Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a umelej inteligencie

Deep model na rozpoznávanie toxických textov pre webový portál

Diplomová práca

Príloha B

Systémová príručka

Vedúci diplomovej práce:

prof. Ing. Kristína Machová, Ph.D.

Diplomant:

Patrik Gecík

Košice 2025

Obsah

1 Funkcia programu

Navrhované riešenie problému automatickej detekcie toxických a netoxických komentárov v textových vstupoch využíva jazyk **Python**. Vývoj a testovanie riešenia bolo realizované prostredníctvom webového vývojového prostredia **Jupyter Notebook** spusteného cez platformu **Google Colab**, čo umožnilo efektívne trénovanie modelov aj s využitím grafických procesorov (GPU). Alternatívne je možné riešenie spúšťať aj v lokálnom prostredí alebo v inom nástroji podporujúcom Jupyter notebooky (napr. VS Code).

Implementácia riešenia je rozdelená do troch logických častí:

1. **Všeobecné predspracovanie dát**

   Táto fáza zahŕňa načítanie datasetov (anglických a slovenských), čistenie textu (odstránenie špeciálnych znakov, zmena na malé písmená, odstránenie diakritiky) a prípravu textových údajov na tokenizáciu a vstup do modelov.

2. **Predspracovanie pre jednotlivé typy modelov**

   Pre model **GRU** sú texty prevedené na sekvencie čísel a doplnené na jednotnú dĺžku. Pre transformerové modely **BERT**, **mT5** a **ByT5** sa využívajú vstavané tokenizéry knižnice `Transformers`, ktoré pripravujú vstupy vo forme `input_ids` a `attention_mask`.

3. **Experimenty**

   Táto časť obsahuje skripty na:

   - **Definíciu architektúry modelu** – výber vhodného modelu a jeho inicializácia,
   - **Trénovanie modelu** – optimalizácia parametrov na trénovacích dátach,
   - **Testovanie modelu** – hodnotenie výkonnosti na testovacích dátach,
   - **Vizualizáciu výsledkov** – zobrazenie metrík ako sú *presnosť*, *recall*, *F1 skóre* a Confusion Matrix.

Cieľom programu je vytvoriť spoľahlivý klasifikátor, ktorý bude schopný efektívne identifikovať toxické správanie v textových komentároch, a to nielen v angličtine, ale aj v slovenskom jazyku.

2 Analýza riešenia

2.1 Čistenie textu a vyváženie dát

Pri predspracovaní dát bolo dôležité odstrániť neúplné záznamy a zabezpečiť jednotné formátovanie textov. Následne boli vypočítané váhy pre jednotlivé triedy, aby sa eliminoval problém nevyváženosti datasetu (väčší počet netoxických ako toxických komentárov).

Odstránenie záznamov s chýbajúcimi údajmi Najskôr boli z datasetov odstránené všetky záznamy s chýbajúcimi údajmi v stĺpcoch `comment_text` a `toxic`:

```
train_texts = train_df.dropna(subset=['comment_text', 'toxic'])
test_texts = test_df.dropna(subset=['comment_text', 'toxic'])
```

Čistenie textov Textové vstupy často obsahujú HTML značky, viaceré medzery alebo zakódované HTML znaky. Na ich odstránenie bola použitá nasledovná funkcia:

```
import re
import html


def clean_text(text):
    text = html.unescape(text)              # Dekódovanie HTML entít
    text = re.sub(r'<[^>]+>', '', text)     # Odstránenie HTML značiek
    text = re.sub(r'\s+', ' ', text)        # Odstránenie viacnásobných medzier
    return text.strip()                     # Odstránenie okrajových medzier
```

Aplikovanie čistenia na texty:

```
train_texts['comment_text'] = train_texts['comment_text'].apply(clean_text)
test_texts['comment_text'] = test_texts['comment_text'].apply(clean_text)
```

Vyváženie tried pomocou váh Pri trénovaní modelov sme narazili na problém nevyváženého počtu vzoriek jednotlivých tried. Tento problém bol riešený výpočtom váh pre funkciu straty (`loss function`) podľa výskytu tried:

```
from sklearn.utils.class_weight import compute_class_weight
import torch
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_labels),
    y=train_labels
)
class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)
```

Váhy boli následne použité pri definovaní stratovej funkcie `CrossEntropyLoss`, ktorá je citlivá na nerovnomerné rozloženie tried:

```
import torch.nn as nn

criterion = nn.CrossEntropyLoss(weight=class_weights)
```

Týmto spôsobom sa dosiahlo, že model neprirađoval vyššiu váhu častejšie sa vyskytujúcim netoxickým komentárom, ale pristupoval spravodlivo ku všetkým kategóriám.

