University of Applied Sciences in Tarnów

Department of Computer Science

Emotion Sentiment Analysis

Field of study: Computer Science
Course: Big Data and Warehousing II
Academic year: 2023/2024

Authors:

- Kamil Rataj, 35712
- Marcin Golonka, 35677
- Kijowski Kacper, 35207

- Jakub Jędrychowski, 35204

1. Project description and conceptual design.

1.1. Project description.

This project aims to develop a robust emotion and sentiment analysis system using Big Data techniques. The objective is to create a model capable of accurately identifying and classifying emotions expressed in text data, specifically focusing on social media posts. This project addresses the growing need for understanding and analysing emotions in digital communication, which has significant implications for various fields, including marketing, customer service, and social research.

The project leverages a collection of publicly available datasets containing text and associated emotion labels, sourced from Kaggle. By utilising these datasets, the project explores the effectiveness of various Big Data and machine learning techniques in effectively analysing and extracting emotion-related insights from text data.

1.2. Conceptual design.

The conceptual design of this project involves the following key components:

- **Data Acquisition and Preprocessing:**
    - **Data Sources:** Publicly available datasets from Kaggle focusing on text and emotion labels.
    - **Data Extraction:** Utilising the Kaggle API to download and process datasets.
    - **Data Cleaning and Normalisation:** Implementing data cleaning techniques to remove noise, inconsistencies, and irrelevant information. This includes:
        - Standardisation of column names and emotion labels
        - Removing null values
        - Cleaning text data (removing mentions, links, punctuation)
        - Replacing chat words with their full form
        - Removing stop words
        - Normalising whitespace
    - **Data Deduplication:** Removing duplicate entries.
- **Data Model Design:**
    - **Data Storage:** Leveraging a data lake approach with suitable technologies like Azure Data Lake or Snowflake.
    - **Data Schema:** Designing a schema that effectively captures the text and emotion labels from the datasets.
- **ETL Process:**
    - **Data Extraction:** Extracting data from the data lake based on defined queries or filters.
    - **Data Transformation:** Applying necessary transformations to the extracted data, including:
        - Tokenization of text data
        - Padding sequences
        - Encoding emotion labels using techniques like One-Hot Encoding.
        - Data Loading: Loading the transformed data into the desired format for training and analysis.

- **Model Development:**
    - **Machine Learning Approach:** Employing a recurrent neural network (RNN) architecture, specifically utilising Bi-directional Long Short-Term Memory (LSTM) layers.
    - **Model Layers:** Implementing layers like embedding, dropout, batch normalisation, and dense layers to improve model accuracy and prevent overfitting.
    - **Model Training:** Training the model using the preprocessed data, optimising hyperparameters (epochs, batch size) to achieve optimal performance.
- **Model Evaluation and Testing:**
    - **Evaluation Metrics:** Assessing the model's performance using metrics such as accuracy, loss, and classification report.
    - **Model Deployment:** Saving the trained model for future use.
    - **New Text Prediction:** Testing the trained model on new, unseen text data to evaluate its ability to predict emotions accurately.

This conceptual design provides a roadmap for implementing the project, showcasing the key components involved in building a robust emotion and sentiment analysis system. The project aims to leverage Big Data technologies and machine learning techniques to gain meaningful insights from text data, contributing to a better understanding of emotions in digital communication.

2. Data model design.

The data model design for the emotion classification project involves several key components. This section outlines the structure and organisation of the data as well as the preprocessing steps taken to prepare the data for modelling.

2.1 Data Sources

The data for this project is sourced from various Kaggle datasets, which include text samples labelled with different emotions. The datasets used are:

- `tweet_emotions.csv`
- `Emotion_classify_Data.csv`
- `emotion_sentimen_dataset.csv`
- `Text.csv`

These datasets are combined to create a comprehensive dataset for training the emotion classification model.

2.2 Data Schema

The data schema for the combined dataset includes the following fields:

- `text`: A string containing the text data.
- `emotion`: A string representing the emotion label associated with the text.

Each dataset may have different column names for these fields, which are standardised during preprocessing.

2.3 Data Preprocessing Steps

To ensure consistency and quality of the data, several preprocessing steps are applied:

2.3.1 Standardisation of Column Names:

- The column names in each dataset are standardised to `text` for the text data and `emotion` for the emotion labels.

