Statistical Natural Language Processing D
ELEC-E5550

Multilingual Toxicity Detection

Project Report

An Binh Bui **100598496** (anbinh.bui@aalto.fi)
Irmuun Tuguldur **100564190** (irmuun.tuguldur@aalto.fi)
Pham Quang Bach **100553130** (bach.pham@aalto.fi)
Ojaswi Tyagi **100809169** (ojaswi.tyagi@aalto.fi)

April 2026

1 Introduction

The prevalence of large-scale online communities on entertainment and social media platforms has inadvertently created grounds for harmful content to spread at an unprecedented speed. This is especially the case when internet users are increasingly engrossed in their online identities. Toxic behaviors such as bullying, harassment, hate speech, and death threats can create hostile environments that compromise the integrity of online spaces, as well as negatively impact the user's mental health and life beyond the virtual world. Many platforms have struggled with effective content moderation, often resorting to restricting or disabling user comments, which greatly hinders platform's usability. Thus, toxicity detection to filter out hateful and damaging contents, such as hate speech and harassment, is more crucial and relevant than ever before.

In this report, we aim to determine computationally efficient and effective methods that would be useful in tackling multilingual toxicity detection. To do so, we have implemented and assessed various machine learning and deep learning methods that have been created to address this. We have used Logistic Regression, Support Vector Machine (SVM), Convolutional Neural Network (CNN), multilingual Bidirectional Encoder Representations from Transformers (mBERT), and the cross lingual Robustly Optimised BERT Approach (XLM-RoBERTa) to classify short texts from the toxicity detection dataset provided by the course's competition. The competition dataset comprises of training, development, and test sets. The very large training set was entirely in English, while the development and test set contained English, German, and Finnish.

2 Methods

We use five separate models with three differing levels of complexity in order to assess each architecture's ability to adapt to the dataset. Simpler methods such as SVM and Logistic Regression are used to create a baseline. A Convolutional Neural Network was used to bridge the gap between the traditional statistical methods and modern deep learning methods. Finally we evaluate two complex transformer based models: XLM-RoBERTa and mBERT.

2.1 Baseline models: Logistic Regression and Support Vector Machines

The Logistic Regression model is a supervised machine learning model that predicts the probability of a specific outcome occurring as a linear combination of one or more independent variables. This maps directly to a binary regression model where the labels are 0 or 1 by choosing a cutoff value, normally 0.5 [1]. Support Vector Machines (SVM) are supervised machine learning algorithms that classify data by finding an optimal line or hyperplane which maximizes the distance between each class. Because of its relatively simple construction and ability to perform well in higher dimensional spaces, SVMs are commonly used within classification problems. In addition to efficiently performing linear classification, SVMs can also perform non-linear classification by using the kernel trick. Herein, a kernel function is used to represent the data by their pairwise kernel distance, which maps them to a higher dimensional space [2]. However, non-linear classification typically requires training time that scales at least quadratically with the sample size, making it impractical for our dataset of nearly 100,000 samples. Therefore, for this comparison, we will use a simpler linear SVM instead. These simple linear models are light weight and can be implemented without large or complex libraries. Because of this reason, we have chosen them as a baseline to compare the other models with.

2.2 CNNs

Convolutional Neural Networks (CNNs) are deep learning architecture designed to take advantage of successive convolution and pooling layers to learn increasingly complex patterns without the need of manual feature engineering. CNNs operate by applying learnable filters across local regions of the input to extract hierarchical features, with pooling layers to progressively reduce spatial resolution while preserving the most important information [3]. Originally popularized for computer vision, CNNs in the context of Natural Language Processing (NLP) treat text as a one dimensional spatial signal, utilizing sliding convolutional kernels to identify local dependencies between adjacent tokens [4]. CNNs were chosen as they provide a decent "middleground" between the simpler SVM or Logistic Regression models and complex transformer based models like mBert and XLM-RoBERTa.

2.3 Transformer based models

Transformer based models such as Bidirectional Encoder Representations from Transformers (BERT) are constructed for deep bi-directional pre-training from unlabeled entries by taking in both context direction in every

layer [5]. BERT generates a unique vector for every word in a sentence based on the entire surrounding context, i.e. words after and before it [6]. Therefore, BERT preserves the contexts in which the each word exists, allowing for more accurate and nuanced classification. The BERT model is based on Transformer model (included in figure 1) and was pre-trained with Masked LM, where a portion of the input was masked then predicted, and Next Sentence Prediction (NSP) [5]. The training data for this was large dataset consisting of the BookCorpus with 800 million words and the English Wikipedia article texts with 2500 million words [5]. For fine-tuning the BERT model, input data corresponding to the task at hand is used, then the parameters are tuned accordingly [5]. The input example for the model is shown below in figure 2. In figure 3, we can see that the pre-training and fine-tuning for the BERT model share the same structure apart from the output layer [5].