2.2 Vytvorenie modelu

V tejto časti sú predstavené všetky modely použité pri riešení úlohy detekcie toxických komentárov v textových vstupoch. Každý model reprezentuje inú architektúru a prístup k spracovaniu prirodzeného jazyka.

LSTM (Long Short-Term Memory)

LSTM je typ rekurentnej neurónovej siete (RNN), ktorá si dokáže pamätať dlhodobé závislosti v sekvenčných dátach. V rámci riešenia bol LSTM použitý ako základný sekvenčný model pre binárnu klasifikáciu textu. Implementácia modelu prebiehala v knižnici PyTorch.

```
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim, num_layers
        super(LSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers, batch_first=True,
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        embedded = self.embedding(x)
        lstm_out, _ = self.lstm(embedded)
        lstm_out = self.dropout(lstm_out[:, -1, :])
        output = self.fc(lstm_out)
        return output
```

GRU (Gated Recurrent Unit)

GRU je špeciálny typ rekurentnej neurónovej siete (RNN), podobný LSTM, avšak s jednoduchšou architektúrou. Namiesto troch brán (vstupná, výstupná, zabúdacia) používa iba dve (update a reset). GRU bol použitý ako základný sekvenčný model pre porovnanie s LSTM.

```
class GRUModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(GRUModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = self.embedding(x)
        _, h = self.gru(x)
        out = self.fc(h[-1])
        return torch.sigmoid(out)
```

BERT (Bidirectional Encoder Representations from Transformers)

BERT je predtrénovaný transformerový jazykový model vyvinutý spoločnosťou Google. Model pracuje na princípe tzv. maskovaného jazykového modelu, ktorý predpovedá zakryté slová v kontexte viet. V riešení bol použitý na klasifikáciu anglických komentárov.

```
from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2
)
```

mT5 (Multilingual T5)

mT5 je viacjazyčný variant modelu T5 (Text-to-Text Transfer Transformer), ktorý premapúva všetky NLP úlohy na formát text-na-text. V našom prípade bol použitý na klasifikáciu slovenských komentárov. Vstup aj výstup boli zakódované ako text.

```
from transformers import MT5ForConditionalGeneration


model = MT5ForConditionalGeneration.from_pretrained("google/mt5-small")
```

ByT5 (Byte-Level T5)

ByT5 je varianta T5 architektúry, ktorá na rozdiel od mT5 pracuje na úrovni bajtov. Nevyžaduje tokenizáciu slov, čo je výhodné pri práci s jazykmi so špecifickými diakritikami alebo morfologicky bohatými jazykmi ako slovenčina.

```
from transformers import AutoModelForSequenceClassification


model = AutoModelForSequenceClassification.from_pretrained(
    "google/byt5-small", num_labels=2
)
```

Zhrnutie výsledkov Výsledky boli porovnávané medzi jednotlivými modelmi na základe rovnakých metrík. Najvyššiu presnosť na anglických dátach dosiahol model ByT5 (93,4 %), zatiaľ čo na slovenských dátach opäť model ByT5 (77,6 %). GRU a LSTM modely dosiahli nižšiu výkonnosť, avšak s podstatne menšou výpočtovou náročnosťou.

Každý z týchto modelov bol trénovaný na identifikáciu toxického komentára ako binárnu klasifikačnú úlohu, pričom výstupom bola pravdepodobnosť triedy `toxický` alebo `netoxický`.

2.3 Trénovanie modelu

Po definovaní architektúry modelu nasleduje fáza trénovania. Každý model bol trénovaný samostatne na trénovacích dátach so zvolenými hyperparametrami. Cieľom trénovania je minimalizácia chybovej funkcie pomocou optimalizačných algoritmov.