2.3.2 Standardisation of Emotion Labels:

- Emotion labels are mapped to a consistent set of labels using a predefined dictionary. This ensures that different representations of the same emotion are unified.

2.3.3 Handling Missing Values:

- Any records with missing values in the `text` or `emotion` columns are removed to ensure the dataset is clean.

2.3.4 Text Cleaning:

- Text data is cleaned to remove user mentions, links, and special characters. This involves using regular expressions to strip unwanted elements.

2.3.5 Chat Words Replacement:

- Common chat abbreviations and slang are replaced with their full forms to normalise the text data.

2.3.6 Stop Words Removal:

- Stop words, which are common words that do not contribute much to the meaning of the text, are removed using NLTK's list of English stop words.

2.3.7 Whitespace Normalisation:

- Multiple consecutive spaces in the text are replaced with a single space to ensure uniform formatting.

2.3.8 Dropping Duplicates:

- Duplicate records based on the `text` column are removed to ensure each text sample is unique.

2.4 Data Integration

The preprocessed datasets are integrated into a single dataframe. This involves concatenating the individual datasets and performing the necessary cleaning and standardisation steps.

2.5 Data Transformation

The following transformations are applied to prepare the data for model training:

2.5.1 Label Encoding:

- The emotion labels are encoded into numerical values using scikit-learn's `LabelEncoder`. This is necessary for training the neural network.

2.5.2 Text Tokenization:

- The text data is tokenized and converted into sequences of integers using Keras' `Tokenizer`. This step translates words into numerical format that the neural network can process.

2.5.3 Padding Sequences:

- The tokenized sequences are padded to ensure all sequences have the same length. This is required for batch processing in neural networks.

2.6 Train-Test Split

The final dataset is split into training and testing sets to evaluate the model's performance. Typically, an 80-20 split is used where 80% of the data is used for training and 20% for testing.

2.7 Data Summary

- **Number of Records**: The combined dataset has approximately 1,262,301 records after cleaning and preprocessing.
- **Number of Unique Emotions**: The dataset includes 14 unique emotion labels.

The careful design and preprocessing of the data model ensure that the dataset is clean, consistent, and suitable for training an accurate emotion classification model.

3. Interface implementation for datasource connection and data import.

3.1. Data Sources and imports

The data source for this project was Kaggle, a platform for data scientists and machine learning practitioners to share and access datasets. To connect to Kaggle and import the required datasets, we implemented a Python-based interface utilising the Kaggle API. This API provides a programmatic way to interact with Kaggle's data repositories and download datasets directly into our Google Colab environment.

We first configured the Kaggle API by setting up authentication credentials. These credentials, stored securely in a JSON file, allowed us to authorise our Google Colab instance to access Kaggle's resources. Once authenticated, we developed a Python function that utilises the Kaggle API to download specific datasets identified by their unique Kaggle identifiers. This function efficiently handles the download process, including the extraction of zip files and the conversion of datasets into the required formats.

Here's a breakdown of the technologies, process, and challenges:

- **Technologies:**
    - **Kaggle API:** This Python library provided the core functionality for interacting with Kaggle's datasets.
    - **Python:** We used Python's requests library to make API calls to Kaggle and zipfile library to handle zip file extraction.
    - **Google Colab:** The platform where we developed and executed our code.
- **Data Extraction Process:**
    - **Authentication:** We used a JSON file containing our Kaggle API credentials to authenticate our Google Colab instance.
    - **Dataset Identifier:** We defined a list of Kaggle dataset identifiers (e.g., 'simaanjali/emotion-analysis-based-on-text') representing the datasets we wanted to download.
    - **API Call:** Using the kaggle.api.dataset_download_files function, we made API calls to Kaggle to initiate the download process for each dataset.

   - **Zip Extraction:** Once downloaded, we used the zipfile library to extract the contents of the ZIP files.
   - **Data Conversion:** We converted the extracted files into the required formats (e.g., CSV) for further processing.

- **Efficiency:**

   - The Kaggle API and the requests library facilitated efficient data extraction.
   - Utilising Google Colab's infrastructure allowed for parallel processing, speeding up the download process.

- **Challenges and Limitations:**

   - **Rate Limits:** The Kaggle API has rate limits, which meant we had to be mindful of the number of requests we made within a specified time frame.
   - **Data Availability:** Some datasets were not readily available or had limitations in terms of the volume or type of data.
   - **File Size:** Large datasets posed a challenge in terms of download time and storage capacity.
   - **Google Colab daily limits:** Colab has a limited daily amount of processing power one account can use.

