel trenowania z parametrami i paskiem postepu

6 Wyniki

6.1 Metodologia

Podzial stratyfikowany 80/20 (train/test) z `random_state=42`.

Metryki:

- Accuracy
- Precision
- Recall
- F1-Score

6.2 Porownanie Modeli

Tabela 5: Wyniki modeli

Model	F1	Accuracy	Czas [s]	Parametry
HerBERT	**0.895**	**0.897**	847	110M
BERT (multi)	0.872	0.876	892	110M
MLP (best)	0.817	0.825	1.09	265k

Interpretacja:

- HerBERT > BERT o 2.3 p.p. – potwierdza wartosc specjalizacji jezykowej
- MLP osiaga 91% wydajnosci HerBERT przy 777× krotszym czasie treningu

6.3 Analiza Hiperparametrow MLP

W celu znalezienia optymalnej konfiguracji modelu MLP przeprowadzono systematyczna analize wplywu pieciu hiperparametrow na jakosc klasyfikacji. Przetestowano lacznie 12 roznych konfiguracji, a wyniki poddano analizie statystycznej.

Parametr **architektura** okresla liczbe i rozmiar warstw ukrytych sieci. Testowano architektury od plytkich (2 warstwy: 128-64) do glebszych (3-4 warstwy: 512-256-128). Wieksza liczba warstw teoretycznie pozwala na uczenie bardziej zlozonych reprezentacji, ale moze prowadzic do overfittingu na malych zbiorach danych.

Parametr **learning rate** okresla szybkosc uczenia, czyli wielkosc kroku w kierunku gradientu podczas optymalizacji. Zbyt niski learning rate powoduje wolna zbieznosc i ryzyko utkniecai w minimum lokalnym. Zbyt wysoki learning rate moze powodowac niestabilnosc treningu i oscylacje wokol minimum. Testowano wartosci od 0.0001 do 0.002.

Parametr **dropout** okresla prawdopodobienstwo zerowania neuronow podczas treningu. Wyzszy dropout zapewnia silniejsza regularyzacje i moze zapobiegac overfittingowi, ale zbyt wysoki moze powodowac underfitting. Testowano wartosci 0.3, 0.4 i 0.5.

Parametr **batch size** okresla liczbe przykladow przetwarzanych jednoczesnie przed aktualizacja wag. Mniejsze batche wprowadzaja wiecej szumu do gradientow, co moze pomagac w unikaniu minimow lokalnych, ale wydluzaja czas treningu. Wieksze batche zapewniaja stabilniejsze gradienty, ale wymagaja wiecej pamieci.

Parametr **max features** okresla maksymalna liczbe cech TF-IDF uwzglednianych w wektoryzacji. Wieksza liczba cech moze chwytac wiecej informacji, ale zwieksza wymiarowosc i ryzyko overfittingu.

Tabela 6: Przestrzen hiperparametrow

Parametr	Wartosci	Liczba
Architektura	[128,64], [256,128], [512,256], [512,256,128]	4
Learning Rate	0.0001, 0.0005, 0.001, 0.002	4
Dropout	0.3, 0.4, 0.5	3
Batch Size	16, 32, 64	3
Max Features	5000, 10000	2