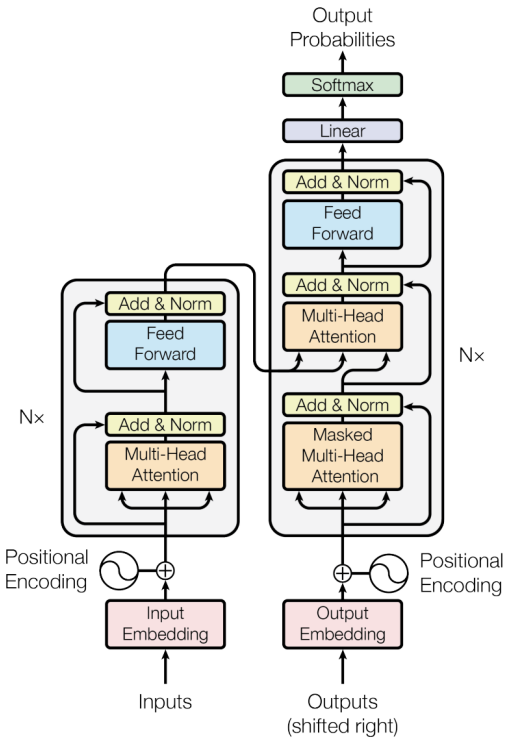

Figure 1: The Transformer model's architecture [7]

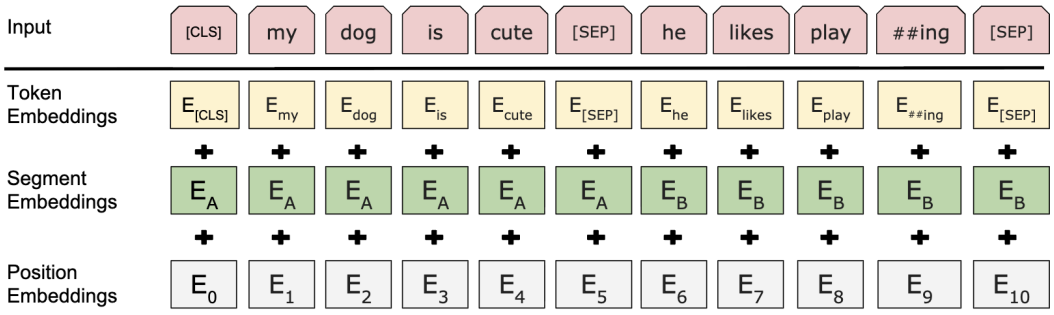

Figure 2: BERT input example [5]

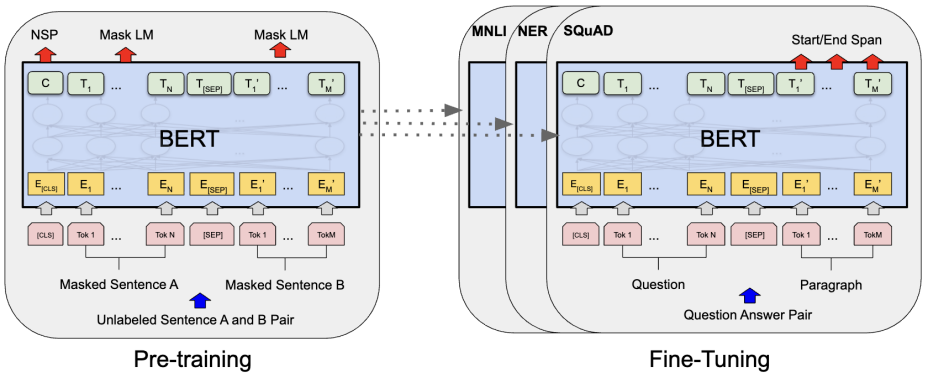

Figure 3: Procedures for pre-training and fine-tuning of BERT [5]

Other transformer based models such as mBERT and XLM-RoBERTa marked a major shift. mBERT is based on the BERT architecture and has been trained on Wikipedia articles from 104 languages with the highest number of entries.[8]. RoBERTa is a BERT model with an improved approach for pretraining, with the usage of dynamic masking, the removal of NSP loss, large-batch training, and a larger set of Byte-Pair Encoding (BPE) [9]. By leveraging bidirectional self-attention, these models captured contextual relationships more effectively and generalized across multiple languages within a single shared architecture, notably outperforming other models we have tried.

