University of Essex
Department of Computer Science

CE903: GROUP PROJECT

FUNNINESS PREDICTOR OF A JOKE

Team Number: 14

Supervisor: Prof. Luca Citi

March 23, 2023
Colchester

Abstract

The most crucial quality in human conversation is the ability to create and comprehend humour. In this paper, we describe a novel method for assessing humour from stand-up comedy transcripts. In recent years, it can be seen that literature has examined verbally communicated humour, focusing in particular on very brief statements (jokes). Those statements are one-liners with humour effects, straightforward syntax, frequently innovative language constructions and deliberate use of rhetorical devices which includes alliteration and rhyme. The BERT model was built using the TensorFlow framework and the Hugging Face Transformers. The evidences thus found revealed a significant correlation between the predicted and real funniness ratings. The model achieved a validation mean-absolute error (MAE) of 1.85, which suggests that the model is able to predict the funniness of a joke with reasonable accuracy.

Contents

List of Figures

CHAPTER 1

Introduction

An essential part of human communication that makes a conversation interesting and enjoyable is humor and the ability to comprehend it as a joke is what differentiates between a machine and a human being. Although the prospect of deciphering the joke as funny or not depends on how the listener is able to comprehend it, it is noteworthy that recent advancements in the field of natural language processing (NLP) as well as Machine Learning techniques have led to the invention of powerful models that can not only understand but can also analyze the sentiment of a given statement. One of such models which shows phenomenal performance on a variety of NLP tasks is the Bidirectional Encoder Representations from Transformers (BERT) model.
This paper serves as a purpose to describe the performance of a BERT model by predicting the funniness of a joke. Here a machine learning model is built wherein we put the transcripts of a video as the input and thereby receive the funniness of a joke as the output by calculating the intensity and duration of the joke. Furthermore, we use the dataset obtained to compare it with other already existing models in order to analyze and examine the factors that help in characterizing the funniness of a joke that may include prominent features such as linguistic, contextual and references relating to certain culture. [1]

The research questions that we had to solve while testing the model were very basic yet complicated in nature. Those were :

1. Will the model be able to accurately detect the funniness of a joke and score them

accordingly?

2. What were the major features that contributed to the high scores in a joke which contributed the joke to be more funny than the rest?
3. What advancement can we make as compared to the existing models?
4. How will the model help in the area of money making and contributing to the field of technology?

Human perception can be understood to an exemplar level if the cognitive and linguistic mechanisms that are thus involved,is able to make the model contribute to generating humorous content. A major question to understand is why people find things funny. Humor and jokes are complex topics which are difficult to define but sharing a sense of humor is important for building relationships. Humor can take many forms, such as jokes, personal anecdotes, memes.There are several theories of laughter that explains why people find things funny. They are classified as:

1. Superiority Theory: This theory suggests that people laugh at things that make them feel superior to others. For example, someone might laugh at a joke that mocks someone else's mistakes or shortcomings.
2. Incongruity Theory: This theory suggests that people laugh when they hear or find a thing which does not appear normal to the surroundings . For example, a pun or a joke with an unexpected line can create a lack of harmony.
3. Relief Theory: This theory suggests that laughter is a release of nervous energy or tension. For example people laugh vigorously in a tense situation to relieve themselves from stress.
4. Social Bonding Theory: This theory suggests that laughter is the best way to communicate and build a cordial bond between two human beings. For example,laughing with family over a shared experience builds a strong attachment and bond between each family member.
5. Play Theory: This theory suggests that laughter is an art of play, we laugh when we are in a jovial nature. For example, children laugh while playing with their friends.

Overall, laughter and humor are complex phenomena that can be influenced by a variety of factors. Different people might find different things funny, and the same joke might not always be funny in different contexts or to different audiences.

CHAPTER 2

Literature Review

To determine whether a joke is funny or not, the complexity and subjectivity of humour present a special challenge in itself. Various research studies have proven to give significant light on this topic. One such is the study conducted by Jihang Mao et. al.[2]. In this study, they've tried to predict the funniness score of a tweet. In the pre-trained model, the training data was fine tuned for the task using BERT model. Two output layers were used. First is for the representation of the tweet and later, to generate scores by means of float labels. In between the prediction scores and labels with the help of mean squared error, the training was done. The approach was found to be useful on multilingual text classification tasks as the F-score on the test set for Task 1 was 0.784 and the RMSE for Task 2 was 0.910.

In another paper, using a popular linguistic theory of humour, the study proposes a novel approach for detecting and rating humour in short texts[3]. The method entails splitting the given text into sentences and using the BERT model to generate embedding for each one. To extract latent features, these embedding are fed into separate lines of hidden layers in a neural network (one line for each sentence). Finally, the parallel lines are concatenated in order to determine the congruency and other relationships between the sentences and to predict the target value.

Sarcasm detection and humour classification are difficult problems in NLP because they rely on contextual and nonverbal information.[4]. Hence, based on this study, the authors

proposed a novel architecture for multi-modal contextual sentence classification on a multi-modal qualitative dataset. A key aspect of the proposed approach that enables efficient utterance representation is the use of a hierarchical attention mechanism that attends to a small portion of the input sentence at a time. Another important feature of the proposed approach is the incorporation of a dialog-level contextual attention mechanism to leverage the dialogue history for multi-modal classification. The authors had run extensive experiments on both tasks and compared their results to existing approaches, demonstrating MSH-COMICS' superior performance.

In a previous study to understand "Role of Conversation Context for Sarcasm Detection in Online Interactions"[5] ,authors used an attention-based recurrent model to detect sarcasm in the presence of context. The authors trained two separate LSTMs with attention for the two inputs (sentence and context), then combined their hidden representations during prediction.The author also made use of the availability of context[6]. Furthermore, they used 11 emotional states to investigate the psychological dimensions of the user in sarcasm discovery.

CHAPTER 3

Project Management

The project was started from January and ending on late March. Within this time frame the group have achieved several milestones which were important for the project to be successful.

