Effects of different types of data for Sentiment Analysis models for Slovak language

Erik Božík and **Tomáš Nágel**

1 Introduction

The aim of this project was to train and fine-tune models using data from multiple sources and then evaluate their performance. Due to the limited availability of sentiment analysis datasets in Slovak, it was necessary to acquire experimental training data. Three different methods were used to gather data, each covering a distinct scope and employing a unique strategy. The methods used were: Translating dataset from english, Web-scraping reviews and syntheticly generated data. We then trained a simple LSTM model and fine-tuned DistilBERT, both with the focus to fit our computational capacities. The evaluation was conducted on a mix of the datasets.

2 Datasets

2.1 Translated Tweets

2.1.1 Gathering

For the first dataset, we chose to translate a well-established dataset already used for Sentiment Analysis. We selected the Twitter Sentiment Analysis Training Corpus, which contains 1,578,612 entries. Translating such a large dataset required a solution that could run locally, so we utilized an open-source translation model capable of translating to Slovak and small enough that it would not run for days. We created a simple API to translate the data in batches efficiently.

2.1.2 Characteristics

The original dataset contained four columns: ItemID, Sentiment, SentimentSource, and SentimentText. For our purposes, only the Sentiment and SentimentText columns were relevant. We successfully translated 1,138,103 entries, which is 72% of the original dataset. Upon reviewing the texts, we noticed a significant use of slang in the tweets. However, the translation model often left slang phrases untranslated. Phrases like "OMG", certain abbreviations, and character-based emojis (e.g., ":o") were not translated. Consequently, the training data contains a mix of Slovak and non-literally English words and sentences. The dataset covers a broad range of topics from everyday life, but the quality is compromised due to translation inaccuracies and language mix-up.

SentimentText	Sentiment
včera večer som sa dostal sooo chorý dúfam, že môžem urobiť fotenie zajtra a v stredu.	0
cat power's "sea of love" rendering je magické	1

Table 1: Two sampled rows from the Translated Tweets Dataset

This dataset contains a total of 594,008 positive reviews (approximately 52%) and 544,095 negative reviews (approximately 48%).

2.2 Heureka reviews

2.2.1 Gathering

This dataset was collected using a Python scraper built with the Scrapy library, starting from this URL. The scraper navigated through each shop and gathered data from pages like this one. Most reviews included a "+" or "-" before each text, indicating whether the comment was positive or negative. We extracted each text and determined the sentiment based on these indicators. The scraper ran for three days with a one-second delay between requests.

2.2.2 Characteristics

We successfully scraped a total of 3,048,277 entries. However, many reviews lacked marked sentiment or review_text, resulting in 1,810,983 valid entries (approximately 60%). The dataset includes two columns: review_text and sentiment.

review_text	sentiment
Rýchlosť	1
maju v ponuke tovar ktori nie je	0
Dobra spokojnost	1
Nenašiel som žiadnu nevýhodu.	0

Table 2: Four sampled rows from the Heureka Dataset

We identified an incorrect trend in the data: many reviews marked with a minus actually contained non-negative statements such as "no disadvantages" or "none" (as shown in the last row of Table 2). We discovered this issue after training all the models which means that it impacted the model's performance. Despite recognizing the problem, we couldn't find a straightforward way to filter these entries, so we did not retrain.

The dataset contains 1,372,124 positive entries (approximately 75%) and 438,859 negative entries (approximately 25%). This imbalance was likely caused by the fact that the stores are listed from most to least well-known. More popular stores tend to have more positive reviews than negative.

2.3 Synthetic data

2.3.1 Gathering

We tried multiple ways to acquire this data in csv format and we ended up using the openAI api. The prompt used for generation was:

Role	Content
system	You are an assistant that generates unique reviews of products in Slovak with maximum of 25 words and different lengths into csv with columns review_text and sentiment (1-positive, 0-negative)
user	Napíš 130 pozitívnych a negatívnych recenzií striedavo

This request was send to chatGPT 3.5 turbo using the batch functionality (this enabled us to generate more for less money).

2.3.2 Characteristics

The generation of this dataset did not go as expected. Initially, we aimed to generate close to a milion entries which would be comparable to the Heureka or Translated Tweets datasets, but we only managed to generate 171,741 entries. Our efforts were constrained by budget limitations and the API tier. Although we attempted to prompt directly through the chatGPT front end, generating a comparable number of entries this way would have taken days.

