

Project Feasibility Twitter (X) Sentiment Analysis for Amtagio

Individual Project

Submitted by: Caroline Ziegler (22108564)

Submitted to: Dr. Andreas Robotis

Program: Global Business Germany (EGB4)

Module Code: MT414

Submission Date: 08.04.2024

Project Introduction

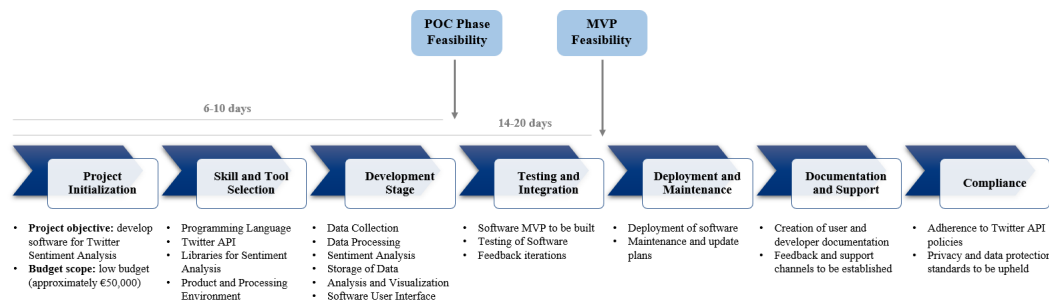
In the dynamic world of digital marketing, the ability to analyze public sentiment in real-time is not just an advantage but a necessity. Moreover, leveraging influencer partnerships has become a pivotal strategy for brands seeking to enhance their visibility and engagement. Amtagio, a start-up that has recently ventured into the travel accessories market with the innovative Amtagio digital tag, is placing a significant emphasis on influencer-led social media marketing, particularly on Twitter. The challenge, however, is their current inability to analyze and understand the real-time public reaction to these influencer campaigns, a crucial factor in evaluating their effectiveness and optimizing future strategies. The young entrepreneur behind Amtagio is driven by the ambition to develop a software tool capable of performing real-time sentiment analysis on Twitter. This tool aims to provide immediate insights into the public's perception of the influencer marketing efforts, thereby gauging the impact on the Amtagio digital tag's market presence. With a budget of €50,000, the entrepreneur is faced with several key considerations:

- **Budget Adequacy:** Is the allocated budget of €50,000 sufficient to cover the development costs of the software?
- **Data Accessibility:** Can reliable and comprehensive Twitter data be accessed efficiently, considering potential API limitations and the analysis depth required to generate valuable insights?
- **Development Demands:** What is the estimated effort and timeframe necessary to develop this software, particularly with functionalities tailored to analyzing sentiment in real-time?
- **Technical Requirements:** Which tools and skills are essential for creating a software that not only meets Amtagio's current needs but is also scalable and adaptable to future marketing strategies?

By accurately assessing public sentiment through targeted Twitter analysis, the start-up aims to refine its marketing approaches, enhance customer engagement, and ultimately, secure a dominant position in the market. The following report will assess the feasibility of this software, providing Amtagio with a roadmap to develop this software.