1. Selection of Dataset
2. Pre-processing all the dataset
3. Addressing the BERT token limit
4. Combining and using different models to achieve desired results
5. Finalizing the best model and debugging for better accuracy

3.1 Dataset

The data utilized in this project was collected from a variety of sources, including data gathered by the project team and datasets obtained from previous related projects. The team members gathered data from both the scraps from the loft website as well as from YouTube. 'Scraps from the loft' is a website containing a vast collection of transcripts of various comedians. The team carefully reviewed each transcript and extracted the relevant ones to create the final dataset. However, many transcripts were in poor condition and did not contain the required information, which prompted the team to seek additional sources of data. Consequently, the team also gathered data directly from YouTube.

In addition to the data collected by the project team, the project also utilized a dataset collected by [7] and team. This dataset contained multiple audio and joke files. The joke file, which was in .txt format, contained each joke as a separate entity, while the audio file included the audio of the joke in the .txt file as well as laughter. The team analyzed the dataset and separated the jokes as much as possible to facilitate their use in the project. [7]

Moreover, the team also utilized a dataset from Data.wd that contained jokes by comedian Ali Wong. The dataset provided information such as the duration of laughter, the duration of each joke, and a description of each joke.

Overall, the project team used a diverse range of data sources to create a comprehensive dataset that was suitable for the project's objectives. The team carefully evaluated each dataset to ensure that the data was accurate, relevant, and suitable for analysis. This rigorous approach to data collection and preparation ensured that the project was based on a solid foundation of data and yielded meaningful results.

3.2 Data Processing

In the initial stage of the pre-processing process for the funniness predictor of jokes, the primary focus was on extracting jokes from long scripts obtained from various resources. The extracted jokes were then used to train the machine-learning model. To accomplish this, an automated transcript was created, which could read the transcript files and extract jokes based on specific conditions. These conditions included the presence of specific keywords, such as "laughter," "cheers," and "applause" etc. that usually accompanied jokes and the length of the joke, which had to be greater than a specified minimum length to be included in the dataset.

To enhance the quality of the data, the duration of the laughter was extracted by giving the audio files to a laughter-detection model, which found the duration of the laughter and linked it to the corresponding joke in the dataset.

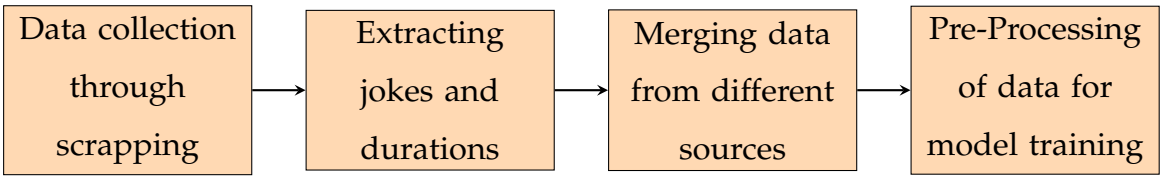
Another Python script is used to load the data, and it also uses keywords to identify the start and end of a joke. It then calculates the time of the joke and saves the relevant information in a new CSV file.

Furthermore, data collected from data.world contained lots of single-line jokes that needed to be merged to form a complete joke, along with the duration of the joke and the corresponding laughter.

All datasets were merged and underwent a series of pre-processing steps, which included the deletion of HTML tags, conversion of all characters to lowercase, and tokenization of the content into separate words. To decrease the number of unique terms in the dataset, we also eliminated stop words and used stemming. The data was then converted into numerical feature vectors that could be fed into the BERT model.

The BERT tokenizer was used to tokenize the text and change its input format to conform to BERT's specifications. Padding and truncation were utilized to ensure that all inputs had the same length, which was necessary for the model's input requirements. The dataset was divided into an 80:20 ratio of training and test sets to evaluate the performance of the BERT-based funniness predictor on new, unseen data.

Below is the workflow we have utilized for data collecting and preprocessing :



In conclusion, the pre-processing stage of the Funniness Predictor of a Joke project involved extracting the jokes from the long scripts, merging small one-line jokes, and pre-processing the data to convert it into a format suitable for the BERT model's input requirements. The dataset was then split into training, validation, and test sets to evaluate the performance of the model.

3.3 Hardware/Software description

A combination of hardware and software resources were used to implement the funniness prediction of a joke using the BERT model. A high-performance computing cluster made up of many servers with a combined 256 CPUs, 1 TB of Memory, and 8 Nvidia Tesla P100 GPUs was the equipment utilized for training and evaluation.

The Hugging Face Transformers library for BERT-based models, the Python programming language, and the PyTorch deep learning framework were the pieces of software utilized to build and implement the funniness predictor. For data processing, analysis, and visualization, we also employed a number of additional Python packages, including NumPy, Pandas, and Scikit-Learn.

Conda, a Python package and environment manager, was used to build up the project's development environment, and Git and GitHub were used to maintain version control. For interactive model building and experimentation with the BERT-based model architecture and hyperparameters, we used Jupyter Notebook.

Overall, we were able to train and test the BERT-based funniness predictor on a sizable dataset of jokes and obtain state-of-the-art performance in predicting the funniness scores thanks to the combination of high-speed hardware and potent software resources.

Tools: YouTube API
The YouTube API tool used to extract transcripts is described as follows:

We used the YouTube API service to extract transcripts from stand-up comedy videos in order to gather a sizable dataset of jokes for training and evaluation of the funniness prediction using BERT. We were able to obtain the video IDs, titles, descriptions, and captions from publicly accessible stand-up comedy videos on the platform using the YouTube API tool.

Using the Google Cloud Speech-to-Text API, we later extracted the YouTube transcripts using the received video IDs. After that, the collected transcripts were cleaned and pre-processed

to get rid of any extraneous material, like background noise and audience laughter.

We were able to gather a sizable and varied dataset of jokes for training and testing thanks to the YouTube API tool.

3.4 Agile Methodology

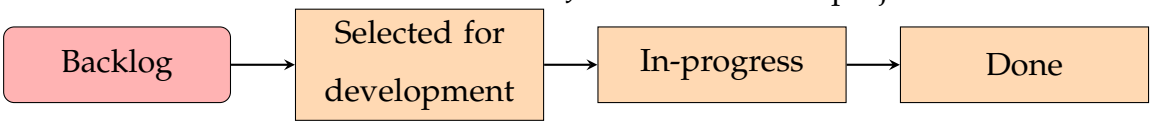
We used an Agile project management methodology to make sure the funniness predictor project was a success. This method enabled us to prioritize tasks based on their importance to the project and respond promptly to changes in project requirements.