Trénovanie LSTM modelu

Model bol trénovaný pomocou optimalizátora Adam s binárnou stratovou funkciou `BCELoss`. Dátový vstup bol vo forme číselných sekvencií, doplnených na jednotnú dĺžku pomocou `pad_sequences`.

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


for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        inputs, labels = batch
        outputs = model(inputs)
        loss = criterion(outputs.view(-1), labels.float())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Trénovanie GRU modelu

Trénovanie GRU modelu prebieha analogicky k LSTM, pričom hlavný rozdiel spočíva v použití vrstvy `GRU` namiesto `LSTM`. Výstup sa z posledného skrytého stavu prenesie do klasifikačnej vrstvy:

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
criterion = nn.BCELoss()

for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        inputs, labels = batch
        outputs = model(inputs)
        loss = criterion(outputs.view(-1), labels.float())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Trénovanie BERT modelu

BERT model bol trénovaný pomocou knižnice `Transformers` s využitím `Trainer` API. Trénovanie prebiehalo na anglických dátach.

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="./results_bert",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    evaluation_strategy="epoch",
    save_total_limit=2,
    load_best_model_at_end=True
)

trainer = Trainer(
    model=bert_model,
```

```
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset
)

trainer.train()
```

Trénovanie mT5 modelu

Model mT5 bol trénovaný vo formáte text-na-text. Textový vstup bol transformovaný na vetu ako napr. `"toxic classification: ..."`, výstupom bolo `"toxic"` alebo `"non-toxic"`.

```
from transformers import T5Tokenizer, MT5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("google/mt5-small")
model = MT5ForConditionalGeneration.from_pretrained("google/mt5-small")

# Vstupné a výstupné texty
input_texts = ["toxic classification: " + t for t in train_texts]
target_texts = ["toxic" if l == 1 else "non-toxic" for l in train_labels]

# Trénovanie pomocou Trainer API
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=mt5_dataset,
    eval_dataset=mt5_eval_dataset
)
trainer.train()
```

Trénovanie ByT5 modelu

ByT5 model bol použitý priamo na bajtovej úrovni bez tokenizácie slov. Využíval rovnaké nastavenia ako BERT, pričom model bol robustný aj voči chybnému kódovaniu a diakritike v slovenskom jazyku.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("google/byt5-small")
model = AutoModelForSequenceClassification.from_pretrained(
    "google/byt5-small", num_labels=2
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=byt5_dataset,
    eval_dataset=byt5_eval_dataset
)

trainer.train()
```

2.4 Vyhodnotenie modelu

Vyhodnocovanie výkonu modelov bolo realizované pomocou vlastných funkcií, ktoré kombinovali výpočty metrik pomocou knižnice `scikit-learn`, vizualizácie cez `matplotlib` a `seaborn`, ako aj podporu progres barov pomocou `tqdm`.

Použité knižnice Na implementáciu a vizualizáciu hodnotiacich metód boli použité nasledovné knižnice:

- `torch`, `torch.nn`, `torch.optim` – základný framework pre modely a trénovanie,
- `tqdm` – prehľadný progres bar počas trénovania a testovania modelu,
- `sklearn.metrics` – výpočet metrík: `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `confusion_matrix`,
- `matplotlib.pyplot` – vykreslenie vývoja trénovacích metrík,
- `seaborn` – vykreslenie Confusion Matrixu vo forme heatmapy.

Metóda vyhodnocovania modelu Vyhodnocovanie bolo zrealizované funkciou `evaluate_model_with_tqdm()`, ktorá iteratívne prechádzala testovací dataset, aplikovala model, a zhromažďovala predikcie aj skutočné hodnoty.

```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix
)


accuracy = accuracy_score(all_labels, all_preds)
precision = precision_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds)
f1 = f1_score(all_labels, all_preds)
```