Code Snippets:

```python
import kaggle

def download_kaggle_datasets(dataset_list: str='datasets_source.txt') -> None:
    """
    Download each dataset specified in the list from Kaggle using the Kaggle API.
    Skips blank lines in file.

    Args:
    dataset_list (str): Path to a file containing a list of dataset identifiers on
Kaggle.
    """
    try:
        with open(dataset_list, 'r') as file:
            for line in file:
                line = line.strip()
                if line:
                    try:
                        kaggle.api.dataset_download_files(line, path='datasets/',
unzip=True)
                        print(f"Successfully downloaded {line}")
                    except Exception as e:
                        print(f"Failed to load {line}: {e}")
    except FileNotFoundError:
        print(f"File {dataset_list} not found.")
    except Exception as e:
        print(f"An error occurred while reading {dataset_list}: {e}")

download_kaggle_datasets('datasets_source.txt')
```

This snippet demonstrates the Python function used to download datasets from Kaggle. The function takes a list of Kaggle dataset identifiers as input and uses the `kaggle.api.dataset_download_files` function to download each dataset into the specified directory. The code also includes error handling for situations where files are not found or other errors occur during the download process.

This detailed explanation showcases the implementation of the data source connection and import process in the project. The use of the Kaggle API, combined with Python libraries and Google Colab, enabled efficient extraction of necessary datasets for the emotion and sentiment analysis system.

4. ETL / Data processing

1. Extraction:

Data was extracted from the downloaded datasets in CSV format. This was a straightforward process using the `pandas` library, which allowed for efficient reading of CSV files into DataFrames.

2. Transformation:

- **Text Cleaning:** This involved removing noise and irrelevant information from the text data. Specific steps included:
    - Removal of User Mentions and URLs: Patterns like `@username` and `http://...` were removed to focus on the core content of the text.
    - Removal of Punctuation: Punctuation marks were removed to simplify the text and focus on words.
    - Normalisation of Whitespace: Multiple consecutive spaces were replaced with single spaces to ensure consistency.
    - Chat Word Replacement: Common internet slang or abbreviations (e.g., "LOL", "BRB") were replaced with their full forms.
- Stop Word Removal: Stop words, common words with little semantic value (e.g., "a", "the", "is"), were removed using NLTK's stop word library. This helped to reduce the dimensionality of the data and focus on more meaningful words.
- Stemming and Lemmatization: Although not implemented in this specific project, stemming (reducing words to their root forms) and lemmatization (finding the base form of a word) can be useful text transformation techniques for sentiment analysis.
- Tokenization and Padding:
    - Tokenization: The cleaned text was tokenized, which involves splitting the text into individual words. We used the `Tokenizer` class from the TensorFlow library to perform this step. This tokenization process assigns a unique index to each word.
    - Padding: The tokenized sequences were padded to a uniform length using the `pad_sequences` function. This step ensured that all input

sequences had the same length, which is required for training a recurrent neural network model.

3. **Loading:**

Data Structure: The transformed data was loaded into a suitable data structure for training the machine learning model. We used a combination of `NumPy` arrays and TensorFlow's `Dataset` class to create a data structure suitable for the RNN model.

4. **Feature Engineering:**

The tokenization and padding steps described above also served as feature engineering techniques. These processes converted the raw text data into numerical representations that the machine learning model could understand.

5. **Data Quality Checks and Validation:**

- Data Validation: We checked for consistency in the emotion labels, ensuring they were correctly assigned and represented a reasonable range of emotions.
- Data Distribution: We analysed the distribution of emotion labels within the dataset to identify any potential biases or imbalances.

This ETL process transformed the raw text data into a format suitable for training and evaluating our emotion and sentiment analysis model. The cleaning and transformation steps were crucial for ensuring that the model could learn effectively from the data, leading to improved accuracy and performance.