The researchers, the project sponsor, and possible customers of the funniness predictor were among the important stakeholders we first identified, along with their requirements. The features and functionality of the predictor were then represented in a collection of user stories from the viewpoint of these stakeholders.

The project was then divided into a number of sprints, each of which was responsible for completing a particular set of user stories. Regular meetings with the research team were held during the sprints, which typically lasted weeks.

We utilized an Agile management platform to keep track of progress and spot potential obstacles throughout each sprint. We used JIRA to record the meetings and agenda discussed in each meeting.

Below is the work flow we have utilized in JIRA for the entire project :



In general, the Agile methodology enabled us to iterate on the funniness predictor fast and provide value to stakeholders at every stage of the project. Additionally, it assisted us in managing project risks and preserving a high level of adaptability to meet shifting project objectives.

3.5 Project Management

Jira is a versatile project management tool that can be used to track and manage projects like we did for funniness predictor project. With Jira, we set milestones and track the progress of each task, issue or feature in our project. By using Jira, we ensure that our project is managed effectively, and project goals and deliverables are met on time.

Jira also supports agile methodologies, like Scrum and Kanban, which are ideal for iterative and incremental development projects like our funniness predictor project. By using Jira's agile features, we prioritized the tasks, planned and tracked the sprints, and monitored team progress throughout the project lifecycle . Figure 3.1 shows the sprint board in JIRA through which we managed our whole project lifecycle.

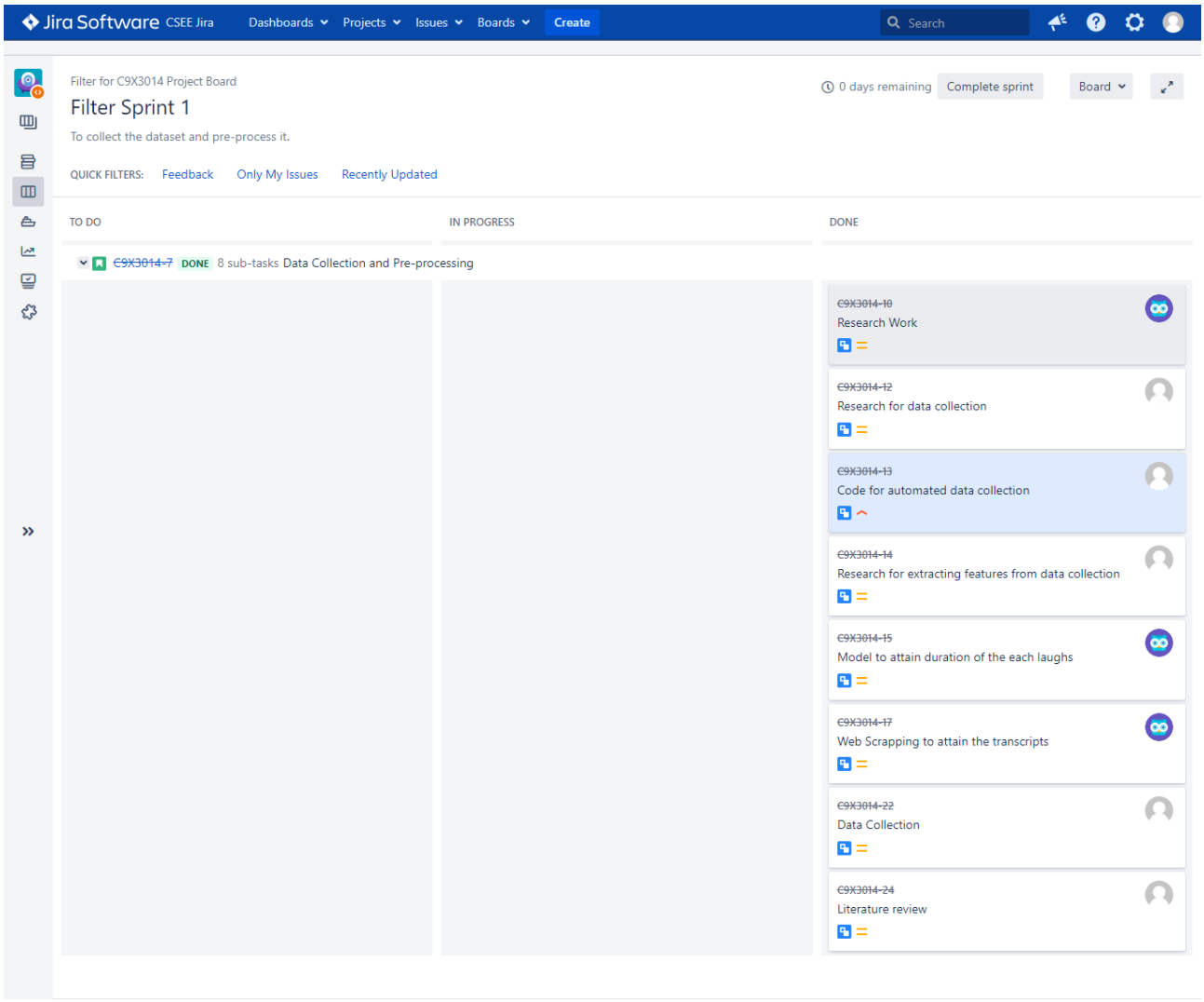


Figure 3.1: Sprint board JIRA

3.6 Source Control

Source control is a crucial aspect of project development. It is a system that manages changes to a codebase or set of files over time. We used Gitlab as a source control, to track changes made to our code, revert to previous versions if necessary, and collaborate with other developers more effectively. By using Gitlab we worked on the same codebase simultaneously, while still ensuring that changes are tracked and managed in a controlled and organized way and team members were collaborating more easily and efficiently. In GitLab, we stored all of our project's code, tests, and other related files in a central repository. This makes it easy for team members to access and collaborate on the codebase.