XLM-RoBERTa is a model based on Robustly Optimized BERT Approach (RoBERTa). However, unlike RoBERTa, which is only trained on english datasets, XLM-RoBERTa has been trained on a large multilingual dataset. [10].

3 Experiments

With the given course competition dataset, we use the training set to train the models and the development set to evaluate, compare, and tune the models. The test set was used for generating predictions, which were then submitted on the competition's Codabench page for the ranking. We also used the outputted language-specific results from the page for model comparison.

The models used in this project are - logistic regression (baseline), Support vector machine (baseline), Convolutional Neural Network, mBERT, and XLM-RoBERTa.

For the baseline Logistic Regression and SVM models, we chose to use a character level term frequency–inverse document frequency (TF-IDF) vectorizer. This was chosen to specifically capture sub-word patterns and cross lingual features that would otherwise be lost in word level representations.

For the CNN, mBERT, and XLM-RoBERTa models, input sequences were standardized to the length of 128 tokens via padding and truncation. The specific tokenization method that was utilized is that of XLM-RoBERTA model and was imported from the open source Transformer library by Hugging Face. By using a shared tokenizer and vocabulary for the CNN, we ensured that the performance differences were a result of architectural capabilities rather than discrepancies in data preparation.

The CNN was trained using multiple kernel sizes operating in parallel to detect local n-gram features. To match the transformer models, the CNN embedding layer was scaled to the full tokenizer vocabulary, comprising of over 100 languages. This resulted in a sizeable increase in the model's parameter count.

mBERT and XLM-RoBERTa were pre-trained models imported from the same Hugging Face library, while the baseline and CNNs were trained from scratch. All models were implemented with PyTorch.

For the initial comparison, we looked at the overall accuracy and F1 score of the different models when evaluated against the development dataset. Accuracy was used as a straightforward indicator, while F1 score was used to further assess how well the model fares with regards to false positives and false negatives. The calculation of these scores was implemented with the help of the f1_score and accuracy_score function from the scikit-learn library. Due to the substantial computational cost of the transformer based models, with the mBERT in particular needing 104 minutes to complete an epoch of training, we decided to compare the evaluation of the development set only after one epoch for all the models for fairness of comparison. Then, we have also compared the models using the language-specific accuracy and F1 score against the test set's generated predictions.

4 Results

Table 1 contains the evaluation results for all models after a single training epoch on the development set. XLM-RoBERTa emerged as the best performing architecture, achieving the highest recorded accuracy (0.911) and F1 score (0.8996), followed closely by mBERT. The remaining models achieved similar results with a special note on the CNN, whose performance measures quite closely to the transformer based models. While the CNN and SVM achieved similar accuracy levels, the CNN's slightly superior F1 score (0.842 vs 0.793) suggests its better able to generalize across potentially imbalanced classes. As expected, the Logistic regression baseline achieved the lowest accuracy and F1, hinting to the need for more complex models.

Model	**Accuracy**	**F1**
XLM-RoBERTa	**0.910988**	**0.899642**
mBERT	0.896818	0.885428
CNN	0.857878	0.842604
SVM	0.856818	0.793261
Logistic Regression	0.836212	0.769115

Table 1: Evaluation results after 1 epoch against the development set, with best results highlighted in bold

Table 2 shows the valuation results against the test set for all the models. XLM-RoBERTa maintained its status as the best performing model, achieving the highest results in nearly all metrics. The only exception to this being the German F1 score, where mBERT performs slightly better.

We observe a visible gap in performance between the pretrained models and those trained from scratch. This gap is especially clear in the language specific metrics, particularly for Finnish, where the transformer based models' ability to leverage pre-existing cross-lingual representations allow them to adapt much better to the complex linguistic structures that the simpler models struggle to identify.

We do acknowledge that a possible reason for the gap in performance could be the limited 1 epoch training limit. As the transformers are pretrained they adapt much better to the dataset than the others. Pretrained versions of CNNs, SVMs, and Logistic Regression models tailored for text classification were not found during our research.