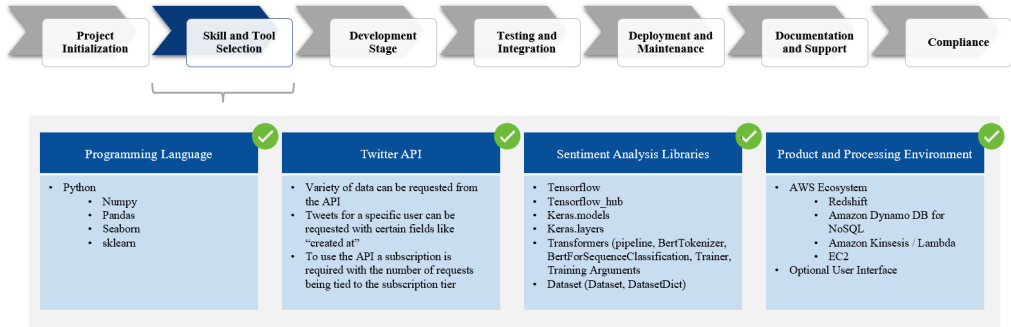
Project Scope

Having introduced the case of the Amtagio entrepreneur, it is essential to outline the project scope. With a budget of €50,000 and an ambitious 2–3-month deadline, the objective is clear: develop software to assess influencer-led marketing campaign efficiency swiftly and accurately. The project can be anchored in two critical phases of feasibility: the Proof of Concept (POC) Phase and the Minimum Viable Product (MVP) Feasibility. Initially, the POC Phase seeks to ascertain if the idea can take off within the confines of the allocated budget and tight timeframe. It’s about assessing if the project is theoretically sound and practical to pursue. Next, the MVP Feasibility phase takes over, diving into whether the POC's promising results can evolve into a basic yet functional version of the product, capable of delivering early insights into the effectiveness of Amtagio's marketing endeavors. While the project flow extends beyond these initial phases, outlining a trajectory from deployment to long-term compliance, these further stages serve as a comprehensive blueprint for the entire project lifecycle.



Technical Evaluation of Open-Source Development Blocks

With the scope clearly defined, the project moves from initialization to a crucial phase where specific technical requirements are outlined, signifying the start of assessing technical feasibility. This step is pivotal for an entrepreneur possessing technical literacy, particularly when operating within the constraints of a limited budget, necessitating efficient tool selection for successful project execution. For the project, all needed skills and tools are available (see figure below), the cost of which will be outlined in the next section in-depth. Consequently, the next step is to evaluate the feasibility of developing the software moving forward into the development stage.

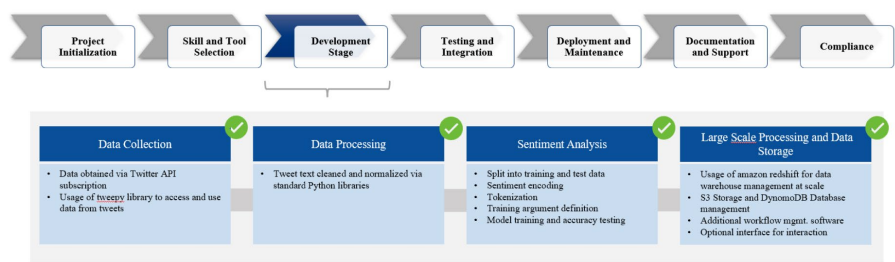


To obtain the tweets from a specific person between certain dates, the Twitter API will be utilized. However, it is of utmost importance to verify that the API can deliver the required information, like username, date and related tweets, as this data forms the foundation of the solution. Generally, the Twitter API can be called using an access token and keys (Twitter, Inc., 2024a). To send requests to the API directly from Python the Tweepy library can be used, which has an implementation for direct access to the Twitter API (Hasan, 2024). According to the Twitter API documentation, each tweet has certain fields as attributes (Twitter, Inc., 2024c). One of these fields is the author field, which denotes the user who sent the tweet, as well as the date when the tweet was posted (Twitter, Inc., 2024a). Using the Tweepy library, it is possible to directly query tweets related to a user (The Tweepy Team, n.d.). Additionally, the created field needs to be called to get the time each tweet was created. This is also possible via (Tweepy Project, n.d.). Then, the tweets can be filtered easily by the 'created at' variable with the required time span. The necessity of a database management system depends on the frequency and scope of the full query and related storage requirements. For these requests, the Twitter rate limit needs to be observed, which is one of the reasons that the PRO subscription for the API is required to have sufficient calls (Twitter, Inc., 2024b) to support the overall project objective. Summarizing, it can be said, that the data required can be obtained in a suitable way. Because the Twitter API returns text, there is no need to clean pictures or other types of data from the responses. However, links will need to be cleaned and the text will need normalization. For this, standard cleaning options in Python are available see, for example, Titanpointe - Kaggle, 2024b. For the sentiment analysis to work properly, only English language text can be used, as the model was trained on English. Therefore, language detection will be employed in Python, for which code is readily available; see, for example, Stack Overflow, Inc., 2021. Using these methods, cleaning the obtained data in a way that makes it suitable for a machine learning (ML) model is definitely feasible.