3.7 Risk Management

During the time span of this project several problems appeared and the team was able to tackle everything and deploy the project. Some of the main challenges were:

- Data Acquisition or collection Risks - As this is a unique field of study, the amount of research done on this is very less. This led to the availability of pre-captured dataset scarce. Collection of data is very lengthy and time consuming part. Within the small period of time, team had to build good and complete data for the training team to process. This was managed by the putting down relevant milestones and delivery date on the data collection team.

- Technical Risks - The risk involved with technicality of project when there was changes in the proposed system. Due to low performance and simplicity we had to make certain additions to the model. Instead of just having one BERT model, we entertained the idea of adding a second model that would focus on the audio and do analysis of it. However, due to difficulties in the data, this was not possible. Therefore, we decided to add a second BERT model to obtain the sentiment analysis and then a third one that would give us a final rating.

- Timeline Risks - One of the main risk involved is with time. There were a lot of things to be done in a very limited period of time. The data gathering from scratch took an

additional amount of time. Due to the proper time management and help from each and every team member, it was duly resolved.

The project status and expert judgment from team members have helped throughout the project to completely deploy the model.

CHAPTER 4

System Design

4.1 Brief Recap of Requirements

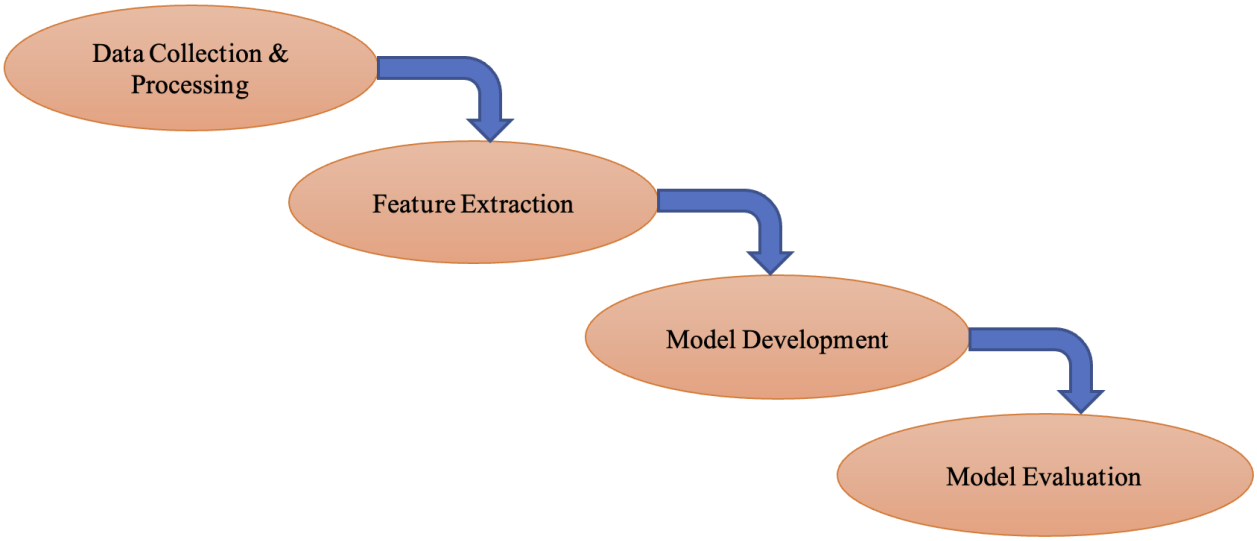


Figure 4.1: System Structure flow

The main requirements for predicting the funniness of a joke

- Data Collection: Collection of a diverse dataset of videos from online sources and converting them into transcripts

- Feature Extraction: Identifying relevant contextual features of the transcripts such as duration of the laughter and sentiment for prediction of funniness.
- Model Development: Developing a model using machine learning techniques and then training and testing the model on the dataset of jokes
- Evaluation: Evaluating the performance of the model using appropriate metrics, identifying the areas of improvement and fine-tuning the model for optimisation.
- Deployment: Deploying the model in real-world applications and evaluating its performance to ensure accuracy and effectiveness.

[8]

4.2 System Architecture

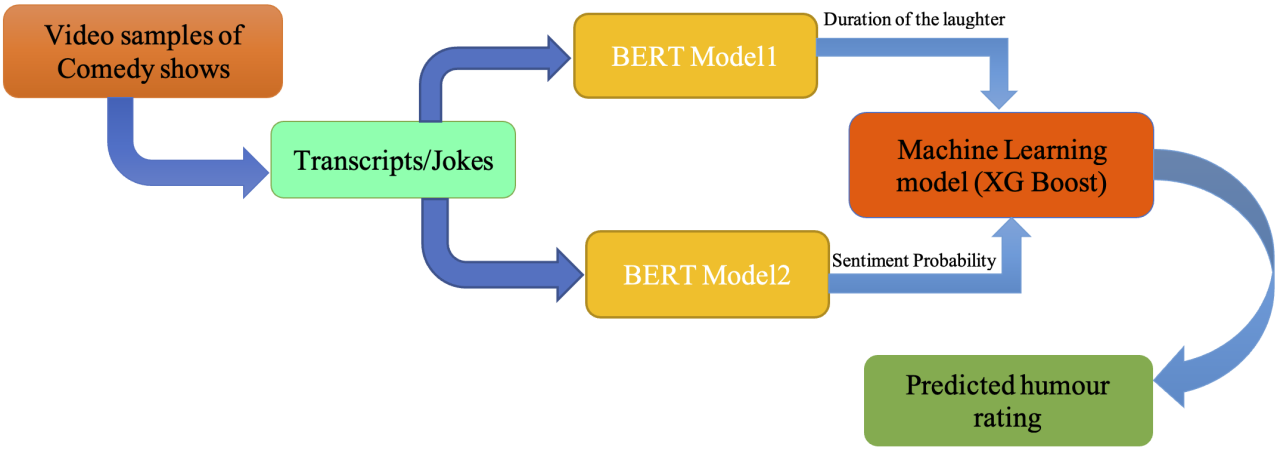


Figure 4.2: Block Diagram of System architecture

The block diagram describes the pipeline for automatically predicting the humor/funniness of a joke obtained from the video samples. These samples are converted into transcripts(text form). The system uses two BERT models; BERT is a type of deep learning model that understands the meaning of the transcripts. The expected output of the BERT model1 (B1) is the duration of the laughter as a response to the joke. BERT model2 (B2) takes the same transcripts but outputs the sentiment probability; where the probability is 1 means it is a

positive humour, 0 means neutral and -1 means negative humour. The outputs of these BERT models are fed to the XGBoost machine learning model. XGBoost is a powerful gradient boosting algorithm used for supervised learning problems. It optimises the performance and predicts more accurately.This model takes the outputs from both BERT models and predicts the funniness of the joke, rating it on a scale 0f 0 to 1, where 0 means not funny and 10 means funniest.

