PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

6th Semester Project Report on

Context Analyzer

Submitted by

Harsha K Y (PES1201801839)

Jan – May, 2020

under the guidance of

Mr. Tamal Dey

Associate Professor

Department of Computer Applications

PES University, Bengaluru - 560085

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER APPLICATIONS

PROGRAM – MASTER OF COMPUTER APPLICATIONS

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER APPLICATIONS
PROGRAM – MASTER OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project entitled

Context Analyzer

is a bona fide work carried out by

Harsha K Y (PES1201801839)

in partial fulfilment for the completion of 6th semester project work in the Program of Study MCA with specialization in Data Science under rules and regulations of PES University, Bengaluru during the period Jan. 2020 – May 2020. The project report has been approved as it satisfies the 6th semester academic requirements in respect of project work.

Internal Guide	**Dean - Faculty of Engineering Technology**	**Chairperson - Dept. Of Computer Applications**
Mr. Tamal Dey, Associate Professor, Department of Computer Applications	**Dr. Keshavan B K**	**Dr. Veena S**

Name and Signature of Examiners

Examiner 1	**Examiner 2**	**Examiner 3**

DECLARATION

I, **Harsha K Y**, hereby declare that the project entitled, **Context Analyzer**, is an original work done by us under the guidance of **Mr. Tamal Dey**, Professor, PES University and is being submitted in partial fulfilment of the requirements for completion of 6th Semester course work in the Program of Study **MCA**. All corrections/suggestions indicated for internal assessment have been incorporated in the report. The plagiarism check has been done for the report and is below the given threshold.

PLACE: Bengaluru

DATE:

HARSHA K Y
PES1201801839

ACKNOWLEDGEMENT

The satisfaction and euphoria are that successful completion of any task would be incomplete without mentioning the people who made it possible.

I would like to express my deep sense of gratitude to respected Vice Chancellor of PES University, **Dr. Suryaprasad K**, for giving the opportunity to work on this project.

I take this occasion to thank my sincere and heartfelt thanks to Dean, Faculty of Engineering and Technology **Dr. Keshavan B K, PES University** and Chairperson, Department of Computer Applications **Dr. Veena S** for their motivation, support and for providing a suitable working environment.

With a great pleasure, I express my sincere gratitude to my guide **Mr. Tamal Dey, Associate Professor, Department of Computer Applications**, for providing me with right guidance and advice at the crucial junctures which helped me in completing the project work on time. I am wholeheartedly thankful to her for giving me valuable time, suggestions and for showing me the right way in completing my project successfully.

I extend my sincere thanks to our project **coordinator Mr Tamal Dey, Assistant Professor, Department of Computer Applications,** for providing schedule and timelines and documenting information about project.

I also thank other faculty members and friends at this occasion.

CONTENTS

Page No.

LIST OF FIGURES

LIST OF TABLES

Page No.

ABSTRACT

Text classification is a very important task in supervised machine learning. A piece of text is assigned to one or more classes or categories. This can be done manually or with the help of powerful machine learning algorithms. The problem with doing this manually is that it takes up a lot of time and resources. Let’s say you own a blogging website or a news website. Every article that is being posted has to be classified and put into a category. Making people read these articles manually is both time consuming and expensive. It would be easier if the computer itself classified these articles, as soon as they are posted. This is where natural language processing comes into play. Natural Language Processing or NLP, is a Machine Learning (ML) task that is used to train an ML model to recognize text data and get meaningful insights from it. This means that a trained ML model will be able to go through some text data and give us some context on it. So, if you pass an article as input, this model will be able to tell you where it belongs. NLP can also be used to do other interesting tasks such as Sentiment Analysis. This means that a model will be able to tell if some text data is positive, negative, or neutral about any topic that is in discussion. Context Analyzer provides solutions to both of these tasks.

1. INTRODUCTION

1.1. Project Description

The goal of this project is to provide Natural Language Processing (NLP) services in the form of APIs. NLP is one of the major tasks in Machine Learning (ML). Whenever someone finds themselves wanting a computer program that can carry out NLP tasks for them, they would have to go through very time consuming and difficult processes of data collection, data cleaning, finding people skilled enough to do something with that data, build an effective model that can ultimately be used to do the NLP tasks. Instead, an individual or an organization can make use of the APIs provided as a result of this project do these tasks. All they have to do is make API calls with their inputs and the API returns a JSON object with predictions. This is going to save the users months of time and effort. Users can make calls using any language that supports fetching JSON data from the browser. This gives them incredible flexibility while building their applications. It also allows them to focus more of their manpower on their main products. There is always a rising need for machines more powerful, intelligent, and faster than us to take over the tedious and repetitive tasks that we just don’t want to do. ML has provided solutions exactly to these tasks over the years. If someone is conducting a survey on a certain topic, they would not want to spend hours looking at tweets and determining the general sentiment about that topic. They would rather feed that information to a program and let it tell them what kind of sentiment the tweets are showing. The tricky part is getting that program to work properly and accurately. The project uses three different models, trained for three different tasks (Sentiment Analysis, Category Prediction, Spam Detection). All of them averaging an accuracy of over 94%. So, accuracy isn’t a problem anymore. The project was built on node.js using a tensorflow.js integration. This means most of the tasks are done on the browser itself, making it faster and requiring very less computational power.

1.1.1. Problem Definition

Text classification is a very important task in supervised machine learning. A piece of text is assigned to one or more classes or categories. This can be done manually or with the help of powerful machine learning algorithms. The problem with doing this manually is that it takes up a lot of time and resources. Let’s say you own a blogging website or a news website. Every article that is being posted has to be classified and put into a category. Making people read these articles manually is both time consuming and expensive. It would be easier if the computer itself classified these articles, as soon as they are posted. This is where the need for Natural Language Processing arises. Natural Language Processing or NLP, is a Machine Learning (ML) task that is used to train an ML model to recognize text data and get meaningful insights from it. This means that a trained ML model will be able to go through some text data and give us some context on it. So, if you pass an article as input, this model will be able to tell you where it belongs. NLP can also be used to do other interesting tasks such as Sentiment Analysis. This means that a model will be able to tell if some text data is positive, negative, or neutral about any topic that is in discussion. Our phones and email accounts are bombarded with spam every day. The only way to filter out the spam is by either making users flag the messages as spam manually or filter out the messages at the server end itself using an effective program. Context Analyzer provides solutions to all three of these tasks.

1.1.2. Purpose

The problem of classifying text can be done by the organizations or the users themselves. To do that they would have to start collecting data, hire skilled ML engineers, spend a lot of money and time building out an effective and accurate model. Then some more time has to be spent tuning the model to perform better on all kinds of data. All of this can be avoided when companies just use the APIs provided by this project. The purpose of this project was to automate cartain NLP tasks and also provide them as services through APIs. Integrating APIs is much cheaper and faster than building an entire team to carry out these tasks.