Sentiment analysis, a cornerstone of modern natural language processing (NLP) endeavors, equips machines with the ability to decipher the underlying sentiments embedded within textual data. The advent of advanced deep learning models, notably the Bidirectional Encoder Representations from Transformers (BERT), has enhanced the feasibility of conducting sentiment analysis with high precision. After examining various Kaggle approaches, including logistic regression and other NLP models, BERT was chosen due to its advanced ability to understand language context and nuances and the lower effort and less processing capacity in training of a suitable model for the entrepreneur (Morgado, K., n.d.). BERT's pre-trained model significantly outperforms simpler algorithms in accuracy and efficiency, making it a superior choice for complex sentiment analysis tasks. BERT, introduced by Google in 2018, represents a paradigm shift in understanding the context and nuances of language. Unlike its predecessors that analyzed text in a unidirectional manner, BERT's architecture is ingeniously designed to consider the bidirectional context of words within sentences being paramount for tasks such as sentiment analysis where the interpretation of context, sarcasm, and nuanced expressions is critical. BERT's profound understanding of language nuances stems from its pre-training on a voluminous corpus of internet text, encompassing two primary tasks: masked language modeling and next sentence prediction. Such extensive pre-training imbues BERT with a nuanced understanding of language, rendering it an optimal candidate for specialized NLP tasks like sentiment analysis (Devlin et. al., 2018). The methodology commences with collecting and preparing the dataset, with a particular emphasis on tweet data, which is inherently replete with opinions and sentiments. The dataset necessitates labeling for sentiment—positive, negative, or neutral. Leveraging pre-labeled data obviates the need for manual labeling or the application of one-hot encoding, thereby streamlining the process. This pre-labeled dataset forms the bedrock upon which the model's training is predicated, facilitating the model's learning from exemplars of sentiment expression within tweets. In this approach, tokenizing the dataset is a crucial step where BERT efficiently processes text by breaking it down into understandable pieces, or tokens, facilitating a deeper understanding of linguistic nuances within the sentiment analysis task. The subsequent phase involves fine-tuning BERT for the bespoke task of sentiment analysis on tweet data, utilizing TensorFlow Hub as a conduit to access pre-trained BERT models. The fine-tuning process is pivotal, entailing adapting BERT's pre-trained layers to the nuances of sentiment analysis in tweets by integrating additional layers specific to sentiment classification atop the foundational BERT model. Such augmentation enables the model to adeptly predict sentiment labels for tweets, harnessing BERT's deep linguistic comprehension (Titanpointe, 2024a). The following stage involves training and testing the model. Here, the fine-tuned BERT model is trained on the collated tweet dataset, enabling it to refine its weights and biases for optimal sentiment classification performance. Subsequent testing on a distinct subset of tweet data serves as a litmus test for the model's proficiency in generalizing its sentiment analysis capabilities to novel, unseen data. Evaluating the model's performance reveals the fine-tuning process's efficacy. Evaluating Python libraries, Matplotlib and Plotly were assessed for their data visualization capabilities, leading to the successful development of tools for aggregating sentiment data.

In refining the architecture for a product in a more scaled state in which the required processing power exceeds the available local processing power in training and implementation, the solution adopts an advanced and integrated approach leveraging Amazon Web Services' (AWS) powerful ecosystem, specifically designed to enhance efficiency, scalability, and robust data management. Central to this architecture is the use of Amazon Redshift, a fully managed, petabyte-scale data warehouse service that provides the backbone for data storage, management, and analysis. The process begins with the ingestion of streaming data, for which Amazon Kinesis is employed. Kinesis facilitates the real-time collection and processing of large streams of data, such as tweets, enabling seamless data flow into the system. Once ingested, the data is processed for preliminary tasks, including filtering and language detection, using AWS Lambda. Lambda's serverless compute service allows for the execution of code in response to triggers, making it ideal for lightweight, real-time data processing tasks. Following initial processing, raw data is stored in Amazon S3, providing a secure and scalable storage solution. S3 acts as a data lake, hosting both raw and processed data, which can be easily accessed for further analysis or visualization. For the efficient handling of metadata and quick-retrieval data needs, Amazon DynamoDB is utilized. DynamoDB's NoSQL database service offers fast and flexible data management, suitable for storing user information, tweet identifiers, and other lightweight data elements. The heavy computational tasks, particularly running the sentiment analysis model, are managed by Amazon Redshift ML. Redshift ML