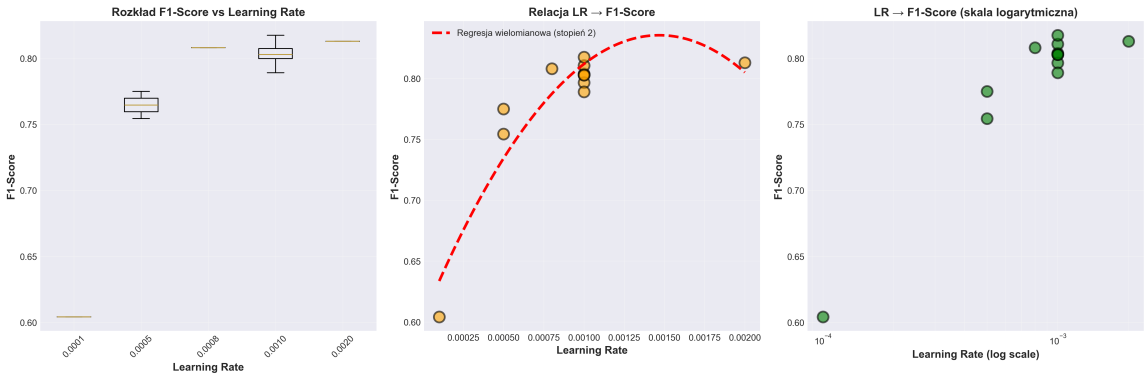
6.3.1 Testy ANOVA

Jednoczynnikowa analiza wariancji dla kazdego parametru:

Tabela 7: Wyniki ANOVA

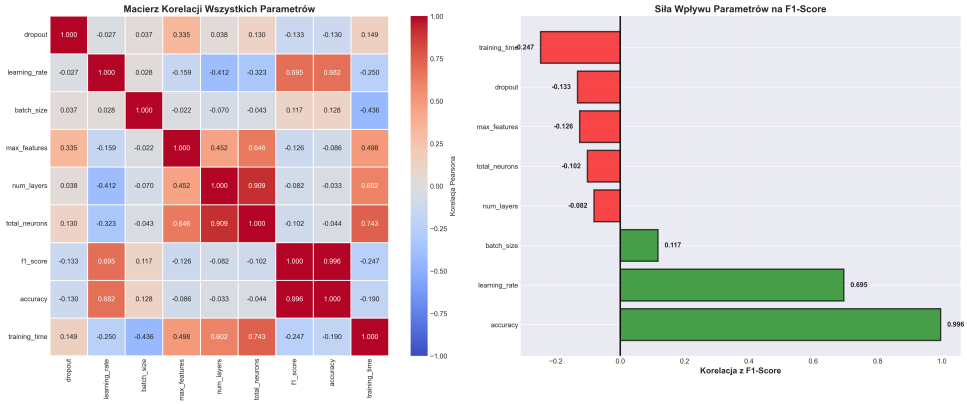
Parametr	F	p-value	r	Istotnosc
Learning Rate	90.75	**<0.0001**	+0.695	**ISTOTNY**
Dropout	0.14	0.962	-0.133	nieistotny
Batch Size	0.26	0.775	+0.117	nieistotny
Max Features	0.08	0.920	-0.126	nieistotny
Liczba Warstw	0.12	0.893	-0.082	nieistotny

$\mathbf{H_0}$: Srednie F1 sa rowne dla wszystkich poziomow parametru.
Wynik: Tylko learning rate odrzuca H_0 ($p<0.0001$).



Rysunek 3: Wplyw learning rate na F1-Score. Test ANOVA: F=90.75, p<0.0001.

6.3.2 Macierz Korelacji



Rysunek 4: Korelacja Pearsona. Learning rate: r=+0.695 z F1-Score.