Additionally, some outputs were clearly incorrect, either lacking sentiment or containing empty strings, which further reduced the number of usable entries. Nevertheless, similar to the Heureka dataset, we ended up with two columns: review_text and sentiment.

review_text	sentiment
Skvelý pomer cena/výkon, silná doporučená.	1
Bohužiaľ, táto vec nefunguje ako by som čakal.	0
Nevýhodná cena za kvalitu.	0
Odporúčam všetkým!	1

The texts in this dataset are very formal, maintaining a literary tone. Upon reviewing the data, we did not encounter sentences that differed significantly from the given prompt. The dataset consists of 86,438 positive sentiments (approximately 50.3%) and 85,303 negative sentiments (approximately 49.7%).

2.4 Conclusion

We obtained three datasets, each with distinct characteristics. While they share some similarities—for instance, both the Heureka and synthetic datasets focus on reviews—they differ significantly in other aspects. Consequently, we anticipate that each trained model will have unique characteristics.

3 Training

3.1 LSTM

3.1.1 Processing

We used the same processing for all of the datasets. The texts were lowered and everything that did not match the regex

```
[^\w\s]
```

(not a word character or space) was replaced with empty string.

3.1.2 Tokenizer

In order for the neural network to deal with the texts we needed to vectorize it and convert it to sequences.

```
from tensorflow.keras.preprocessing.text
     import Tokenizer
from tensorflow.keras.preprocessing.
    sequence import pad_sequences
max_features = 5000
tokenizer = Tokenizer(num_words=
    max_features, split=' ')
tokenizer.fit_on_texts(data['
    SentimentText'].values)
X = tokenizer.texts_to_sequences(data['
    SentimentText'].values)
X = pad_sequences(X)
Y = tweets['Sentiment'].values
```

Listing 1: Code used for tokenizing

For this purpose, we decided to use the Tokenizer from TensorFlow. Since our goal is to train the model exclusively using TensorFlow, this tokenizer provides seamless integration.

3.1.3 Training

Each dataset was split using the train_test_split method from scikit-learn with test/train ratio 80/20. The random state is fixed to 42 everywhere in this project. The model consists of four layers. We selected these layers and their hyperparameters based on the resources listed at the end of this project, where similar issues were addressed. It's crucial to get this right from the start, as each experiment demands significant time and computational resources. A poorly trained model would hinder our ability to properly evaluate the results.

```
from tensorflow.keras.layers import
    Embedding, LSTM, Dense,
    SpatialDropout1D
from tensorflow.keras.models import
    Sequential
embed_dim = 128
lstm_out = 196
model = Sequential()
model.add(Embedding(max_features,
    embed_dim, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2,
    recurrent_dropout=0.2))
model.add(Dense(2, activation='softmax')
    )
model.compile(loss='
    sparse_categorical_crossentropy',
    optimizer='adam', metrics=['accuracy
    '])
```

Listing 2: Composing the layers

The code above was reused for every dataset. We trained the model consistently using 5 epochs, with batch size as the only variable parameter. Since every dataset had different shape, we adjusted the batch size to balance performance and training time. This adjustment was guided by observing the accuracy during training: if the accuracy dropped significantly with a larger batch size, we reduced it. Similarly, if the accuracy remained comparable to smaller batch sizes, we retained the larger size to leverage the GPU and achieve faster training times.

```
from tensorflow.keras.callbacks import
    EarlyStopping
batch_size = 32
epochs = 5

early_stopping = EarlyStopping(monitor='
    val_loss', patience=3, verbose=1,
    restore_best_weights=True)

model.fit(X_train, Y_train, epochs=
    epochs, batch_size=batch_size,
    validation_data=(X_test, Y_test),
    callb
```

Listing 3: Fitting the model

3.1.4 Training results

Training dataset	Batch size	Accuracy	Loss
Translated tweets	32	0.78	0.45
Heureka reviews	256	0.96	0.10
GPT 3.5 generated data	32	0.96	0.10

Table 5: LSTM evaluation on their test sets

Training the model on the translated tweets dataset proved to be more challenging compared to the other two datasets. It seems that the translation quality and language imbalance might have caused problems with fitting the data. Overall, we can say that each model was trained successfully with sufficient accuracy.

3.2 DistilBERT

3.2.1 Processing

Each dataset has been processed the same way as with LSTM. The datasets (a DataFrame) are split into training and testing sets. This is important to evaluate the model's performance on unseen data.