allows data scientists and developers to create, train, and deploy ML models directly within Amazon Redshift using SQL. This integration significantly simplifies the process of applying ML to the vast amounts of data stored in Redshift, enabling the sentiment analysis model to be applied directly to the data within the warehouse. This direct application streamlines the workflow and eliminates the need for complex data movement or transformation processes. Amazon EC2 instances, optimized for compute-intensive tasks, are deployed to support additional processing needs that may arise outside of Redshift ML's scope. These instances can handle tasks such as model fine-tuning or additional data processing requirements, ensuring that the architecture remains flexible and capable of adapting to varying computational demands. This refined architecture, by incorporating Amazon Redshift and Redshift ML into the workflow, not only streamlines the sentiment analysis process but also enhances the scalability and efficiency of the entire system. It provides a comprehensive solution that integrates data ingestion, storage, processing, and analysis within a cohesive framework, leveraging the full suite of AWS services to deliver a robust and effective sentiment analysis solution (AWS, 2024b). Ultimately, it can be shown that from a technical perspective, the project is feasible having access to important data, code, and storage platforms (see also figure below). Parts of open-source code blocks for the project which will then be used are provided in the appendix.



Cost Feasibility

Besides the technical feasibility, the project needs to be assessed on a financial basis considering the \$50,000 budget. In figure below the key software components and their costs are displayed. With monthly costs of \$5,086.99, the total costs per year would sum up to \$61,043.88 being higher than the budget which the entrepreneur has foreseen for the development of the software. In this respect, it would have to be analyzed whether these costs could possibly be borne and whether the budget is annual or one-off. At this point, it needs to be mentioned that the calculation of processing costs for cloud computing and storage has not been conducted in accordance with the processing requirements and the potential number of API calls necessary to determine the feasibility of the model. As a result, these relatively high calculated costs are based on a product that is massively scaled and assumes a certain constant processing power as well as a continuous improvement of the model. It is questionable whether the Twitter API in the selected Pro form even allows data analyses on this scale. The costs for the pure development of the model (training, testing) are significantly lower and can even be reduced to 0 euros + necessary processing capacity if Twitter entries are written out manually (AWS, 2024a).

Cost component	Costs	Source link
EC2 for Training	~\$73.44 (for 24 hours on a p3.2xlarge instance)	
S3 Storage	~\$2.30 (for storing the dataset and models)	
DynamoDB	~\$1.25 (for metadata storage)	
EBS Volume	~\$10 (for additional storage needs)	
Twitter API	\$5,000 / month for PRO subscription	https://developer.twitter.com/en/products/twitter-api
tweepy: FOSS library	\$0	https://docs.tweepy.org/en/stable/install.html
keras: FOSS library	\$0	https://keras.io/getting_started/
Tensorflow hub: FOSS library	\$0	https://www.tensorflow.org/install
BERT: FOSS as part of tensorflow	\$0	https://huggingface.co/docs/transformers/v4.39.1/en/model_doc/bert#overview
Total	\$5,086.99 per month	

Recommendation and Limitation

The project is ambitiously scoped within a €50,000 budget and 2–3 months aims to develop software for evaluating the effectiveness of influencer-led marketing campaigns. Overall, this venture navigates through limitations: analysis is constrained to English tweets, leaving out opinions from a significant portion of the international market. Furthermore, necessitating known user handles for data collection is a restriction, that limits the scope of the analysis. Twitter's API additionally imposes data retrieval limitations, skewing the analysis towards historical data and precluding real-time insights. GDPR considerations mandate stringent data handling, potentially raising the complexity of the project. The inherent bias within Twitter’s user demographic and the potential noise in the training data may skew insights, further complicated by Twitter's representation of market engagement potentially not mirroring the actual market landscape. These challenges underscore the need for meticulous planning and adaptive strategies to navigate the project's limitations effectively. With some limitations the project is generally technically feasible. Whether it is financially feasible in the long-term needs to be calculated by comparing the costs to the value add for the entrepreneur and Amtagio. With the current budget the project could run for between 9 and 10 months. Based on the basically simple possibility and many existing codes and models and the high importance of influencer-led marketing for Amtagio, it would be advisable to reconsider the budget and, if possible, invest more in the project.

Declaration on Plagiarism Assignment Submission Form

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying.