The models' required time to complete training vary greatly, with mBERT in particular needing 104 minutes to complete a single epoch of training, while CNN was much more efficient and managed to complete 10 epochs in 175 minutes. The best performing model, XLM-RoBERTa was relatively efficient and only took 90 minutes to train with 3 epochs.

Model	**Multilingual accuracy**	**Multilingual F1**	**English accuracy**	**English F1**	**German accuracy**	**German F1**	**Finnish accuracy**	**Finnish F1**
XLM-RoBERTa	**0.83**	**0.78**	**0.94**	**0.94**	**0.76**	0.46	**0.59**	**0.68**
mBERT	0.81	0.75	0.93	0.93	0.75	**0.48**	0.42	0.48
CNN	0.69	0.66	0.90	0.90	0.51	0.39	0.43	0.52
SVM	0.74	0.68	0.90	0.90	0.64	0.38	0.31	0.31
Logistic Regression	0.72	0.67	0.88	0.88	0.61	0.42	0.41	0.49

Table 2: Evaluation results against the test set, with best results highlighted in bold

5 Conclusions & Discussion

There is immense potential in these models. XLM-RoBERTa has demonstrated satisfactory performance in detecting multi-lingual hate speech (with the accuracy for the German data being close to 80%), despite being trained only on English data in this project. This suggests that training with non-English data would significantly improve classification performance . While the transformer-based models consistently performed the best, mBERT in particular may not be worth the computation cost, with its training taking an exceedingly long time to complete. Our results indicate that there seems to be a trade-off between performance metrics and the computational resources required to run the models, with higher-performing models typically demanding greater computational resources and runtime.

The CNN, while having decent performance in the overall results, showed a noticeable decline in performance on the test set - particularly lacking success on the language specific metrics. This highlights its susceptibility to highly imbalanced datasets in comparison to the transformer models, even when the same tokenizer function was applied.

It should be noted that while this does demonstrate the potential of these models to function as effective classifiers, there is a lot of work to be done before they are actually usable. For one, the dataset used in this report is relatively simple, with short sentences and minimal use of slang, which is rare in for a real life scenario.

The results demonstrate the potential for these models to be used as toxicity classifiers. However more comprehensive studies with realistic datasets need to be conducted for further development.

6 Division of Labour

Below is how our group's division of labour was done:

1. Irmuun Tuguldur: CNN model, abstract, literature review, project report
2. Pham Quang Bach: SVM model, abstract, literature review, project report
3. Ojaswi Tyagi: XLM-RoBERTa model, logistic regression, literature review, project report
4. An Binh Bui: mBERT model, abstract, literature review, project report

7 References

[1] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013.

[2] M. Awad and R. Khanna, "Support vector machines for classification," in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, 2015, pp. 39–66, ISBN: 978-1-4302-5990-9. DOI: 10.1007/978-1-4302-5990-9_3. [Online]. Available: https://doi.org/10.1007/978-1-4302-5990-9_3.

[3] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed., [Online]. Available, Cambridge, MA: MIT Press, 1995, pp. 255–258. [Online]. Available: https://archive.nyu.edu/handle/2451/14951.

[4] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014. [Online]. Available: https://arxiv.org/abs/1408.5882.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805v2 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805v2.

[6] M. Munikar, S. Shakya, and A. Shrestha, *Fine-grained sentiment classification using bert*, 2019. DOI: 10.1109/AITB48515.2019.8947435.

[7] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon et al., Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] J. Devlin, *Bert/multilingual.md at a9ba4b8d7704c1ae18d1b28c56c0430d41407eb1 · google-research/bert*, 2018. [Online]. Available: https://github.com/google-research/bert/blob/a9ba4b8d7704c1ae18d1b28c56c0430d41407eb1/multilingual.md.

[9] Y. Liu et al., *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: 1907.11692 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1907.11692.

[10] A. Conneau et al., *Unsupervised cross-lingual representation learning at scale*, 2020. arXiv: 1911.02116 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1911.02116.