1.1.3. Scope

The main users of this project will be those looking to integrate NLP tasks into their application with very little human effort or resources and get very efficient results. The web application also provides a UI which can be used to carry out these tasks for one time users. The sentiment analysis API can be used for predicting sentiment of data from social media platforms, reviews on products, etc.. Category prediction is a multi-class classification task that can be used on news articles or blogs to classify them into different classes like politics, entertainment, sports, health, etc.. The spam detection API allows users to determine whether a message (any form of message; like SMS, email etc.) is spam. Initially I identified these three tasks as the major ones. The application is built in a way that it is always easy to add new features or build new APIs and add them.

1.1.4. Proposed Solution

The application consists of three different models all built using a Convolutional Neural Network, or CNN. The three models are trained on; a news dataset, the IMDB reviews dataset, and an SMS Spam dataset. The news dataset is going to be used to train the model for the multi class classification task. The IMDB reviews dataset is going to be used to train the model for sentiment analysis. The SMS spam dataset is used to train the model for spam detection. These models, once put into production will be able to do these classification tasks in mere seconds. This will also be cheaper and more effective.

To provide APIs, Node.js will be used. This has very good integration for tensorflow.js. Which means all ML tasks can be done on the browser itself. Node.js is also an amazing javascript runtime that can be used to build highly efficient endpoints. The web application can also be used as an external tool for the classification tasks. Once the dataset is loaded, the preprocessing starts.

First I use the stop words function to remove the prepositions like “the, of, he, she etc”. It also simplifies the words, for example if there are multiple words like “doing, did, done” it will be converted to “do”. Then the tokenizer is used to turn the text into sequence of numbers. The neural network takes only numbers as input. Once the preprocessing is done, the model is created with several layers. The output layer returns a value denoting which class the text belongs to. Once the model is trained and saved, tensorflow js is used to load it. After this, everything is done on the browser. The predictions are made at the node endpoints.

2. LITERATURE SURVEY

2.1. Background Study

A lot of research goes into NLP almost every day. Providing APIs for NLP tasks is a bit of a complex task. One such way to build NLP APIs is using a robust back-end technology like Node.js and making use of its brilliant integration with Tensorflow.js. Tensorflow.js allows us to interact with ML models directly from the browser. This makes it currently one of the best libraries for building NLP APIs. Tensorflow.js, for a while was running only on experimental Node.js. Recently it was ported to latest stable build of Node.js. It shows that this is a good way to move forward.

2.1.1. Existing Systems

Google Cloud Natural Language provides NLP APIs. They also make use of tensorflow.js to provide the service. OpenNLP and Stanford NLP provide NLP libraries that can be integrated with supported languages. TextRazor provides NLP APIs but they do not use tensorflow.js. All these systems have certain drawbacks and gaps that can be filled with Context Analyzer API. The drawbacks and the solutions for them are mentioned in a future section.

2.1.2. Related Work

Title: Recurrent Neural Network for Text Classification with Multi-Task Learning

Authors: Pengfei Liu, Xipeng Qiu, Xuanjing Huang

Summary: Neural network based methods have obtained great progress on a variety of natural language processing tasks. However, in most previous works, the models are learned based on single-task supervised objectives, which often suffer from insufficient training data. In this paper, we use the multitask learning framework to jointly learn across multiple related tasks.

Based on recurrent neural network, we propose three different mechanisms of sharing information to model text with task-specific and shared layers. The entire network is trained jointly on all these tasks. Experiments on four benchmark text classification tasks show that our proposed models can improve the performance of a task with the help of other related tasks.

Title: Long Short-Term Memory

Authors: Sepp Hochreiter, Jurgen Schmidhuber

Summary: Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is O(1). Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

2.1.3. Drawbacks of Existing Systems

Unlike Google Cloud Natural Language, this project allows users to customize the API results however they want. The response from the API is a JSON object, which means users can fetch any data they want from the output easily. Also, making API calls is easier than ever as all it requires is the input to be appended to the URL. (for ex: “localhost:3000/api/sentiment?predict=your+input”). A simple GET request would give you the predictions in a JSON response. While Open NLP and Stanford NLP provide good libraries, it wont be easy for every user to download the library and write code to make use of it. It is much easier to just make API calls and get output almost instantly. The major problem that Context Analyzer is solving is the ease of use and integration. As stated above, making API calls is very easy and simple for everyone to understand. The JSON response is also very easy to decode and every part of the response can be fetched individually based on the user’s requirements.

2.2. Feasibility Study

2.2.1. Technical Feasibility

The application runs completely on the browser. This does not require any additional installations or downloads. It is extremely easy to add new APIs or features when built. The back-end uses all the es6 features of Node.js which means it is highly responsive and fast. Obviously, the development part is hidden completely from the user. All the user will see is a responsive UI.

2.2.2. Economic Feasibility

The development of this web application barely costs anything. It can be built with computers that have a decent amount of computational power. I decided to use the CPU version of tensorflow and tensorflow.js. This means it can be built using devices without a GPU. All the tools and technologies used were open-source, so, no licensing was required.

2.2.3. Operational Feasibility

Operating the web application is also very easy. The web application is built completely on the browser, so it will be fast and responsive. The application also contains documentation for the APIs. Every new user can go through it and find out how to use the APIs. One time users can also make use of the UI provided for each of the APIs. This takes in the user input and returns predictions. This is effective to show as a demo for the APIs and gives the users an idea of how quickly the results are given.

2.3. Tools and Technologies

2.3.1. Tensorflow / tfjs-node

Tensorflow is an end-to-end open-source platform for ML. It has a wide range of tools and libraries, along with community resources that help us build state-of-the-art ML powered applications. In the context of this project, tensorflow was used to build and train the models. **Tensorflow.js** is an ML library for JavaScript. It allows us to develop ML models in JavaScript and use ML directly in the browser or in **Node.js.** This application made the most use of tensorflow and tfjs-node as it was the core component in building the APIs and models.

2.3.2. Node.js

Node.js is an asynchronous event-driven JavaScript runtime. **Node.js** allows us to build scalable network applications. **Node.js** served as the main back-end for this project for its **REST API** and **asynchronous** capabilities. A lot of asynchronous code is written to support the loading of models and making predictions. **Node.js** powers all of these tasks effectively.

2.3.3. Python 3

Python is a powerful and fast programming language. Python is user-friendly and easy to learn. Python basically runs on any platform and is open-source. Python was largely used as the platform for building and training the models. Tensorflow integrated with Python is the most powerful tool for ML. Python’s endless libraries facilitate users to perform almost any task.