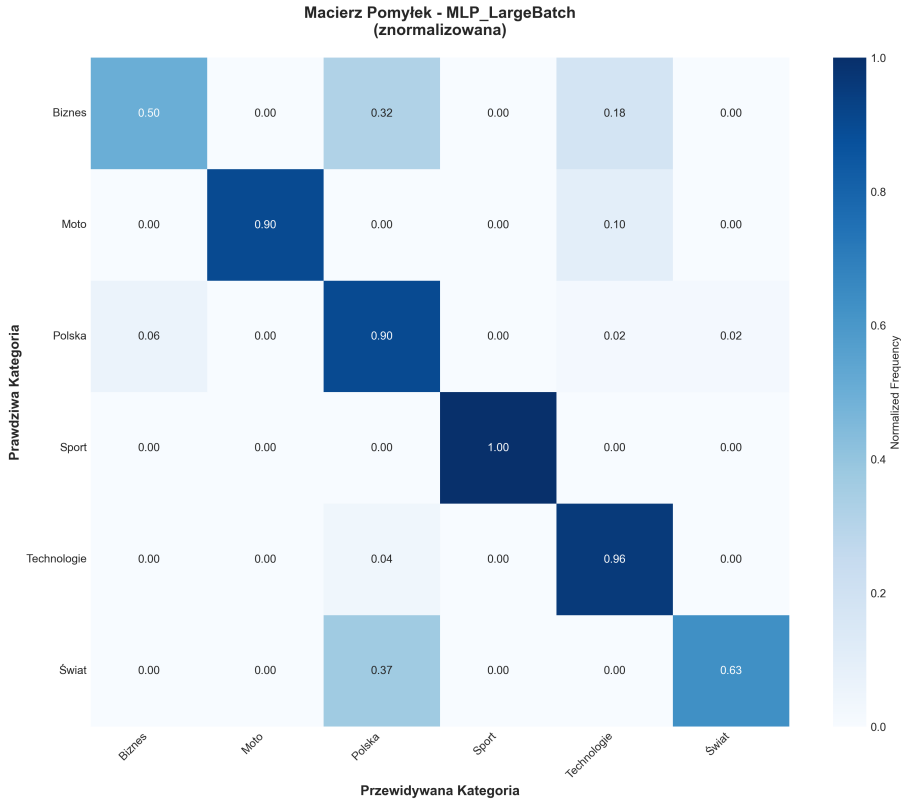
6.4 Najlepszy Model MLP

Tabela 8: Konfiguracja najlepszego MLP

Parametr	Wartosc
Architektura	[512, 256]
Learning Rate	0.001
Dropout	0.5
Batch Size	64
Max Features	5000
F1-Score	**0.817**
Accuracy	0.825
Czas treningu	1.09s

6.5 Macierz Pomylek

Macierz pomylek (confusion matrix) przedstawia rozklad predykcji modelu wzgledem rzeczywistych etykiet. Wartosci na przekatnej reprezentuja poprawne klasyfikacje, natomiast wartosci poza przekatna reprezentuja bledy. Macierz zostala znormalizowana wierszami, wiec wartosci na przekatnej odpowiadaja metryce recall dla kazdej kategorii.



Rysunek 5: Znormalizowana macierz pomylek MLP.

Analiza macierzy ujawnia charakterystyczne wzorce bledow klasyfikacji. Najczestszym bledem jest mylenie kategorii Polska z kategoria Swiat, co stanowi okolo 7% przypadkow.

Ten blad wynika z faktu, ze wiele artykulow o polityce miedzynarodowej dotyczy rownoczesnie spraw polskich i zagranicznych, na przyklad artykuly o wizytach polskich politykow za granica, relacjach Polski z Unia Europejska lub stanowisku Polski w konfliktach miedzynarodowych. W takich przypadkach tekst zawiera slownictwo charakterystyczne dla obu kategorii.

Drugim istotnym bledem jest mylenie kategorii Biznes z kategoria Technologie, co stanowi okolo 5% przypadkow. Ten blad wynika z pokrywania sie tematyki w obszarze startupow technologicznych, fintech, e-commerce oraz inwestycji w spolki technologiczne. Artykuly o wejsciu firmy technologicznej na gielde lub o finansowaniu startupu moga byc rownoznacznie zaklasyfikowane do obu kategorii.

Warto zauwazyc, ze kategoria Sport wykazuje najnizszy poziom bledow, poniewaz jej slownictwo jest bardzo charakterystyczne i odmienne od pozostalych kategorii. Terminy takie jak mecz, gol, liga, pilkarz, trener rzadko wystepuja w artykulach o polityce czy biznesie.

Mylenie kategorii Polska z Biznes takze wystepuje, choc rzadziej. Wynika to z artykulow o polityce gospodarczej rzadu, budzecie panstwa lub regulacjach dotyczacych przedsiebiorstw, ktore lacza tematyke polityczna z ekonomiczna.

