Jokes Classification Preprocessing Report

1. Introduction

This project aims to create a machine learning classifier that can classify various types of jokes according to their humour style. Humour is subjective and diverse, making it challenging for automated systems to understand and recommend jokes effectively. By classifying jokes into predefined categories (such as puns, satire, wordplay, etc.), this classifier will enable a program to understand the specific humour style of a given joke.

Once the classifier accurately identifies the humour type, it will be used to suggest five similar jokes to the user. This system aims to enhance personalised humour recommendations, providing users with content that aligns with their comedic preferences. The overall goal is to improve user engagement with humour-driven applications by making joke suggestions more tailored and contextually relevant.

Classifying jokes into appropriate categories can aid in better understanding humour and improve content recommendation. The goal of this project was to preprocess and classify a large dataset of jokes using the OpenAI API. This report outlines the steps involved in data preprocessing, embedding generation, and batch classification for the jokes dataset.

2. Problem Formulation

The task is a **supervised learning problem** where the objective is to classify jokes into one of 26 predefined categories. The predefined categories, which serve as the labels for this task, include a wide variety of **categorical** joke types, such as Animal Jokes, Puns, Dad Jokes, Knock-Knock Jokes, and many others. This diverse set of categories covers different themes and forms of humour, ensuring that the model learns to classify jokes across a broad spectrum of content.

While others have used TF-IDF, which focuses on using word frequency to label their jokes into distinct categories. To provide our model with a good context of the jokes for categorization, a Natural Language Processing (NLP) approach was used, where embeddings were generated to represent the content of the jokes. These embeddings, consisting of **1536-dimensional** vectors, serve as the **multidimensional input features** for the classification model. The embeddings provide a dense representation of the jokes, capturing their meanings and contexts by mapping them into a vector space where semantically similar jokes are placed closer together.

3. Data Processing

The dataset was sourced from the **r/Jokes Dataset** repository, a large-scale collection of humor data created by Orion Weller and Kevin Seppi. The dataset can be found in the following GitHub repository: orionw/rJokesData. The dataset is multidimensional containing textual features and numeric features such as joke IDs(categorical), titles and joke bodies(text/categorical), scores(integer), and timestamps(continuous numerical). For our model, we really only needed the jokes themselves so the key steps for data cleaning and preparation included:

1. **Quality Control**: Only jokes with scores above 50 were selected, to focus on the cream of the crop
2. **Text Cleaning**: Titles and joke bodies were combined, and unnecessary elements like URLs, special characters, and formatting were removed. Furthermore, jokes with missing or deleted content were skipped and jokes were lowercase and stripped of any special characters or extra spaces. This yielded 88,832 jokes for further classification.
3. **Storing Filtered Data**: Cleaned jokes were saved in a separate file, providing a checkpoint to preserve progress and ensure consistency for subsequent processing steps.

Next, we used two key preprocessing steps to prepare our data:

1. **Embeddings**: We generated 1536-dimensional embeddings using OpenAI's text-embedding-3-small model. Unlike TF-IDF, which focuses on word frequency, embeddings capture semantic relationships and context crucial for understanding humour.
2. **Categorization**: We used OpenAI's gpt-3.5-turbo-0125 model to assign each joke to one of 26 predefined categories (e.g., Puns, Dark Humor) based on its content.

For both processes, we batched jokes (10,000 for embeddings, 7,000 for categorization), made API calls at 30-minute intervals to avoid rate limits, and merged results with our original dataset. This approach provided rich contextual representations and specific categories for each joke, forming the foundation for our classification and recommendation system.

This preprocessing stage successfully prepared the jokes dataset, generated embeddings, and classified jokes based on predefined categories. Embeddings were chosen over TF-IDF due to their ability to capture the contextual and semantic richness of the jokes, making them more suitable for this task. The next stage involves training a classification model to automatically categorise jokes based on their embeddings.

4. Model Selection and Motivation

In this classification task, a **Random Forest model (Which is an implementation of decision trees)** is chosen due to its ability to use high-dimensional input data like the joke embeddings to categorise a datapoint to a distinct feature. Random Forest classifiers consist of an ensemble of decision trees, where each tree is built using random subsets of the data and features, allowing it to capture complex, non-linear relationships between the embeddings and the joke categories. Random Forest is robust to overfitting, especially when the number of features is large, as it averages predictions across multiple trees, reducing variance. This is particularly useful in this case, as embeddings can have complex feature interactions that may not be easily captured by simpler models. Additionally, its performance and scalability make it suitable for large datasets like the 88,000 jokes in this task. For all these reasons, we decided it was the best model for the task.

5. Loss Function (or Evaluation Metric) and Motivation

For the Random Forest classifier, the **Gini impurity loss function** is used to evaluate the quality of splits during tree construction. Gini impurity measures how often a randomly chosen element from the dataset would be incorrectly classified if it were randomly labelled according to the distribution of classes. It is chosen because it provides a computationally efficient way to select the best feature splits, allowing the model to handle large datasets like the 88,832 joke embeddings effectively. Additionally, Gini impurity is well-suited for multiclass classification tasks, as it helps the model focus on reducing classification error with each decision. This makes it well suited for the multidimensional embeddings of our dataset.

6. Training and Validation Strategy

To ensure effective training and evaluation of the Random Forest model, we use a **90/10 stratified split for the training and validation sets**. This means 90% of the data (approximately 79,200 jokes) is allocated for training, while the remaining 10% (8,800 jokes) is used for validation. Stratification is applied to maintain an even distribution of the joke categories across both sets, ensuring the validation set accurately represents the overall dataset. A 90/10 split is chosen to provide the model with a large training set to capture the complexity of the joke embeddings, while retaining a sufficient validation set for reliable performance assessment. Additionally, **5-fold cross-validation** is incorporated within the training set to further evaluate the model's robustness, ensuring it performs consistently across different subsets of the data. This approach minimises overfitting and provides a more comprehensive evaluation of the model's generalisation ability.

7. Feature Selection and Feature Engineering

In this study, **no feature selection was applied** to the 1536-dimensional embeddings used as input features. Embeddings are designed to encapsulate detailed and nuanced information across all dimensions, and reducing the number of features could result in the loss of important information needed for accurate classification. Initially, we considered performing feature selection to reduce dimensionality and improve training efficiency. However, exploratory analysis of feature importance did not reveal clear distinctions that would justify reducing dimensions without impacting model performance. This is likely because embeddings capture semantic information across all dimensions and decreasing one dimension will likely lead to the loss of some information used for categorization. As a result, we retained all 1536 dimensions, ensuring that the model can fully leverage the embedded representation of the jokes.

Appendix:

The code can be found in the following github repository:
https://github.com/henryblu/data_science_project.git