2.3.4. Postman

Postman is an API development tool that helps users test and build powerful APIs. From headers to authentication tokens to simple JSON values, Postman supports all these features. It was largely used in the development part of the web application. All the routes and requests were simulated, monitored, and verified using Postman. Only then would they be integrated with the front-end.

2.3.5. Handlebars.js / hbs

Handlebars.js is used to build semantic templates. Handlebars is compatible with **Moustache** templates as well. Handlebars templates are compiled into JavaScript functions. These templates allow for reusability and dynamic rendering of content. **Hbs** is an express.js view engine for **Handlebars.js.** Hbs can be used with express.js to seamlessly write handlebars templates and integrate them with the HTML code. Hbs supports Partials and Views. Views are your main HTML pages and Partials support rendering of partial content on the views. This makes for building an effective, dynamic, and lightweight front-end.

3. HARDWARE AND SOFTWARE REQUIREMENTS

3.1. Hardware Requirements

- **Development Environment:**
    - **HDD:** 4GB
    - **OS:** Ubuntu 18.04 LTS, Windows 10, Mac OSx
    - **Processor:** Intel i5 Gen 8
    - **RAM:** 8GB

3.2. Software Requirements

- **Server Side:** Node.js 13.x, Express 4.x, tfjs-node 1.5.x
- **Client-Side:** Handlebars 4.x, HTML5, CSS3
- **Environment:** Ubuntu 18.04, Windows 10, Node, Python runtime environment
- **Code Editor / Ide:** VS Code, Jupyter Lab

4. SOFTWARE REQUIREMENT SPECIFICATION

A software requirement specification gives a detailed description of a software system along with its functional and non-functional requirements.

4.1. Users

Users are the people who interact with the application. They might be making use of the API or using the web UI to get predictions.

API User is the user who uses the API to get predictions. They make API calls from their own application and make use of the response.

One-time User is the user who uses the web UI of the application to get predictions for their own input.

4.2. Functional Requirements

4.2.1. Data Collection

- Deciding what kind of data is required for building the models.
- Downloading pre-collected data from Kaggle, and other sources.

4.2.2. Data Preparation and Pre-processing

- Building a dictionary of words from the dataset. This step is important in the pre-processing section of the project as the dictionary is used to get more refined and accurate predictions. This dictionary has the mapping between normalized words and their integer IDs. The gensim.corpora.Dictionary class is used.

- Removing unnecessary fields from the dataset. There might be some unnecessary fields like ‘id’, ‘date’, etc., these should be removed if not necessary for building the model.
- The text data is converted into a list of tokens. The neural network takes only text data as input. The tokenizer() function is used for this. This leaves us with a tokenized list as our ‘X’ and the classes as our ‘Y’. This can now be used as inputs for the neural network.

4.2.3. Train the Model

- This is where the pre-processed data is used to effectively train the model.
- Used keras sequential model with 6 different layers.
- The LSTM layer is what helps the model remember context. This allows for accurate predictions.

4.2.4. Evaluate the model

- Evaluating the models using some metrics. This can be a classification report or f1-score or any metric of the appropriate choice.
- This gives us a good idea of how the model is performing and lets us know what improvements are required.

4.2.5. Make Predictions

- This is when the trained model is used to make predictions.
- The ‘test set’ as its often called, is passed as input to the model and it returns a list of predictions for each row of the test set.
- This gives us a good estimate of how the model will perform.

4.2.6. Server side processing

- The user input from the web UI or the API has to be pre-processed before predicting the class. This is done at the server using tfjs-node.
- The predicted output from the model is turned into JSON and sent as response.

4.3. Non – Functional Requirements

4.3.1. Scaling

Building models for more NLP tasks, and set up APIs for these tasks.

4.3.2. Performance

Increase accuracy of predictions. Increase capacity to handle a large number of API calls at the same time.

4.3.3. Availability

Ensure that the web-app is always functional and available to use for the users. The server handling the API requests must always be up. Any sort of downtime will cause inconvenience to users.

4.3.4. Maintenance

Updates and maintenance of the web-app must be easy. Any updates in the technologies used should be easy to integrate with the previous versions.

5. SYSTEM DESIGN

5.1. Flow Diagram

5.1.1. ML view

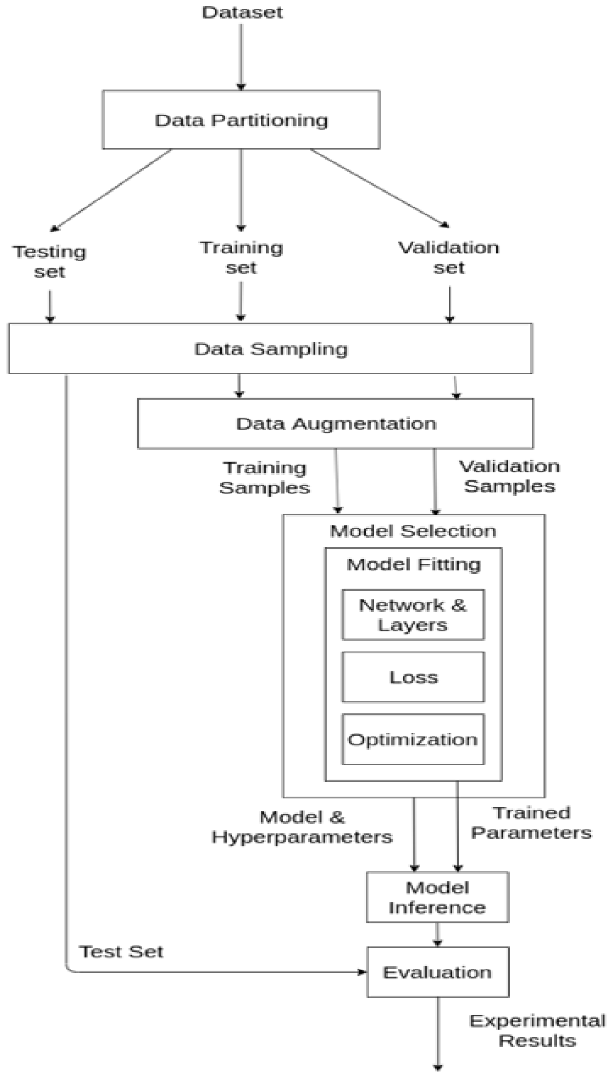


Figure 5.1.1.a – Flow diagram, ML view

The flow diagram for the ML view deals only with the ML processes of the project. It shows how data flows between the different modules of the ML model building process.