I have read and understood the Assignment Regulations. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references (note: No direct quotations are allowed for this essay)

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines recommended in the assignment guidelines and found at:

- <http://www.dcu.ie/info/regulations/plagiarism.shtml>
- <https://www4.dcu.ie/students/az/plagiarism and/>

Dublin, 08.04.2024

Caroline Ziegler

A handwritten signature in black ink, appearing to read 'Caroline Ziegler', is written over a horizontal line.



Reference List

- Amazon Web Services, Inc. (AWS) (2024a). AWS Pricing Calculator. Retrieved March 27, 2024, from https://calculator.aws/#/?nc2=h_ql_pr_calc
- Amazon Web Services, Inc. (AWS) (2024b). Large language models for sentiment analysis with Amazon Redshift ML (Preview). AWS Big Data Blog. Retrieved March 27, 2024, from <https://aws.amazon.com/blogs/big-data/large-language-models-for-sentiment-analysis-with-amazon-redshift-ml-preview/>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. N. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL 2019. Google Research. Retrieved March 27, 2024, from <https://research.google/pubs/bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding/>
- Hasan. (2024). Tweepy: Internet's Most Loved TwitterAPI Python Library. Tweepy.org. Retrieved March 27, 2024, from <https://www.tweepy.org/>
- Morgado, K. (n.d.). Twitter sentiment analysis - Logistic Regression. Kaggle. Retrieved March 27, 2024, from <https://www.kaggle.com/code/kevinmorgado/twitter-sentiment-analysis-logistic-regression>
- Stack Exchange, Inc. (2021, January 3). Dropping non-English text with langdetect. Stack Overflow. Retrieved March 27, 2024, from <https://stackoverflow.com/questions/65551659/dropping-non-english-text-with-langdetect>
- Titanpointe. (2024a). BERT with TensorFlow Hub. Kaggle. Retrieved March 27, 2024, from <https://www.kaggle.com/code/titanpointe/cyberbullying-tweets-eda-automl-dl-bert/notebook#%E2%AC%85%EF%B8%8F%E2%9E%A1%EF%B8%8F-BERT-with-Tensorflow-Hub->
- Titanpointe. (2024b). Cyberbullying tweets EDA + AutoML + DL + BERT. Kaggle. Retrieved March 27, 2024, from <https://www.kaggle.com/code/titanpointe/cyberbullying-tweets-eda-automl-dl-bert?scriptVersionId=158955582&cellId=32>
- The Tweepy Team. (n.d.). Examples. Tweepy Documentation. Retrieved from <https://docs.tweepy.org/en/stable/examples.html>
- Tweepy Project. (n.d.). Examples. Tweepy Documentation. Retrieved March 27, 2024, from <https://docs.tweepy.org/en/stable/examples.html>
- Twitter, Inc. (2024a). Getting access to the Twitter API. Twitter Developer Platform. Retrieved March 27, 2024, from <https://developer.twitter.com/en/docs/twitter-api/getting-started/getting-access-to-the-twitter-api>
- Twitter, Inc. (2024b). Rate limits. Twitter Developer Platform. Retrieved March 27, 2024, from <https://developer.twitter.com/en/docs/twitter-api/rate-limits#v2-limits-pro>
- Twitter, Inc. (2024c). Tweet object. Twitter Developer Platform. Retrieved March 27, 2024, from <https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet>

Appendix

Code to call Twitter API via tweepy (The Tweepy Team. (n.d.)).

Python ▾

 Copy  Caption 

```
import tweepy

bearer_token = ""

client = tweepy.Client(bearer_token)

# Get User's Tweets

# This endpoint/method returns Tweets composed by a single user, specified by
# the requested user ID
user_id = 2244994945

response = client.get_users_tweets(user_id)
# By default, only the ID and text fields of each Tweet will be returned
for tweet in response.data:
    print(tweet.id)
    print(tweet.text)

# By default, the 10 most recent Tweets will be returned
# You can retrieve up to 100 Tweets by specifying max_results
response = client.get_users_tweets(user_id, max_results=100)
```