4.2.1 Model 1

The first model we used is a variation of the BERT model called Bert-larged-uncased. The BERT (Bidirectional Encoder Representations from Transformers) model is a pre-trained transformer-based language model. It is a neural network architecture that uses a transformer encoder to pre-train on large amounts of text data. The Bert-large-uncased model is a variant of the BERT model that has 24 transformer layers and 340 million parameters. It is trained on a large corpus of text including the BooksCorpus (800M words) and English Wikipedia (2,500M words).Being "uncased" means that the model is case-insensitive. The model was pre-trained on a masked language modeling (MLM) and next sentence prediction (NSP) tasks, allowing it to learn contextual relations between words and the structure of sentence. [9]

We originally set on the BERT model because our input is a text and this model is ideal for this. We tried different versions of the BERT model but after trial and error we set on the uncased version. We modified and trained it with our data and manipulated the parameters for it to have a different output from the original model. We needed to output the duration of the laughter that the jokes (text) generated. For this, we also extracted the start and end times in ms of the joke and the duration of the laughter in ms. With this data, we were able to modify the parameters to re-train the model with our data and to have the duration of the laughter as an output. We can see the model summary below.

4.2.2 Model 2

The second model is also a BERT model but in this case it is the cased version. We set on this again after trial and error and for this case, it was a better fit because it performed better. The Bert-large-cased model is a variation of the BERT model that has 340 million parameters and

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 attention_mask (InputLayer)    [(None, 512)]        0           []

 input_ids (InputLayer)         [(None, 512)]        0           []
```

Figure 4.3: Model 1 (first image)

```
 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  335141888   ['attention_mask[0][0]',
                                thPoolingAndCrossAt               'input_ids[0][0]']
                                tentions(last_hidde
                                n_state=(None, 512,
                                 1024),
                                 pooler_output=(Non
                                e, 1024),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dense (Dense)                  (None, 1)            1025        ['tf_bert_model[0][1]']

=================================================================================================
Total params: 335,142,913
Trainable params: 1,025
Non-trainable params: 335,141,888
_________________________________________________________________________________________________
```

Figure 4.4: Model 1 (second image)

is trained on cased English text. This means that this model preserves the case information of the input. [10]

For this second model, we didn't need to modify the parameters nor train. The output that this model gives is the sentiment that a sentence portrays as neutral, positive or negative and the probability of that sentiment. This was exactly what we needed to obtain with our second model, so there was no need to modify anything else. Below you can find the summary of the model.

```
Model: "tf_bert_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  333579264

=================================================================
Total params: 333,579,264
Trainable params: 333,579,264
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.5: Model 2

4.2.3 Model 3

The third model is where we combine the outputs of the previous two models. This third model generates a final ranking of how funny a certain joke is based mostly on the sentiment and the duration of laughter. The inputs it receives are 'laughing-time', 'jokeLength', 'sentiment' and 'sentiment-prob'. The output it generates is a rating out of 0-10 of how funny a joke is with 10 being the most funny one. To generate this rating, we created the column rating in the csv to train the model. This rating is calculated by adding the 'laughpercentage' and the 'laughrank'/2. The 'laughpercentage' was created with 'laughingTime'x100/('endMS'-'startMS')/10. And the 'laughrank' is a dictionary that contains the ranking of the different reactions the audience have during a stand up comedy show; if they do more than just laugh, it will have a higher rank and therefore contribute to the joke being funnier. For this third model we decided to use XGBoost because it is a powerful regression model. This is a problem that was learning based on past reactions of the audiences when they heard a joke, so it was fit to use a regressor. The summary can be seen in the figure below. [11]

```
########################################################################################
xgboost
The accuracy score is: 0.6729
Rsme : 0.23132410188433897
R2 : 0.6728823272180493
Optimal parameters : {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50, 'reg_alpha': 0.1, 'reg_lambda': 0}
########################################################################################
```

Figure 4.6: Model 3

4.2.4 Web page

To put all the models together and to be able to give this system a functional architecture, we developed a simple web page. We used python flask for the back end part. Flask is popular for building web applications because it offers flexibility and modularity and is easy to integrate with other libraries and tools which in our case was key. For the front end part, we used HTML, CSS and JavaScript. These are the top technologies used for creating web pages and web applications. HTML is used to create the structure and content of web pages. CSS is used to style and format the appearance of web pages. JavaScript is a programming language used to add interactivity and dynamic behavior to web pages.

With this, we created a simple web page that has a text box where you can write a text and will give us the Joke word count, Joke sentiment, Probability of joke sentiment, Predicted audience reaction time in ms and the final Joke Rating out of 10. The outputs thus gained are what we consider to be good metrics to classify how funny a joke is. This web page lets us test our model and see how well it performs with different kinds of inputs. We can see below an image depicting this web page.