5.1.2. Web-application view

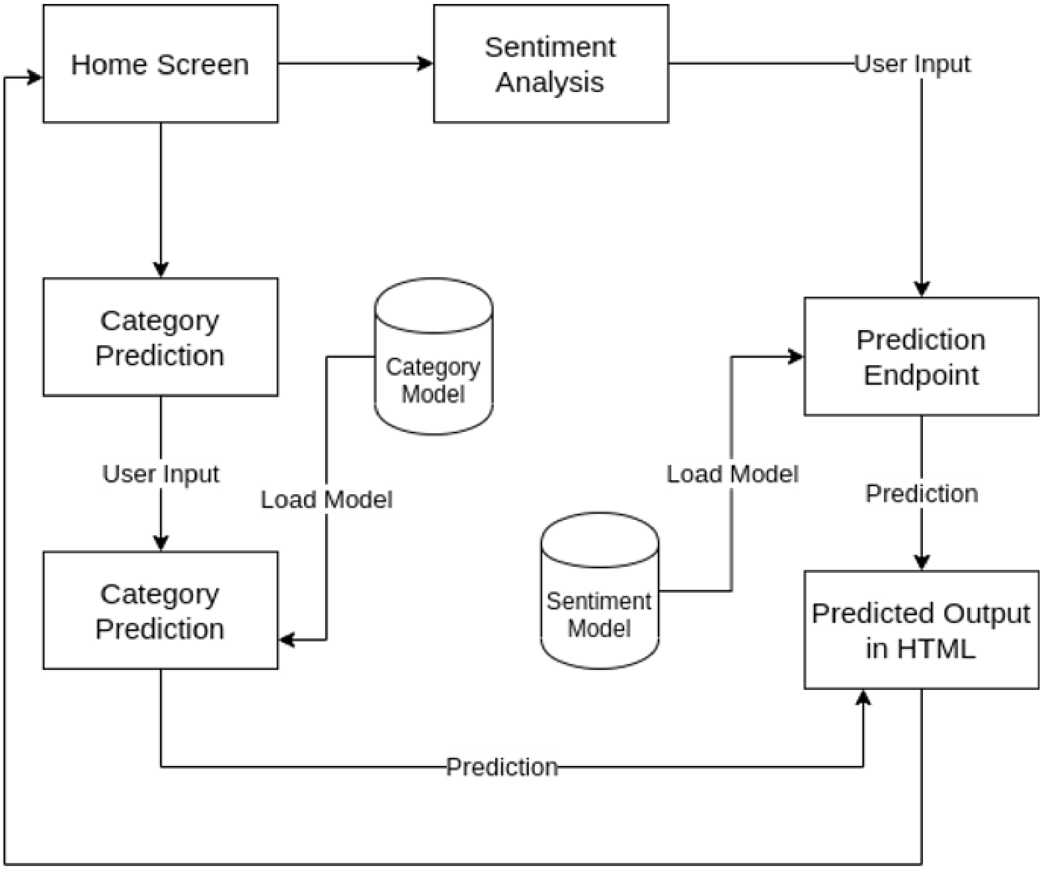


Figure 5.1.2.a – Flow diagram, Web-application view

The flow diagram for the web-application view deals with how the different processes communicate in the web-application from server side to client side, and also how the server communicates with the models saved in memory.

5.2. Detailed Methodology

5.2.1. Text Pre-processing

The datasets contain text data that need to be converted into tensors in order to provide them as input for the CNN. The neural network does not take in raw text data as input, so we need to convert this text data into a sequence of numbers (or a tensor) which is the appropriate input type. But first, a dictionary of words from the dataset is create. Its significance is explained later. This dictionary is built using the **gensim.corpora.Dictionary** class. This contains a dictionary of words with their integer values as keys. This dictionary is of paramount importance as it plays a very important role in increasing the accuracy of the model. The dictionary is then used to convert the dataset into numerical inputs by using the **token2id** method available from **gensim.corpora**. This method converts the dataset into key-value pairs by giving each word in the dataset the integer value from the same word in the dictionary. This process is called **tokenization** and the numbers are called **tokens**. These tokens are then converted into a 1D list which consists only of tokens. The code and outputs from these processes are available in the Implementation section.

The user input for predictions are also pre-processed in the same way. The same dictionary is used. If the dictionary isn’t used the user inputs would be saved as [0,1,2,3,4] and this would lead to very inaccurate predictions. When the dictionary is used, the code checks if the words from the user input exists in the dictionary, and then the appropriate token is assigned. This makes the model give us better predictions.

5.2.2. Building the model

a) **Convolutional Neural Networks**

Convolutional Neural Networks, or CNNs, are specialized neural networks that take in input as a 2D matrix. In the case of this project, the input, that is a list of tokens is converted into a series of convoluted matrices. Each row of the matrix corresponds to a token which may be a word or a character. Each row is a vector that represents a word. In NLP the filters in CNNs slide over sentence matrices, a few words at a time.

b) The layers of the model

All three models used in this project are similar but with slight differences. What all 3 models have in common is that they all have the same layers but the parameters differ to make sure the datasets were used to train the model with maximum efficiency.

i. Embedding

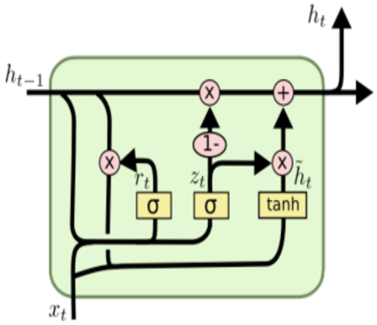
The embedding layer can only be used as the first layer of the model. The embedding layer takes 2 main parameters. The input dimension and the output dimension. The input dimension is an integer that specifies the size of the vocabulary. The output dimension is an integer that specifies what shape the output of the embedding layer must be. This means that we can use the embedding layer to convert higher dimensional data into lower dimensional vector space.

In this project it is implemented like this:

```
model.add(Embedding(len(dictionary), embed_size)
```

ii. LSTM

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word of the sentence “the cars are on the *road*”, we don’t need any further context – it’s pretty obvious the next word is going to be “road”. In such cases where the gap between relevant information and the place that it’s needed is small, RNNs can learn to use the past information. But, there are also cases where we need more context. Consider trying to predict the last word in the sentence – “I grew up in Spain. I speak fluent *Spanish*”. Recent information suggests that the next word is probably the name of the language. But if we want to find out which language, we need the context of “Spain” from further back. Hence, it is entirely possible for the gap between information and the place where it is needed to become very large. As that gap grows, RNNs become unable to connect the information. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 5.2.2.a - LSTM

c) Model architecture

The following diagrams show the architecture of the models. They consist of all the layers in the model, showing input and output shapes for each layer. The architecture diagrams for all 3 models; sentiment analysis, multi-class classification, spam detection are shown below.