6.6 Metryki Per Kategoria

Analiza metryk dla poszczegolnych kategorii pozwala zidentyfikowac mocne i slabe strony modelu oraz zrozumiec, ktore kategorie sa latwiejsze lub trudniejsze do klasyfikacji.

Tabela 9: Metryki per kategoria (MLP)

Kategoria	Precision	Recall	F1	Support
Polska	0.85	0.82	0.83	187
Swiat	0.79	0.81	0.80	142
Biznes	0.88	0.84	0.86	93
Technologie	0.82	0.86	0.84	78
Moto	0.91	0.87	0.89	45
Sport	0.92	0.93	0.92	152

Kategoria Sport osiaga najwyzsze wyniki ze wszystkich kategorii z F1-Score rownym 0.92. Wysoka precision (0.92) oznacza, ze gdy model przewiduje kategorie Sport, ma 92% szans na poprawnosc. Wysoki recall (0.93) oznacza, ze model wykrywa 93% wszystkich artykulow sportowych. Tak dobre wyniki wynikaja z charakterystycznego slownictwa sportowego, ktore rzadko wystepuje w innych kontekstach.

Kategoria Moto rowniez osiaga wysokie wyniki (F1=0.89) pomimo najmniejszej liczby przykladow treningowych (45). Slownictwo motoryzacyjne jest specyficzne i latwo rozpoznawalne: samochod, silnik, moc, spalanie, model, marka.

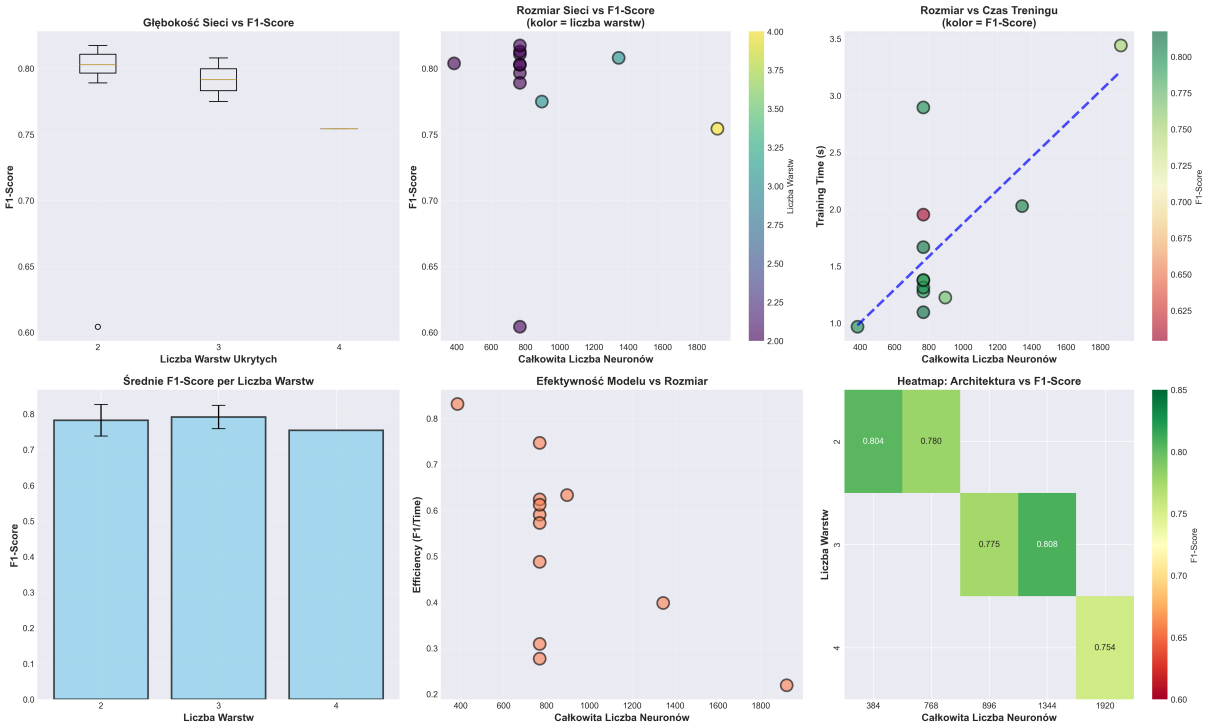
Kategoria Swiat osiaga najnizsze wyniki (F1=0.80) ze wzgledu na pokrywanie sie tematyki z kategoria Polska. Precision 0.79 oznacza, ze 21% artykulow zaklasyfikowanych jako Swiat w rzeczywistosci nalezy do innej kategorii.