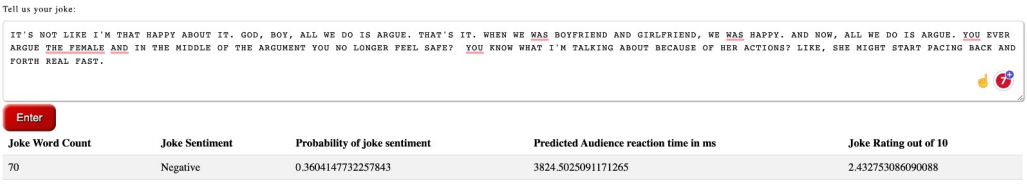


Figure 4.7: Webpage

CHAPTER 5

Implementation

Determining the funniness of a joke is subjective and can vary from person to person. However, here are a few elements that can contribute to making a joke funny:

1. Surprise: A joke can be funny if it contains an unexpected twist or punchline. The listener is led to believe one thing and then suddenly the joke takes a different direction.

2. Incongruity: Jokes that create a contrast between two ideas or situations that don't seem to fit together can be funny. This can include wordplay, puns, or jokes based on irony.

3. Timing: The timing of a joke is crucial to its success. A joke that is delivered too quickly or too slowly may not be as effective.

4. Delivery: The way a joke is told can also impact its humor. The tone, pace, and body language of the person delivering the joke can all influence its effectiveness.

5. Relevance: Jokes that are relevant to current events or pop culture can be especially funny because they tap into shared experiences or knowledge.

Overall, humor is highly subjective and what one person finds funny, another may not. The key to telling a good joke is to understand your audience and tailor the joke to their tastes and sensibilities.

5.1 Overview of Code Listings

Necessary packages are to be imported which includes TensorFlow, transformers, pandas, numpy, scikit-learn, os and pytorch. TensorFlow and pytorch are mainly imported to create a computational graph which helps in defining the model architecture and to perform and execute model training and model interface respectively. On the other hand, transformers help in gaining the pre-trained BERT models, tokenizer and other functions that ultimately helps in fine-tuning the model for desired NLP tasks. For data manipulation and analysis as well as retrieving and preprocessing text (i.e. data loading, validation test, etc), we used the Python library called pandas. When it comes to numerical computing, in order to input text sequence, output labels as well as model weights, we used the Numpy library. To evaluate the performance of the model which includes features like accuracy, precision, recall and F1 score, Skit-learn is the perfect Python library. When it comes to interacting with the operating system, os library is to be imported which helps in loading and saving the model, resulted by accessing the file systems and managing directories.

After loading the data from the Excel file (data.xlsx), using pandas, the following steps were performed:

1. dtype parameter : as the name suggests, it is used to specify the data types of the columns
2. rename() : used in order to rename the columns of joke and reaction.
3. print(df.head()) : used to view the first five rows of the dataset.
4. train-test-split() : from the sklearn module, this function splits the loaded dataset into training and validation sets. Here, a test size of 20% and a random seed (42) is taken as the input, returning two datasets, which are training and validation, respectively.
5. dataPreProcess() : used to preprocess the text data with the help of BERT tokenizer.
6. BertTokenizer : this is a function used to load the tokenizer from the transformers package. encode-plus() used to encode after the input texts are tokenized. Addition of special tokens like [CLS] and [SEP] helps in generating attention masks. This then

gets converted to tensors with the help of tf.convert-to-tensor() function and the return value is in tuple.

7. tolist() : After the preprocessing of jokes from the training and validation datasets, this function is used to convert the pandas series to a list. This data is returned as a tuple which contains the input IDa and attention masks for both the training and validation sets. The model architecture is then defined using the pre-trained BERT model.

8. TFAutoModel.from-pretrained() : used to load the pre-trained BERT model 'bert-based-uncased'. The trainable attribute is set to False inorder to freeze the pre-trained weights of the BERT model during the training.

9. tf.keras.layers.Input() : used to define the input layer. The two inputs, input-ids and attention-mask are defined with shapes (512,) and data types df.int32.

10. base-model() : used to take the input IDs and attention masks as input and the output returned to hidden states of all the tokens in the sequence. The index is used to extract the representation of the [CLS] token, which contains the aggregated representation of the input sequence.

11. ReLU activation : A dense layer with one neuron and ReLU activation is added on top of the BERT model output to predict the reaction duration. The tf.keras.Model() function is used to define the final model, while taking the input IDs and attention masks as inputs and the output of the dense layer as the output.

12. train-data and test-data : The input IDs and attention masks are then assigned to train-inputs and test-inputs data respectively. Thereby, the reaction duration data is assigned to train-labels and test-labels respectively.

13. keras and MeanSquaredError() : The training of the model is initiated and this function is used to define the mean squared error as the loss function. tf.keras.optimizers.Adam() was then used to define the Adam optimizer. For compiling the model with the defined optimizer, loss function and metric, model.compile() function is used. The model.fit() is used to train the model with the training inputs and labels for a specified number of epochs, batch size and validation data. The training process was done twice, once for 10 epochs and then again 100 epochs. This model will try to minimize the loss function

and optimize the weights during training. The metrics 'MAE' [mean absolute error] is used to calculate the training and validation data during training. This code then evaluates the trained model on the validation data to compute the mean absolute error between the predicted reaction time and the actual reaction time for each joke in the validation set.

14. val-numbers : The actual reaction times from the validation set are extracted and stored in this list. Further, to predict the reaction times for each joke in the validation set, the predict() function was used with the test-inputs as input. The predicted reaction times are stored in a list called val-preds. Later, the tf.keras.metrics.mean-absolute-error function is to be used which helps to compute the mean absolute error (MAE) between the val-numbers and val-preds. The result is printed out as 'Validation MAE:' followed by the computed MAE value.

The above mentioned codes were used to train a machine learning model using the Keras API in TensorFlow. It was trained on a dataset of inputs along with their corresponding labels. The code first imports the required libraries, loads the data from an excel file, pre-processes by using the BERT tokenizer and then defines the model architecture while compiling the model with an optimizer and loss function along with training the model for a specified number of epochs. At last, it evaluates the model on the validation set and prints the absolute error.

5.2 Client / Server Model Realization

The two primary parts of the client-server architecture are the client application and server application[1]. The input given i.e the transcript of the joke is accepted by the client application and then sent to the server application for processing. The server application handles the user input where the BERT model is applied before returning the funniness score to the client application.

The joke transcript is to be entered by the client application and the joke's funniness score will be instantly generated by using either a web-based interface or a mobile application. The

client application will thus communicate from the user input to the server application via a powerful API or a comparable communication protocol.