3.2.2 Tokenizer

The tokenizer is responsible for converting raw text into a format that the model can understand, specifically into token IDs and attention masks. The tokenized dataset is further formatted for PyTorch, setting up the necessary columns like input IDs, attention masks, and labels, ensuring that the data is ready for training with the PyTorch framework.

```
from transformers import
    DistilBertTokenizer,
    DistilBertForSequenceClassification
```

```
tokenizer = DistilBertTokenizerFast.
    from_pretrained('distilbert-base-
    multilingual-cased')
def df_to_dataset(df, tokenizer):
    dataset = Dataset.from_pandas(df)
    def tokenize_function(examples):
        return tokenizer(examples['text'
            ], padding="max_length",
            truncation=True, max_length
            =512)
    return dataset.map(tokenize_function
        , batched=True, num_proc=4)
train_dataset = df_to_dataset(train_df,
    tokenizer)
test_dataset = df_to_dataset(test_df,
    tokenizer)
tokenized_dataset = DatasetDict({
    'train': train_dataset,
    'test': test_dataset
})
tokenized_dataset.set_format('torch',
    columns=['input_ids', '
    attention_mask', 'labels'])
```

Listing 4: Code used for tokenizing

3.2.3 Metrics

A function is defined to compute evaluation metrics like accuracy, precision, recall, and F1 score. These metrics are used to assess the performance of the model during and after training.

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis
        =-1)
    precision, recall, f1, _ =
        precision_recall_fscore_support(
        labels, predictions, average='
        weighted')
    acc = accuracy_score(labels,
        predictions)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
}
```

Listing 5: Code for computing metrics

3.2.4 Training

The model is configured for sequence classification with two labels (binary classification). The model is then moved to the selected device to ensure that all computations happen on the correct hardware. Training arguments are specified, including the output directory for results, number of training epochs, batch sizes for training and evaluation, warmup steps, weight decay, logging configuration, and whether to use 16-bit floating point precision if available. These settings are crucial for controlling the training process and optimizing performance. The Hugging Face Trainer class is instantiated with the model, training arguments, datasets, and the metrics function. The Trainer class handles the training loop, evaluation, and other tasks, simplifying the training process. The model is trained using the Trainer's train method. After training, the model is saved, and the evaluation on the test set is performed to obtain validation results, including accuracy and other metrics.

```
batch_size = 16
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=
        batch_size,
    per_device_eval_batch_size=
        batch_size,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=50,
    save_strategy="epoch",
    evaluation_strategy="epoch",
    load_best_model_at_end=True,
    fp16=torch.cuda.is_available(),
    dataloader_num_workers=4
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["
        train"],
    eval_dataset=tokenized_dataset["test
        "],
    compute_metrics=compute_metrics
)
trainer.train()
trainer.save_model("./best_model")
results = trainer.evaluate()
print(f"Validation␣Results:␣{results}")
```

Listing 6: Code for setting up and training the model

3.2.5 Training results

Training dataset	Batch size	Accuracy	Loss
Translated tweets	16	0.80	0.44
Heureka reviews	16	0.97	0.114
GPT 3.5 generated data	16	0.99	0.044

Table 6: DistilBERT evaluation on their test sets

4 Evaluation

We created a combined dataset by sampling 50,000 rows from each of the original datasets. Mixed dataset contained 150,000 rows with 59.4% of positive and 40.6% of negative sentiments. We then evaluated all our trained and fine-tuned models on this mixed dataset.

Model	Train data	Accuracy	Precision	F1
LSTM	Tweets	0.78	0.78	0.78
DistilBert	Tweets	0.85	0.85	0.85
LSTM	Synthetic	0.76	0.79	0.76
DistilBert	Synthetic	0.80	0.80	0.80
LSTM	Heureka	0.82	0.82	0.82
DistilBert	Heureka	0.83	0.83	0.83

Table 7: Evaluation of models on mixed dataset

As expected, the fine-tuned DistilBERT outperformed the LSTM models. However, the margin of improvement is only a few percentage points across each metric. Therefore, the difference is not substantial. The precision ranges from 78% to 83%, which we consider highly satisfactory for our purposes.

5 Future Work

The primary area for improvement lies in the training data itself. As mentioned earlier, there are multiple issues with the training sets that cannot be easily resolved. Therefore, training the models on datasets where these issues have been addressed could be a task for future work. Additionally, there is potential for improvement in the evaluation dataset. Since we did not have completely separate data for evaluation, the evaluation set included one-third of the data used in training for each model.

6 Resources

1. LSTM Sentiment Analysis | Keras
2. Sentiment Analysis with LSTM
3. Sentiment Analysis Using LSTM Networks: A Deep Dive into Textual Data
4. Model Card for DistilBERT base multilingual cased (HuggingFace)
5. Text Classification using BERT Models (HuggingFace)
6. Fine-tune a pretrained model (HuggingFace)