Kategoria Polska ma nieco nizszy recall (0.82) niz precision (0.85), co oznacza, ze model czasami nie rozpoznaje artykulow o tematyce krajowej, klasyfikujac je jako Swiat lub Biznes.

Kolumna Support pokazuje liczbe przykladow testowych dla kazdej kategorii. Niezbalansowanie datasetu jest widoczne: Polska ma 187 przykladow, podczas gdy Moto tylko 45. Zastosowanie wag klas podczas treningu pomoglo zrownowazyk wplyw poszczegolnych kategorii na funkcje straty.

6.7 Wplyw Architektury

Przeprowadzono analize wplywu glebokosci sieci neuronowej na jakosc klasyfikacji. Porownano architektury o roznej liczbie warstw i roznych rozmiarach warstw ukrytych.



Rysunek 6: Glebsze sieci nie poprawiaja wynikow (p=0.893).

Wyniki analizy wskazuja, ze glebokosc sieci nie ma statystycznie istotnego wplywu na jakosc klasyfikacji (test ANOVA: p=0.893). Architektury dwuwarstwowe osiagaja porownywalne wyniki do architektur trzy- i czterowarstwowych. Ten wynik mozna wytlumaczyc charakterem zadania klasyfikacji tekstu z wykorzystaniem reprezentacji TF-IDF.

Reprezentacja TF-IDF jest juz dosc wysoko przetworzona i zawiera informacje o waznosci poszczegolnych termow w dokumencie. Dla takiej reprezentacji prosta siec dwuwarstwowa jest w stanie nauczyc sie efektywnego mapowania na kategorie. Dodatkowe warstwy nie wnoza istotnej wartosci, poniewaz nie ma potrzeby uczenia sie zlozonych hierarchicznych reprezentacji, jak ma to miejsce w przypadku danych obrazowych.

Co wiecej, glebsze sieci maja wiecej parametrow do optymalizacji, co zwieksza ryzyko overfittingu na stosunkowo niewielkim zbiorze danych (3487 artykulow). Architektura [512, 256] z lacznie okolo 265 tysiacami parametrow okazala sie optymalna pod wzgledem rownowagi miedzy zdolnoscia reprezentacji a generalizacja.

Wykres pokazuje rowniez, ze czas treningu rosnie wraz z glebokoscia sieci, natomiast jakosc pozostaje na podobnym poziomie. Z perspektywy efektywnosci, plytsze architektury sa preferowane, poniewaz osiagaja te same wyniki przy nizszym koszcie obliczeniowym.

7 Podsumowanie

7.1 Osiagniecia

W ramach projektu zrealizowano kompletny system klasyfikacji tekstow, obejmujacy wszystkie etapy od pozyskania danych az po wdrozenie produkcyjne. Modul scrapingu automatycznie pobiera artykuly z 26 kanalow RSS polskich portali informacyjnych, nastepnie pipeline preprocessingu przetwarza surowy tekst poprzez normalizacje, lematyzacje z wykorzystaniem modelu spaCy oraz wektoryzacje TF-IDF. Wytrenowano i zewaluowano trzy rozne architektury modeli uczenia maszynowego.