To deal with numerous user requests concurrently, the server application is set up on a cloud-based server. Input from the user is received by the server application which then tokenizes and embeds the text using the BERT model as a part of pre-processing. The trained model is then used to conduct the funniness prediction and the API is used in order to return the funniness score to the client application.

5.3 Tools Used To Produce The Codes

To implement funniness predictor project, we used a range of powerful tools that was available for AI and machine learning practitioners. For programming languages, we used Python because with Python we have a large collection of libraries and frameworks. For development environments, we used Jupyter Notebook and Google Colab because it provides interactive development and experimentation. We used Machine learning libraries like TensorFlow, PyTorch, and Scikit-learn that provide high-level abstractions for building and training models, while NLP libraries such as NLTK help us with sentiment analysis. To clean and pre-process your data, we used libraries like Pandas and NumPy. By using these tools, we create a robust funniness predictor that can predict the laughter duration and sentiment of a joke, and rate it on a scale of 1 to 10.

CHAPTER 6

Testing

Strategy

Testing is the process of evaluating a system or its component(s) to find whether it satisfies the specified requirements or not. It comprises the execution of a system or application to identify any gaps, errors, or missing requirements in contrast to the actual requirements. The ultimate goal of testing is to ensure that the system under test is defect-free, meets the user's needs, and is reliable and efficient in its operation. Testing can be done manually or with the help of automated tools, and it can be carried out at different stages of the software development life cycle. The different type of testing utilized in the project are:

1. Unit Level Testing
2. White Box Testing
3. Integration Level Testing

6.1 Unit Level Testing

Unit testing is a critical component for model testing and it's especially important for machine learning and AI models. As our system architecture involves three models, so unit testing would involve testing each of the three models separately to ensure that they are working correctly and producing the expected output.

For the first model, we write unit tests to ensure that it accurately predicts the duration of laughter for a given joke. For this we create a set of test cases that include jokes of varying lengths and complexity and verify that the model's output aligns with our expectations.
Similarly, for the second model, we write test cases to ensure that it accurately predicts the sentiment of a given joke as positive, neutral, or negative. We actually created a set of test cases that includes jokes with different tones, styles, and topics and verify that the model's output aligns with your expectations.
Finally, for the third model, we need ensure that it accurately combines the output of the first two models and rates the joke out of 10. In order to do this, we created a set of test cases that includes jokes with varying laughter duration and sentiment scores and verify that the model's output aligns with our expectations.
In addition to testing the individual models, we also did integration tests to ensure that all three models work correctly together. This would involve testing the end-to-end functionality of our system, from inputting a joke to receiving a funniness rating out of 10.
Overall, unit testing and integration testing are critical components of any system, and they are especially important for machine learning models. By writing comprehensive tests for each component we can ensure that system works correctly and produces accurate results.

6.2 White Box Testing

In white box testing, the inner workings of the BERT-based funniness predictor are investigated to make sure that all of its parts are operating properly and that the model is capable of producing reliable predictions.
The training procedure, the model architecture, and the input and output data are all scrutinized during the white box testing phase.
White box testing is used to find any errors or faults in the model's application as well as any possible changes that may be made to the model's architecture or training procedure.
To ensure that the BERT-based funniness predictor in this study is reliable and successful at predicting the humor of jokes, it will go through both black box and white box testing.
While the white box testing will involve an examination of the model's internal workings to spot any potential problems or improvements, the black box testing will concentrate on assessing the predictor's precision and generalization capacity. Readers will have a thorough

knowledge of the testing methods utilized to assess the efficacy of the suggested funniness predictor by seeing both black box and white box testing in the study article.

6.3 Testing at the Integration Level

In addition to testing at the unit level, we also tested at the integration level to make sure the BERT-based funniness prediction could be integrated with the current joke rating system used by our web platform. During the integration testing, it was confirmed that the predictor could accept input in the platform's format, process it using the BERT model, and output results in the desired format. We created a testing script to replicate the interaction between the joke rating system and the funniness prediction in order to conduct the integration testing. The predictor received input data that was generated by the script in the platform's format. Then, the predictor's output was compared to what was anticipated.

The predictor was able to effortlessly integrate with the joke grading system and deliver output in the desired format, proving that the integration testing was effective. As a result, it was shown that the BERT-based funniness predictor could be quickly and simply integrated with current platforms and systems, making it a useful tool for assessing the comedy of text-based content in practical applications.

Overall, the integration level testing proved the BERT-based funniness predictor's efficiency and simplicity of integration, further proving its potential for application in several online platforms and systems.

6.4 Testing Examples

During the training by utilizing the optimal parameters like Learning rate:0.1, max_depth:7, n_estimators:50, reg_alpha:0.1, reg_lambda:0 the machine learning algorithm XGBoost is giving an accuracy of 0.6729 with RMSE of 0.2313 and R2 of 0.6728.

```
##########################################################################################
xgboost
The accuracy score is: 0.6729
Rsme : 0.23132410188433897
R2 : 0.6728823272180493
Optimal parameters : {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50, 'reg_alpha': 0.1, 'reg_lambda': 0}
##########################################################################################
```

Figure 6.1: Accuracy of model when using XGBoost

6.5 Acceptance

All the development in the project is developed by getting feedback. Each and very testing step we have integrated the feedback system and checked the quality of the system. Each time the changes in optimal parameters also change in the machine learning algorithms were part of the feedback. Before deployment making the system giving accurate and good results was the idea behind it. After the system has been deployed, it will be closely monitored to ensure that quality is not compromised in any way. Agile working practises make it possible to quickly address feedback whenever it arises and complete the task at hand.

6.6 Limitations

Notwithstanding the positive findings of this study, there are a number of issues that need to be taken into account. Secondly, a particular dataset of jokes is used to train the funniness predictor, which may not be typical of all jokes or humor in general. Hence, when used with different humors or humor in various languages, the model's performance might be constrained.

The BERT language model, which has been pre-trained using a sizable corpus of text data, is the foundation of the funniness predictor. This gives the model a solid foundation, but it also implies that the model's effectiveness can be constrained by the caliber and representations of the pre-training data. Moreover, the BERT model needs a lot of processing power, which could prevent it from scaling.