8 Appendix

Below is the code used to implement the models

installing the libraries

```
pip install --upgrade transformers datasets torch scikit-learn
pip install 'accelerate>=1.1.0'
```

importing libraries and functions

```
import pandas as pd
import csv
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from torch.utils.data import Dataset, DataLoader
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
from transformers.utils.notebook import NotebookProgressCallback
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, f1_score
```

opening the datasets:

```
test_df = pd.read_csv('test.tsv', sep='\t', header=0, quoting=3)
train_df = pd.read_csv('train.tsv', sep='\t', header=0, quoting=3)
dev_df = pd.read_csv('dev.tsv', sep='\t', header=0, quoting=3)
test_df.head()
train_df.head()
dev_df.head()
```

Tokenizer for Logistic Regression and LinearSVC model

```
char_vectorizer = TfidfVectorizer(analyzer='char', ngram_range=(2, 5), max_features=10000)

X_train = char_vectorizer.fit_transform(train_df['text'])
y_train = train_df['label']

X_dev = char_vectorizer.transform(dev_df['text'])
y_dev = dev_df['label']
```

Logistic Regression model

```
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)

# Evaluate on dev set
y_dev_pred = model.predict(X_dev)
print(classification_report(y_dev, y_dev_pred))
```

LinearSVC model

```
model = LinearSVC(verbose=True, class_weight='balanced')
model.fit(X_train, y_train)

y_dev_pred = model.predict(X_dev)
print(classification_report(y_dev, y_dev_pred))
```

Tokenize function for CNN, mBERT, XLM-RoBERTa models

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
def tokenize_function(input):
    return tokenizer(input['text'], padding="max_length", truncation=True, max_length=128)
```

evaluation function

```
def compute_metrics(eval):
    logits, labels = eval
    predictions = logits.argmax(axis= -1)
    return {
        "accuracy": accuracy_score(labels, predictions),
        "f1": f1_score(labels, predictions, average='macro')
    }
```

Processing dataset for the CNN, mBERT and XLM-RoBERTa models

```
#dataset type conversion so it works with the Hugging Face's pretrained model
test_dataset = Dataset.from_pandas(test_df)
train_dataset = Dataset.from_pandas(train_df[['text', 'label']])
dev_dataset = Dataset.from_pandas(dev_df[['text', 'label']])

#tokenising the dataset
tokenized_test = test_dataset.map(tokenize_function, batched=True)
tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_dev = dev_dataset.map(tokenize_function, batched=True)
```

Implementing the CNN model

```
model_name = "xlm-roberta-base"
# Convert to pytorch since the initial tokenization is suited for huggingface models
tokenized_train.set_format(type='torch', columns=['input_ids', 'label'])
tokenized_dev.set_format(type='torch', columns=['input_ids', 'label'])
tokenized_test.set_format(type='torch', columns=['input_ids'])

# Create DataLoaders. Batch size 32 is a good balance for CNNs
train_loader = DataLoader(tokenized_train, batch_size=32, shuffle=True)
dev_loader = DataLoader(tokenized_dev, batch_size=32)
test_loader = DataLoader(tokenized_test, batch_size=32)

class CNNClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, n_filters, filter_sizes, output_dim, dropout = 0.3):
        super(CNNClassifier, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_dim)

        self.convs = nn.ModuleList([
            nn.Conv1d(in_channels = embed_dim, out_channels = n_filters, kernel_size = fs)
            for fs in filter_sizes
        ])
```

```
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, input_ids):
        embedded = self.embedding(input_ids)
        embedded = embedded.permute(0, 2, 1)

        conved = [F.relu(conv(embedded)) for conv in self.convs]

        pooled = [F.max_pool1d(conv_out, conv_out.shape[2]).squeeze(2) for conv_out in conved]

        cat = self.dropout(torch.cat(pooled, dim = 1))

        return self.fc(cat)
```

Evaluation loop of the CNN model

```
vocab_size = tokenizer.vocab_size
# used the full tokenizer vocabulary, which ended up slowing things down considerably. To hasten the proce
embed_dim = 128 # Could change to 768 to make the comparison to the transformers more fair, but decided on
n_filters = 100 # to ensure enough chances to detect
filter_sizes = [2, 4, 7] # modified the standard 3, 4, 5 to better match the longer dataset. Toxicity also
output_dim = 2 # toxic or not so dim of 2

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

cnn_model = CNNClassifier(vocab_size, embed_dim, n_filters, filter_sizes, output_dim)
cnn_model.to(device)

optimizer = torch.optim.Adam(cnn_model.parameters(), lr = 0.001)
criterion = nn.CrossEntropyLoss()
```

Training loop for CNN and save Model

```
# num_epochs = 10
# for epoch in range(num_epochs):
    cnn_model.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        labels = batch['label'].to(device)

        outputs = cnn_model(input_ids)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    cnn_model.eval()
    all_logits = []
    all_labels = []

    with torch.no_grad():
        for batch in dev_loader:
            input_ids = batch['input_ids'].to(device)
            labels = batch['label'].to(device)
```