Code Snippets:

```python
def clean_text(text: str) -> str:
    """
    Preprocess the text data by removing user mentions, links
    and unnecessary characters.

    Args:
    text (str): The text to preprocess.

    Returns:
    str: The preprocessed text.
    """

    text = re.sub(r'@\w+', '', text)  # Remove user mentions
    text = re.sub(r'http\S+|www.\S+', '', text)  # Remove URLs
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text)  # Remove punctuation

    return text
```

`clean_text` **Function:**

- This function demonstrates the text cleaning process. It removes user mentions (`@username`), URLs (`http://...`), and punctuation using regular expressions.
- This function uses the re library for regular expression matching.

```
def remove_stop_words(text: str) -> str:
    """
    Removes stop words from the input text.

    Args:
    text (str): The input text.

    Returns:
    str: The text without stop words.
    """
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not
in stop_words]
    return ' '.join(filtered_words)
```

`remove_stop_words` **Function**:

- This function removes common stop words from the text using NLTK's stop word library.
- The function uses a `set` for efficient membership checks.

```
# Tokenization and Padding
tokenizer = Tokenizer(num_words=60000, lower=True)
tokenizer.fit_on_texts(texts)  # Fit tokenizer on the cleaned text data
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(texts)  # Convert text to sequences
padded_sequences = pad_sequences(sequences, maxlen=100)  # Pad sequences to uniform length
```

Tokenization and Padding:

- The `Tokenizer` class is used to convert the text data into sequences of integers, where each integer represents a unique word.
- The `pad_sequences` function ensures that all sequences have the same length, which is crucial for training an RNN model.
- The `maxlen` parameter specifies the desired length of the padded sequences.

5. Data analysis.

5.1. Preprocessing and Cleaning

Concise Description of the Analysis:

The preprocessing and cleaning step involves standardising the data, removing unnecessary characters, normalising text, and ensuring the data is in a suitable format for further analysis and modelling.