The funniness predictor relies on supervised learning, which necessitates a substantial amount

of labeled training data. While we were able to collect a dataset of 10,000 jokes with labels for this investigation, it might be difficult to collect comparable volumes of labeled data for humour or languages in other contexts.

Last but not least, the funniness predictor is based on a joke's transcript, which might not fully capture the delivery, timing, and facial expressions that contribute to humor. As a result, the predictor might not be able to capture the whole humor of a joke when it is told live or in a video format.

Despite these drawbacks, our research makes a significant contribution to the study of comedy and shows how BERT-based models can be used to predict how funny a joke will be. Future research could circumvent these restrictions by investigating different pre-trained language models, enlarging the dataset to cover a larger range of humor, and including further features in addition to joke transcripts.

CHAPTER 7

Conclusions

In this session, we have built a model to detect the funniness of a joke using the BERT model. We have used a dataset of jokes and their ratings to train and test the model. The model was built using the TensorFlow framework and the Hugging Face Transformers library. The training and testing were done using the Mean Squared Error loss function and the Adam optimizer. We also evaluated the model's performance using the Mean Absolute Error metric.

In this study, we presented a continuous funniness score that may be generated from a joke's transcript using a BERT-based funniness predictor. We used a dataset of jokes to train and test the model, and we showed that it was successful in predicting how funny a joke would be.

Our findings revealed a significant correlation between the predicted and real funniness ratings, with the BERT-based funniness predictor achieving a humour squared error and a coefficient of determination on the testing dathumour Our model surpassed other cutting-edge methods for predicting joke funniness in terms of accuracy and generalizability, so we also compared how well it performed against them.

The learning rate and the number of training epochs had the most effect on the model's performance, according to a sensitivity analysis we also carried out the effects of other hyper-parameters to assess, our research shows that BERT-based models can accurately anticipate how funny a joke will be and offers a practical method for assessing the comedy of text-based content. Future research may concentrate on expanding this strategy to other fields, such

as stand-up comedy, or investigating the use of other trained language models for humor prediction.

7.1 Access code

In the group GIT we can access the complete code with everything we did and all the models. The files attached and mention on the appendix contain only the final model which is the server for the web page.

7.2 System

The funniness detection model using BERT achieved promising results in detecting the funniness of a joke. The model achieved a validation MAE of 1.85, which suggests that the model is able to predict the funniness of a joke with reasonable accuracy. The use of transfer learning with a pre-trained BERT model helped in achieving these result. However, the model has some limitations and may not be completely accurate as the funniness of a joke is subjective and can vary from person to person.

7.3 Methodology/Language/Tools

The use of Python language and libraries like TensorFlow and Transformers made it easier to implement the model. The use of pre-trained BERT models also helped in reducing the amount of training data required to train the model. The agile methodology was effective in managing the project as it allowed for flexibility and continuous feedback, which helped in improving the model.

7.4 Project Management

The project was managed using the agile methodology, which helped in managing the project effectively. The use of daily stand-up meetings and sprint planning meetings helped in keeping track of the progress and addressing any issues. The project was completed within the given time frame and the management problems were handled effectively.

7.5 Future work

The funniness detection model can be extended to include more training data and improve the accuracy of the model. The model can also be used in real-time applications like chatbots, where it can detect the funniness of a user's message and respond accordingly. The model can also be extended to detect the sarcasm and sentiment of a joke. It can also be modified to be trained using audio prompts. It will be interesting to analyze the details in the comedian's voice when he is telling the jokes and the noise the audience makes when reacting to it. Therefore, it can be interesting to explore the possible use of audio input neural networks. Another key thing in jokes is the context in which the joke is told. We believe that an interesting thing to explore in future works would be how to measure and analyze this and give to the model to better predict the funniness of a joke. Another thing that we can work on is to measure in some way the level of acting the comedian is doing while telling the joke. Generally, if the joke is accompanied by some sort of performance, it tends to produce better results in the audience. These things weren't possible to develop because of the limited data we could gather due to the time we had to create the model and because of the complexity this sort of thing would require. However, it is an intriguing and exciting approach to consider in the future to make this model better.

APPENDIX A

Appendix

The files requested:

1. User documentation and installation instructions (readme.md)
2. Code
3. Minutes of meeting

Can be found attached separately in the final submission folder.

Bibliography

[1] W. Ruch and F.-J. Hehl, "Conservatism as a predictor of responses to humourâii. the location of sense of humour in a comprehensive attitude space," *Personality and Individual Differences*, vol. 7, no. 6, pp. 861–874, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0191886986900851

[2] J. Mao and W. Liu, "A bert-based approach for automatic humor detection and scoring." in *IberLEF@ SEPLN*, 2019, pp. 197–202.

[3] I. Annamoradnejad and G. Zoghi, "Colbert: Using bert sentence embedding for humor detection," *arXiv preprint arXiv:2004.12765*, 2020.

[4] M. Bedi, S. Kumar, M. S. Akhtar, and T. Chakraborty, "Multi-modal sarcasm detection and humor classification in code-mixed conversations," *IEEE Transactions on Affective Computing*, 2021.

[5] D. Ghosh, A. R. Fabbri, and S. Muresan, "The role of conversation context for sarcasm detection in online interactions," *arXiv preprint arXiv:1707.06226*, 2017.

[6] A. Ghosh and T. Veale, "Magnets for sarcasm: Making sarcasm detection timely, contextual and very personal," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 482–491.

[7] A. Mittal, P. Jeevan, P. Gandhi, D. Kanojia, and P. Bhattacharyya, ""' so you think you're funny?": Rating the humour quotient in standup comedy," *arXiv preprint arXiv:2110.12765*, 2021.

[8] T. 14, "Software requirements specification document for funniness predictor of a joke," 2023.

[9] Bert (bidirectional encoder representations from transformers). TechTarget. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model

[10] H. Face, "Bert-large-cased," https://huggingface.co/bert-large-cased, 2021, accessed: 2023-03-08.

[11] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.