```
            logits = cnn_model(input_ids)

            all_logits.append(logits.cpu().numpy())
            all_labels.append(labels.cpu().numpy())

    all_logits = np.concatenate(all_logits, axis=0)
    all_labels = np.concatenate(all_labels, axis=0)

    results = compute_metrics((all_logits, all_labels))
# print(f"Epoch {epoch+1}: Accuracy = {results['accuracy']}, F1 = {results['f1']}")

print(f"Accuracy: {results['accuracy']}")
print(f"Macro F1: {results['f1']}")

save_path = "cnn_model.pt"

# Save weights and hyperparameters
torch.save({
    'model_state_dict': cnn_model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'vocab_size': vocab_size,
    'embed_dim': embed_dim,
    'n_filters': n_filters,
    'filter_sizes': filter_sizes,
    'output_dim': output_dim
}, save_path)

print(f"Model saved successfully to {save_path}")
```

Implementing the xlm-roberta model, and saving the trained model after implementation

```
trainMode = False

if trainMode:
    model_name = "bert-base-multilingual-cased"
else:
    model_name = "saved_mBERT"

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

model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels= 2)
model.to(device)

training_args = TrainingArguments(
    output_dir= "./results",
    eval_strategy="epoch",
    per_device_train_batch_size= 16,
    per_device_eval_batch_size= 16,
    num_train_epochs= 1,
    weight_decay= 0.01,
    report_to= "none",
    save_strategy= "no"
)

trainer = Trainer(
    model= model,
```

```
    args= training_args,
    train_dataset= tokenized_train,
    eval_dataset= tokenized_dev,
    compute_metrics= compute_metrics,
)

if trainMode:
    trainer.train()

if not trainMode and NotebookProgressCallback in [type(cb) for cb in trainer.callback_handler.callbacks]:
    trainer.remove_callback(NotebookProgressCallback)

results = trainer.evaluate()
print("Results:", results)

if trainMode:
    trainer.save_model("saved_mBERT")
    tokenizer.save_pretrained("saved_mBERT")

#generating the prediction dataset
test_preds = trainer.predict(tokenized_test)
test_preds = np.argmax(test_preds.predictions, axis=-1)
test_df['predicted'] = test_preds
test_df[['id', 'predicted']].to_csv('test_predictions.tsv', sep='\t', index=False)
```

implementing the mBERT model, and saving the trained model after implementation

```
TRAIN_MODEL = False # Set this to True to train from scratch, False to load the saved model
if TRAIN_MODEL:
    model_name = "xlm-roberta-base"
else:
    model_name = "my_saved_model"

# Prepare model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
model.to(device)

training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    report_to="none",
    save_strategy="no" # Keep this to "no" to prevent filling up the disk again
)

# Even if we don't train, we can use the Trainer for evaluation and prediction
trainer = Trainer(
```

```
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_dev,
    compute_metrics=compute_metrics,
)

if TRAIN_MODEL:
    trainer.train()
else:
    print("Skipping training, using the loaded model.")

# After training, you can evaluate its performance on the development split
# and make predictions on the test set.

# If we skipped training, the notebook progress callback will crash on evaluate.
# So we remove it to prevent the error.
if not TRAIN_MODEL and NotebookProgressCallback in [type(cb) for cb in trainer.callback_handler.callbacks]
    trainer.remove_callback(NotebookProgressCallback)

trainer.evaluate()

# Prepare test data
test_dataset = Dataset.from_pandas(test_df[['id', 'text']])
tokenized_test = test_dataset.map(tokenize_function, batched=True)

# Generate predictions
test_predictions = trainer.predict(tokenized_test)
test_preds = np.argmax(test_predictions.predictions, axis=-1)

# Ensure the column is named 'predicted' and saved as a .tsv file
test_df['predicted'] = test_preds
test_df[['id', 'predicted']].to_csv('test_predictions.tsv', sep='\t', index=False)
print("Saved predictions to test_predictions.tsv that has", len(test_df), "samples")

# Save the model only if we trained it
if TRAIN_MODEL:
    trainer.save_model("my_saved_model")
    # Save the tokenizer too, if needed
    tokenizer.save_pretrained("my_saved_model")
    print("Model saved to my_saved_model")
else:
    print("Skipping save because training was not performed.")
```