5.1.1. Script/Code:

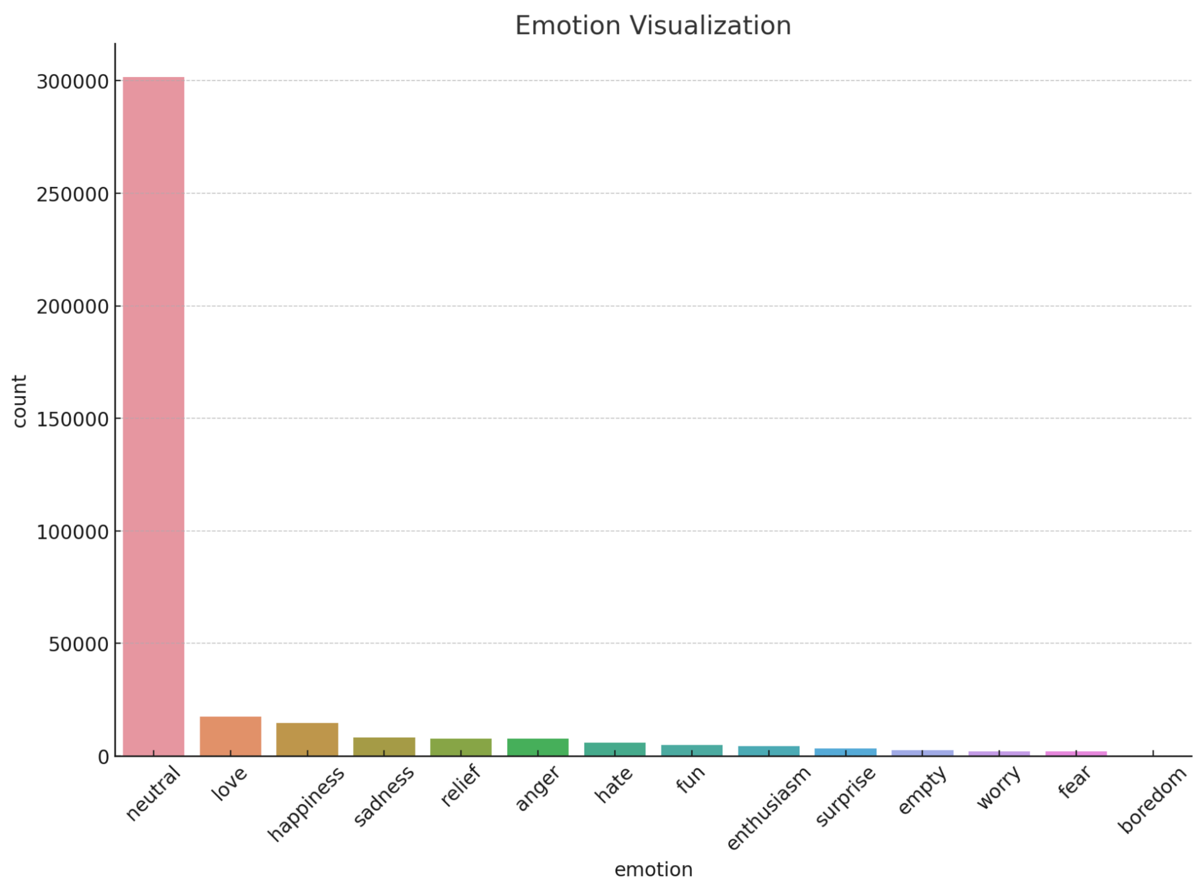
The following code snippets demonstrate the preprocessing and cleaning steps taken on the datasets:

```
# Standardise column names
def standardize_column_names(frame: pd.DataFrame, column_map: dict) ->
list[pd.DataFrame]:
    frame.columns = map(str.lower, frame.columns)
    reverse_mapping = {synonym: standard for standard, synonyms in
column_map.items() for synonym in synonyms}
    for old_name, new_name in reverse_mapping.items():
        if old_name in frame.columns:
            frame.rename(columns={old_name: new_name}, inplace=True)
    return frame

# Standardise labels
def standardize_emotion_names(dataframe: pd.DataFrame, emotion_dict:
dict) -> pd.DataFrame:
    reverse_emotion_dict = {synonym: emotion for emotion, synonyms in
emotion_dict.items() for synonym in synonyms}
    dataframe['emotion'] =
dataframe['emotion'].replace(reverse_emotion_dict)
    return dataframe

# Clean text data
def clean_text(text: str) -> str:
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text)
    return text

# Replace chat words
```

```
def replace_chat_words(text: str) -> str:
    words = text.split()
    for i, word in enumerate(words):
        if word.lower() in chat_words:
            words[i] = chat_words[word.lower()]
    return ' '.join(words)

# Remove stop words
def remove_stop_words_from_dfs(df_list: list[pd.DataFrame], stop_words)
-> list[pd.DataFrame]:
    df_words_list = []
    for _df in df_list:
        _df["text"] = _df['text'].apply(lambda x: ' '.join([word for
word in x.split() if word not in stop_words]))
        df_words_list.append(_df)
    return df_words_list

# Normalise whitespace
def normalize_whitespace_in_dfs(df_list: list[pd.DataFrame]) ->
list[pd.DataFrame]:
    df_stemmed_list = []
    for _df in df_list:
        _df['text'] = _df['text'].str.replace(r'\s+', ' ', regex=True)
        df_stemmed_list.append(_df)
    return df_stemmed_list

# Concatenate data frames
def concat_dataframes(dataframes: list[pd.DataFrame]) -> pd.DataFrame:
    final_dataset = [df[['text', 'emotion']] for df in dataframes]
    return pd.concat(final_dataset, ignore_index=True)

# Remove duplicates
def remove_duplicates_and_count(df: pd.DataFrame) -> pd.DataFrame:
    df_unique = df.drop_duplicates(subset=['text'])
    return df_unique

# Execution of preprocessing steps
df_loaded_list = load_datasets_from_file('datasets.txt')
df_std_list = [standardize_column_names(_df, column_mapping) for _df in
df_loaded_list]
df_std_list_2 = [standardize_emotion_names(_df, emotions_dictionary) for
_df in df_std_list]
df_std_list_3 = [_df.dropna() for _df in df_std_list_2]
```

```
df_clean_list = [_df['text'].apply(clean_text) for _df in df_std_list_3]
df_clean_list_2 = [_df['text'].apply(replace_chat_words) for _df in
df_clean_list]
df_words_list = remove_stop_words_from_dfs(df_clean_list_2, stop_words)
df_stemmed_list = normalize_whitespace_in_dfs(df_words_list)
df_merged = concat_dataframes(df_stemmed_list)
df_unique = remove_duplicates_and_count(df_merged)
df_unique = df_unique[['text', 'emotion']]
df_shuffled = df_unique.sample(frac=1).reset_index(drop=True)
```

5.1.2. Visualisation:



5.1.3. Analysis Results:

- The dataset contains 382,153 unique records after preprocessing.
- The distribution of emotions is heavily skewed towards the 'neutral' emotion, followed by 'love', 'happiness', 'sadness', and other emotions.
- Visualisation indicates the imbalance in emotion classes, which will be a critical factor to consider during model training to ensure it does not bias towards the majority class.