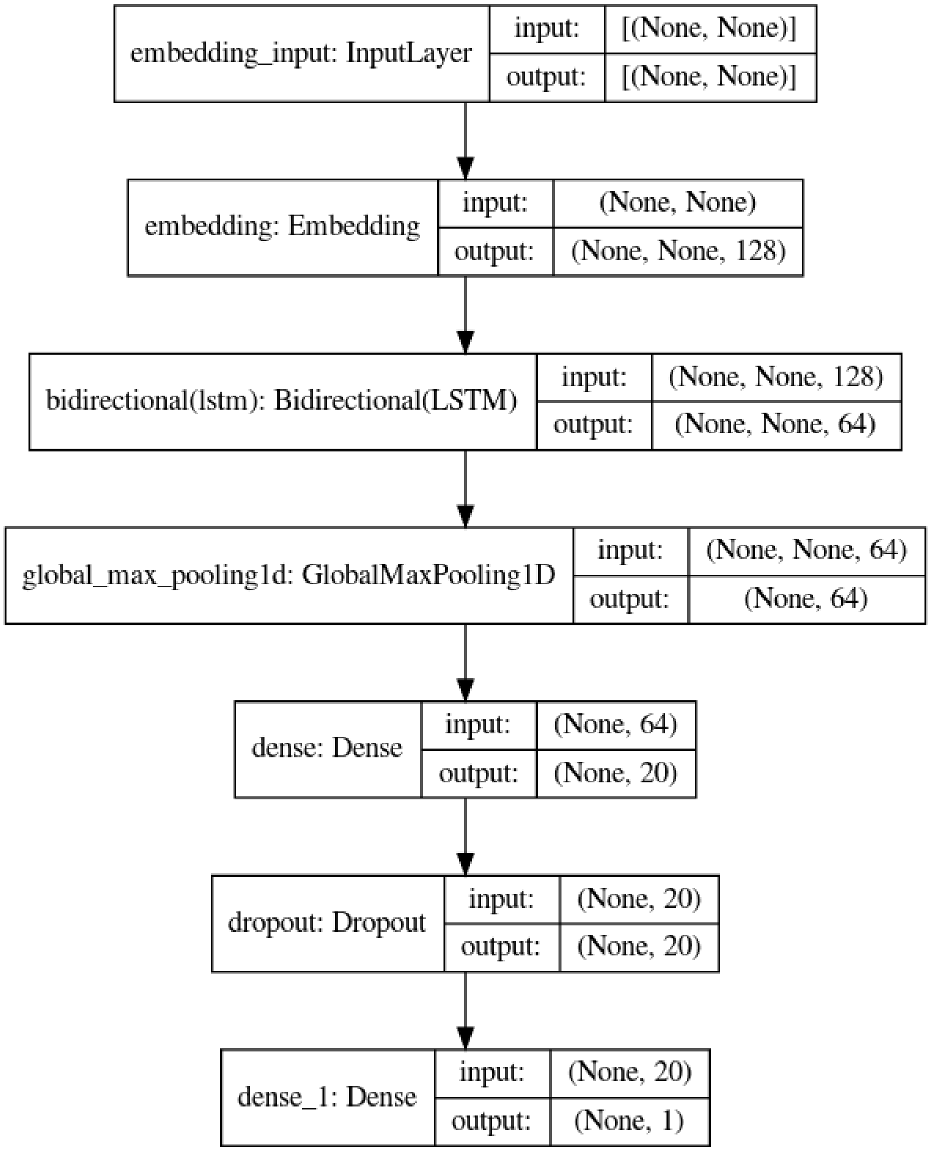


Figure 5.2.2.b – Architecture of Sentiment Analysis model

The above diagram shows the layers and input and output shapes of each layer for the sentiment analysis model.

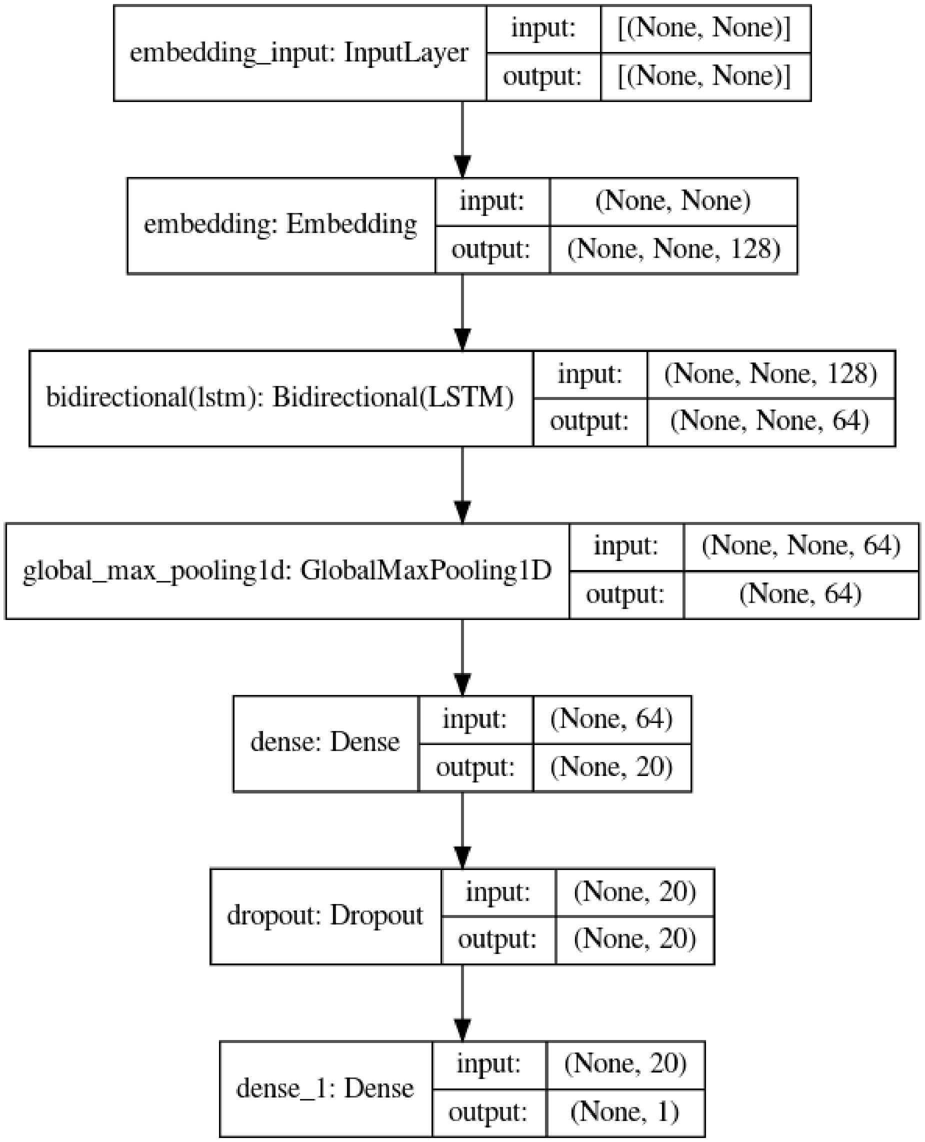


Figure 5.2.2.c – Architecture of Multi-class Classification model

The above diagram shows the layers and input and output shapes of each layer for the multi-class classification model.