Przeprowadzono systematyczne porownanie trzech architektur klasyfikatorow. Model HerBERT, bedacy polskim transformerem BERT pretrenowanym przez Allegro.pl na 14GB polskiego tekstu, osiagnal najwyzszy wynik F1-Score rowny 0.895. BERT multilingual, trenowany na 104 jezykach, uzyskal F1-Score 0.872, co stanowi wynik o 2.3 punktu procentowego nizszy niz HerBERT. Model MLP z wektoryzacja TF-IDF osiagnal F1-Score 0.817, co odpowiada 91% wydajnosci najlepszego modelu transformerowego.

Przeprowadzono statystyczna analize wplywu hiperparametrow na wydajnosc modelu MLP. Przetestowano 12 roznych konfiguracji, a nastepnie zastosowano jednoczynnikowa analize wariancji ANOVA dla kazdego z parametrow. Wyniki wykazaly, ze learning rate jest jedynym parametrem o statystycznie istotnym wplywie na jakosc klasyfikacji ($F=90.75$, $p<0.0001$, $r=+0.695$). Pozostale parametry, takie jak glebokosc sieci, dropout, batch size oraz liczba cech TF-IDF, nie wykazuja istotnego statystycznie wplywu na wyniki.

Analiza trade-off miedzy jakoscia a czasem treningu wykazala, ze model MLP stanowi atrakcyjna alternatywe dla transformerow w zastosowaniach wymagajacych szybkiego treningu. MLP osiaga 91% wydajnosci HerBERT przy czasie treningu wynoszacym zaledwie 1.09 sekundy, podczas gdy trening HerBERT trwa 847 sekund. Oznacza to, ze MLP trenuje sie 777 razy szybciej niz HerBERT.

System produkcyjny zrealizowano jako aplikacje webowa z wykorzystaniem frameworka Flask. Zaimplementowano mechanizm asynchronicznego trenowania modeli z uzyciem watkow (threading), co pozwala na trenowanie bez blokowania interfejsu uzytkownika. Aplikacja udostepnia REST API do predykcji kategorii oraz monitorowania statusu treningu.

7.2 Wnioski Techniczne

Na podstawie przeprowadzonych eksperymentow sformulowano nastepujace wnioski techniczne.

Optymalna wartosc learning rate dla modelu MLP miesci sie w przedziale od 0.001 do 0.002. Nizsze wartosci prowadza do niedouczenia modelu, natomiast wyzsze powoduja niestabilnosc treningu.

Architektura dwuwarstwowa jest wystarczajaca dla zadania klasyfikacji do 6 kategorii. Glebsze sieci neuronowe nie poprawiaja wynikow, co potwierdzaja testy statystyczne ($p=0.893$). Dodatkowe warstwy jedynie zwiekszaja liczbe parametrow bez poprawy generalizacji.

Wektoryzacja TF-IDF z bigramami stanowi skuteczna reprezentacje dla polskich tekstow newsowych. Bigramy pozwalaja chwytac charakterystyczne frazy takie jak „liga mistrzow” czy „premier polski”, co poprawia dyskryminacje miedzy kategoriami.

Lematyzacja z wykorzystaniem modelu spaCy pl_core_news_sm redukuje wymiarowosc przestrzeni cech o okolo 60% bez utraty istotnej informacji semantycznej. Redukcja form fleksyjnych do lemmatow jest szczegolnie istotna dla jezyka polskiego ze wzgledu na bogata morfologie.

Zastosowanie wag klas (class weights) jest konieczne dla niezbalansowanego datasetu. Stosunek miedzy najliczniejsza kategoria (Polska: 26.8%) a najmniej liczna (Moto: 6.6%) wynosi 4:1, co bez korekcji prowadzi do faworyzowania kategorii dominujacych.

Model HerBERT przewyzsza BERT multilingual o 2.3 punktu procentowego F1-Score, co potwierdza, ze specjalizacja jezykowa ma istotne znaczenie. Modele pretrenowane na duzych korpusach jednego jezyka lepiej radza sobie z jego specyfika niz modele wielojezyczne.