Model development blueprint with BERT

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report
from transformers import pipeline
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from datasets import Dataset, DatasetDict
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer
from transformers import DataCollatorWithPadding

training_data_path = '/Users/chantalstefan/Documents/Scripting/Data Analytics Portfolio für Git/Unsupervised Learning/twitter_training.csv'
validation_data_path = '/Users/chantalstefan/Documents/Scripting/Data Analytics Portfolio für Git/Unsupervised Learning/twitter_validation.csv'

training_df = pd.read_csv(training_data_path)
validation_df = pd.read_csv(validation_data_path)

columns = ['ID', 'Topic', 'Sentiment', 'Text']
training_df.columns = columns
validation_df.columns = columns

training_df.dropna(subset=['Text'], inplace=True)
validation_df.dropna(subset=['Text'], inplace=True)

training_df = training_df[['Sentiment', 'Text']]
validation_df = validation_df[['Sentiment', 'Text']]

sentiment_encoding = {'Positive': 3, 'Neutral': 2, 'Negative': 1, 'Irrelevant': 0}
training_df['Sentiment'] = training_df['Sentiment'].map(sentiment_encoding)
validation_df['Sentiment'] = validation_df['Sentiment'].map(sentiment_encoding)

train_dataset = Dataset.from_pandas(training_df)
val_dataset = Dataset.from_pandas(validation_df)
datasets = DatasetDict({'train': train_dataset, 'validation': val_dataset})

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

def tokenize_function(example):
    return tokenizer(example['Text'], truncation=True, padding='max_length', max_length=512)

tokenized_datasets = datasets.map(tokenize_function, batched=True)

tokenized_datasets = tokenized_datasets.rename_column("Sentiment", "labels")

tokenized_datasets.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])

model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=4)

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy='epoch',
    save_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='accuracy',
)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer, return_tensors="pt")

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['validation'],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()

validation_df['labels'] = validation_df['Sentiment'].map(sentiment_encoding)
validation_df = validation_df[['labels', 'Text']].dropna(subset=['Text'])

def tokenize_function(examples):
    return tokenizer(examples["Text"], padding="max_length", truncation=True)

tokenized_validation_dataset = Dataset.from_pandas(validation_df).map(tokenize_function, batched=True)
tokenized_validation_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])

trainer.model = BertForSequenceClassification.from_pretrained('./results/checkpoint-best')

predictions = trainer.predict(tokenized_validation_dataset)

predicted_labels = predictions.predictions.argmax(-1)
true_labels = predictions.label_ids

accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Validation accuracy: {accuracy}")

from sklearn.metrics import classification_report
report = classification_report(true_labels, predicted_labels, target_names=sentiment_encoding.keys())
print(report)
```


Model development with the open-source keras framework

```
from keras.models import Sequential
from keras.layers import LSTM, GRU, SimpleRNN
from keras.layers import Dense, Activation, Dropout
from keras.layers import Embedding
from keras.layers import BatchNormalization
from keras.utils import to_categorical
from sklearn import preprocessing, decomposition, model_selection, metrics, pipeline
from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D, Flatten, Bidirectional, SpatialDropout1D
from keras.preprocessing import sequence, text
from keras.callbacks import EarlyStopping
import tensorflow as tf
import tensorflow_hub as hub

# Assuming X_train, y_train, X_test, y_test are previously defined and properly preprocessed for BERT.

train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test)) # Include y_test if you have labels for evaluation.

BUFFER_SIZE = 4000
BATCH_SIZE = 64 # Keep BATCH_SIZE consistent or adjust as needed before batching the dataset.
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/2'
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'

# These instances are not used in the function create_model; consider using them directly or remove if not needed.
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
bert_model = hub.KerasLayer(tfhub_handle_encoder)

def create_model(model_link, preprocess_link):
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(preprocess_link, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(model_link, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)

    model = tf.keras.Model(text_input, net)

    loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    metrics = tf.metrics.BinaryAccuracy()
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss=loss,
                  metrics=metrics)

    return model

# Assuming you are not using a distributed training strategy.
# If you are, define it accordingly and wrap model creation and compilation in `with strategy.scope():`
model = create_model(tfhub_handle_encoder, tfhub_handle_preprocess)

DATA_SIZE = tf.data.experimental.cardinality(train_dataset).numpy()
print("Size of the dataset:", DATA_SIZE)
STEPS_PER_EPOCH = DATA_SIZE // BATCH_SIZE

model.fit(train_dataset, epochs=10, steps_per_epoch=STEPS_PER_EPOCH)

loss, accuracy = model.evaluate(test_dataset)
print(f"Test accuracy: {accuracy}")
```