Vizualizácia confusion matrixu Na grafickú reprezentáciu Confusion Matrixu sme použili knižnicu `seaborn`. Vykreslenie matice pomohlo pri overení správne klasifikovaných a chybne predikovaných príkladov medzi triedami:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
cm = confusion_matrix(y_test, y_pred)


plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.show()
```

Trénovanie s priebežným hodnotením Počas trénovania boli metriky aktualizované každú epochu, pričom na testovacích dátach sa zároveň sledovala najlepšia dosiahnutá presnosť. Ak nedošlo k zlepšeniu po určitý počet epôch (`patience`), model trénovanie ukončil predčasne – tzv. **early stopping**.

Implementované vyhodnocovanie zabezpečilo objektívne porovnanie výkonnosti rôznych architektúr. Každý model bol vyhodnotený rovnakým spôsobom, čo zaručilo konzistentnosť pri zbere výsledkov pre finálnu analýzu.

3 Popis vývojových prostredí

3.1 Nasadenie na web

Po vytvorení a otestovaní modelu nasledoval krok nasadenia modelu na webové rozhranie. Tento proces umožňuje používateľom nahrať textový vstup alebo komentár, ktorý je následne klasifikovaný modelom ako toxický alebo netoxický.

Použité technológie

- **PHP** – backendový jazyk, ktorý načítava trénovaný model a odosiela požiadavky na inferenciu.
- **Python skript** – samostatne spustiteľný skript, ktorý načíta model a vykoná

predikciu.

- **HTML/CSS/JavaScript** – jednoduché rozhranie pre používateľský vstup.
- **Webhosting/Websupport** – nasadenie webovej aplikácie na server.

Postup nasadenia

1. Export trénovaného modelu do súboru (.pth alebo .bin).
2. Vytvorenie Python skriptu s načítaním modelu a predikciou podľa vstupného textu a jazyka.
3. Vytvorenie PHP skriptu, ktorý prijíma vstup od používateľa a volá Python skript pomocou `shell_exec()`.
4. Príkaz vyzerá nasledovne:

   ```
   $command = "\"C:\\Users\\Dell\\AppData\\Local\\Programs\\Python\\Python312\\
   ```

5. Výstup z Python skriptu sa načíta a zobrazí používateľovi.

Výsledok

Webové rozhranie umožňuje zadať text, ktorý je po kliknutí na tlačidlo „Analyzovať“ odoslaný PHP skriptu. Ten následne spustí Python skript s parametrami, načíta model, vyhodnotí vstup a zobrazí výsledok (napr. „toxický“ alebo „netoxický“).

Ukážka kódu (PHP a Python)

PHP skript:

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $text = escapeshellarg($_POST['text']);
    $lang = $_POST['lang'];
    $command = "\"C:\\Users\\Dell\\AppData\\Local\\Programs\\Python\\Python312\\
```

```
    $result = shell_exec($command . " 2>&1");
}
```

Python skript (predict_bert.py):

```
text = sys.argv[1]
lang = sys.argv[2] if len(sys.argv) > 2 else "sk"
model_path = "best_bert_model_sk.pth" if lang == "sk" else "bert_model.pth"

# Načítanie modelu a tokenizeru
...

# Predikcia a výstup
print(f"\n Výsledok: {labels[pred]} ({probs[pred].item():.2%})")
```

4 Systémové riešenie webového rozhrania

V rámci diplomovej práce bolo vytvorené jednoduché webové rozhranie, ktoré slúži ako nadstavba nad trénovaný model neurónovej siete. Umožňuje používateľovi nahrať vlastný text a získať predikciu toxicity v reálnom čase.

Architektúra systému

Systém je rozdelený na dve hlavné časti:

- **Backend (PHP + Python)** – PHP skript prijíma požiadavku, spúšťa Python skript s modelom a vracia výsledok.
- **Frontend (klient)** – HTML stránka s jednoduchým formulárom na odosielanie textu.

Princíp fungovania

1. Používateľ zadá text do poľa na webstránke.
2. Text sa odošle metódou POST PHP skriptu.
3. PHP skript zostaví príkaz na spustenie Python skriptu s argumentmi.
4. Python skript načíta model a vykoná klasifikáciu.
5. Výsledok sa vráti a zobrazí na stránke.

Výhody riešenia

- Jednoduché nasadenie na klasický webhosting (bez potreby Python servera).
- Prepojenie PHP rozhrania s výkonným modelom v Pythone.
- Možnosť prispôsobenia výsledku a vzhľadu webu.