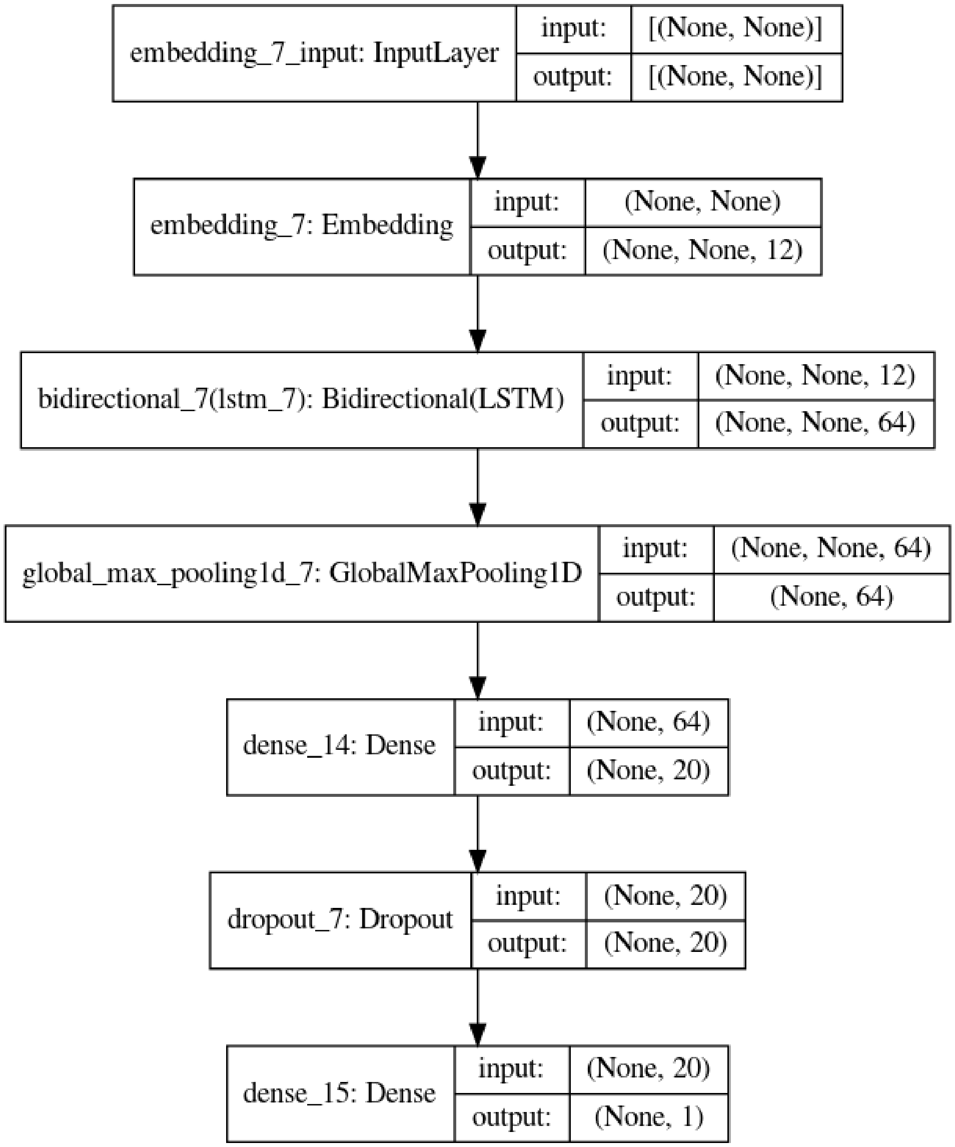


Figure 5.2.2.d – Architecture of Spam Detection model

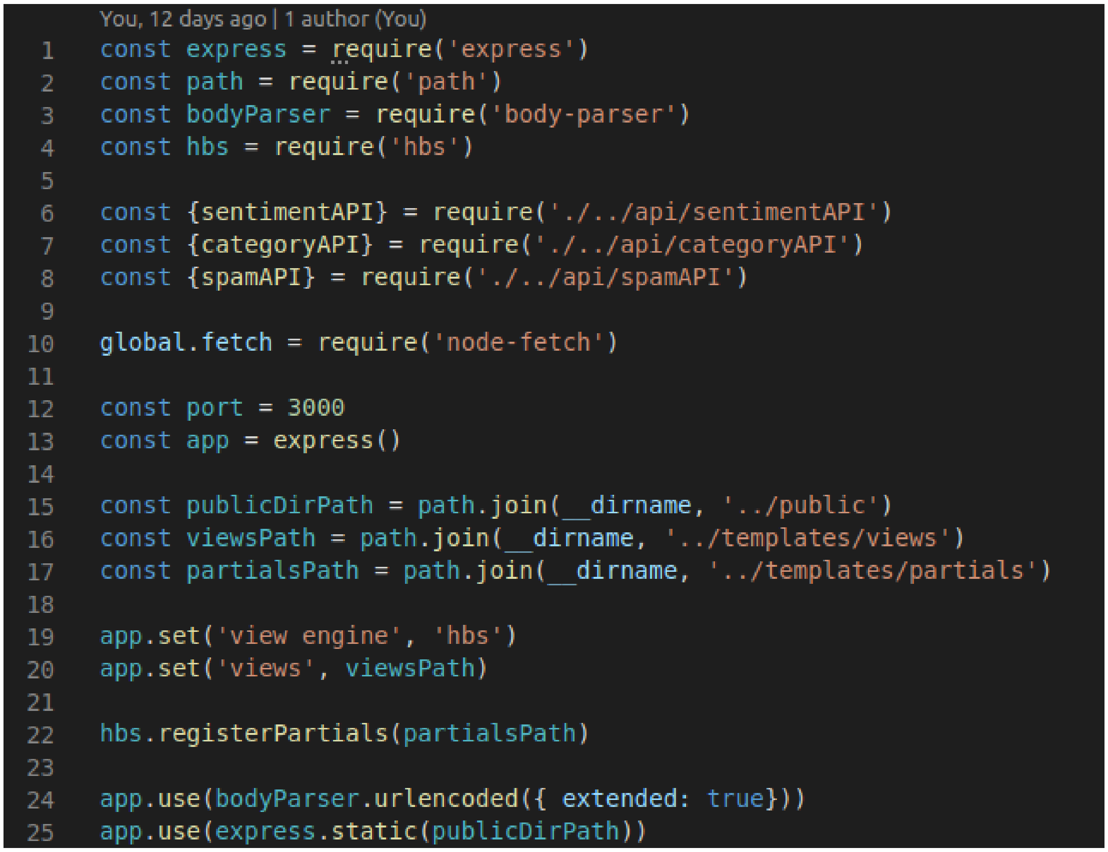
The above diagram shows the layers and input and output shapes of each layer for the spam detection model.

6. IMPLEMENTATION

6.1. Sample Source Code and Description

6.1.1 Setting up routes for the web application

First, I set up the prerequisites for the server to run. That includes, installing express.js and importing it. Also importing hbs, body-parser, path. Here I also set up the view engine as hbs, this tells the server to use handlebars as the view engine and it serves up hbs files at the client side from the folder specified.



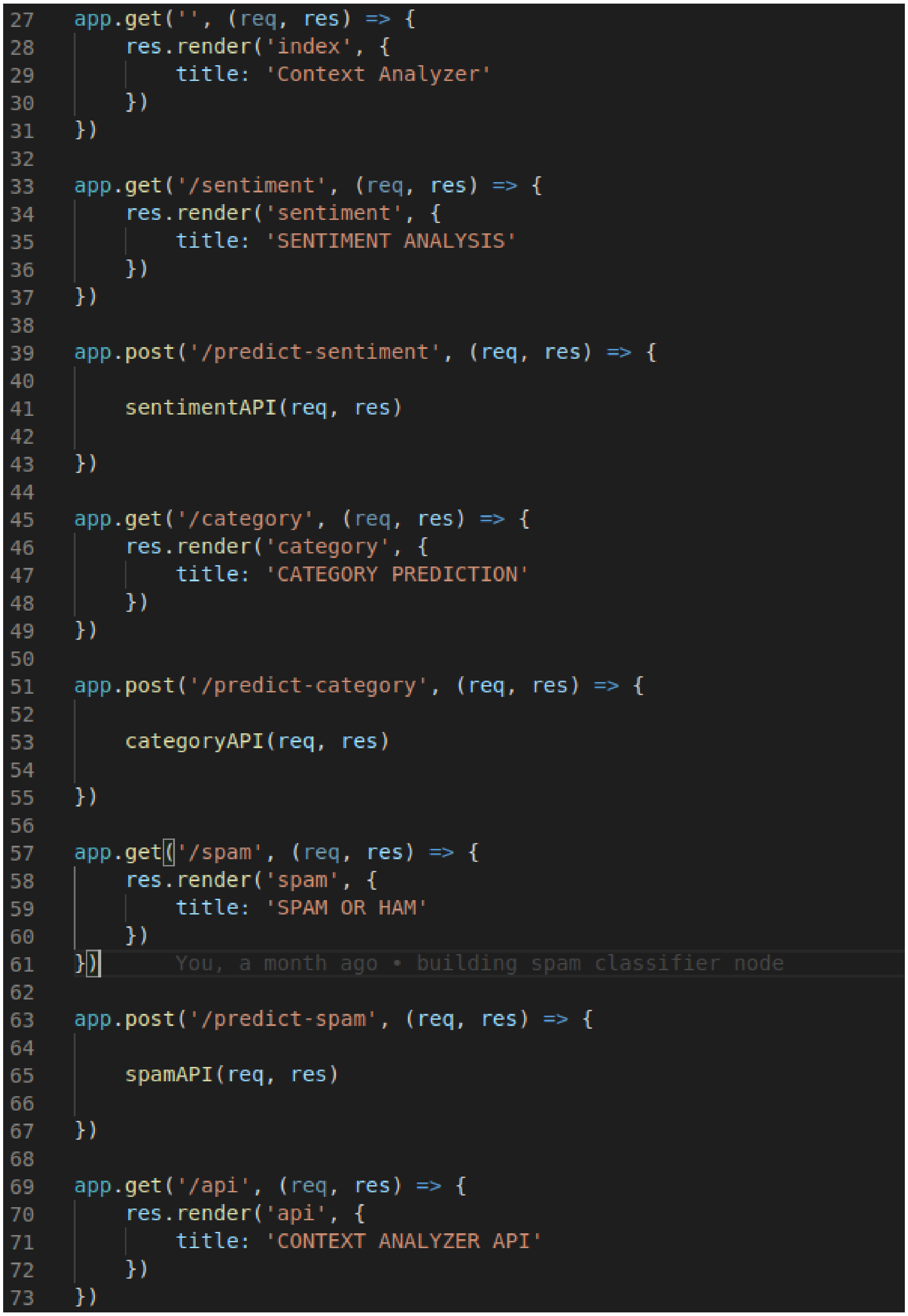
You, 12 days ago | 1 author (You)

```
const express = require('express')
const path = require('path')
const bodyParser = require('body-parser')
const hbs = require('hbs')

const {sentimentAPI} = require('./../api/sentimentAPI')
const {categoryAPI} = require('./../api/categoryAPI')
const {spamAPI} = require('./../api/spamAPI')

global.fetch = require('node-fetch')

const port = 3000
const app = express()

const publicDirPath = path.join(__dirname, '../public')
const viewsPath = path.join(__dirname, '../templates/views')
const partialsPath = path.join(__dirname, '../templates/partials')

app.set('view engine', 'hbs')
app.set('views', viewsPath)

hbs.registerPartials(partialsPath)

app.use(bodyParser.urlencoded({ extended: true}))
app.use(express.static(publicDirPath))
```

Figure 6.1.1.a – server.js set-up

After this, the routes can be written. Most of the GET requests render a hbs file. For example, a GET request made to ‘/sentiment-analysis’ would render a hbs file for the sentiment analysis task.

The routes look like this.



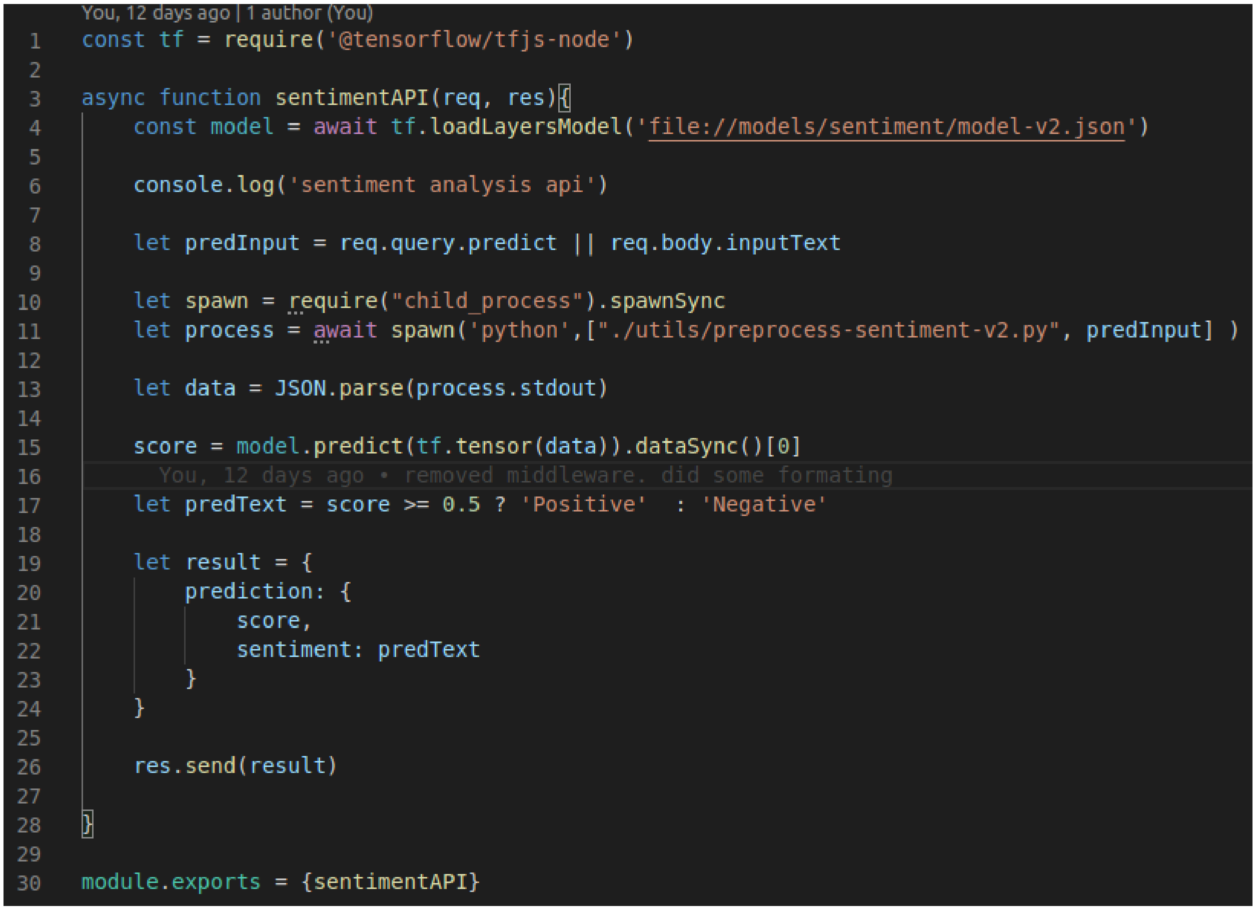
```
app.get('', (req, res) => {
    res.render('index', {
        title: 'Context Analyzer'
    })
})

app.get('/sentiment', (req, res) => {
    res.render('sentiment', {
        title: 'SENTIMENT ANALYSIS'
    })
})

app.post('/predict-sentiment', (req, res) => {

    sentimentAPI(req, res)

})

app.get('/category', (req, res) => {
    res.render('category', {
        title: 'CATEGORY PREDICTION'
    })
})

app.post('/predict-category', (req, res) => {

    categoryAPI(req, res)

})

app.get('/spam', (req, res) => {
    res.render('spam', {
        title: 'SPAM OR HAM'
    })
})        You, a month ago • building spam classifier node

app.post('/predict-spam', (req, res) => {

    spamAPI(req, res)

})

app.get('/api', (req, res) => {
    res.render('api', {
        title: 'CONTEXT ANALYZER API'
    })
})
```

Figure 6.1.1.b – server.js routes

6.1.2. The API logic

When the API calls happen, a function runs to load the model and make predictions and send those predictions as response.

The code for sentiment analysis API.



```
You, 12 days ago | 1 author (You)
const tf = require('@tensorflow/tfjs-node')

async function sentimentAPI(req, res){
    const model = await tf.loadLayersModel('file://models/sentiment/model-v2.json')

    console.log('sentiment analysis api')

    let predInput = req.query.predict || req.body.inputText

    let spawn = require("child_process").spawnSync
    let process = await spawn('python',["./utils/preprocess-sentiment-v2.py", predInput] )

    let data = JSON.parse(process.stdout)

    score = model.predict(tf.tensor(data)).dataSync()[0]
      You, 12 days ago • removed middleware. did some formating
    let predText = score >= 0.5 ? 'Positive'  : 'Negative'

    let result = {
        prediction: {
            score,
            sentiment: predText
        }
    }

    res.send(result)

}

module.exports = {sentimentAPI}
```

Figure 6.1.1.c – sentimentAPI.js

The other APIs are also setup in a similar way. The asynchronous function setup makes sure that the model is loaded properly. The input is preprocessed and the model is used to predict the processed input. The predictions are sent as a JSON response.

6.1.3. Pre-processing and Model Generation

6.1.3.i. Pre-processing

All of the above handles the server side of the application. But the most important part of the application is the models. Without them none of this can be used in any meaningful way. So, here is how the models are generated.

The pre-processing is pretty similar for all of the models. Only the size of the input differs as each dataset is different. The code for pre-processing looks something like this.

Code from sentiment.ipynb

```
MAX_SEQUENCE_LEN = 130
UNK = 'UNK'
PAD = 'PAD'

def text_to_id_list(text, dictionary):
    return [dictionary.token2id.get(tok, dictionary.token2id.get(UNK))
            for tok in text_to_tokens(text)]

def texts_to_input(texts, dictionary):
    return sequence.pad_sequences(
        list(map(lambda x: text_to_id_list(x, dictionary), texts)),
            maxlen=MAX_SEQUENCE_LEN,
            padding='post', truncating='post',
            value=dictionary.token2id.get(PAD))

def text_to_tokens(text):
    return [tok.text.lower() for tok in nlp.tokenizer(text)
            if not (tok.is_punct or tok.is_quote)]

def build_dictionary(texts):
    d = Dictionary(text_to_tokens(t)for t in texts)
    d.filter_extremes(no_below=3, no_above=1)
    d.add_documents([[UNK, PAD]])
    d.compactify()
    return d
```

The above code is used to create the dictionary that was mentioned in the methodology section. This is also used to convert the raw text input into a list of tokens that will be used as input to the neural network.

```
x_train = texts_to_input(df.review, dictionary)
```

Converts df.reviews into list of tokens and stores it in `x_train`.

6.1.3.ii. Model Generation

After the dataset is processed. x_train, y_train, x_test, y_test are created for training and testing purposes. y_train and y_test are created by converting the labels ‘positive’, ‘negative’ into 1 and 0.

```
df['sentiment'] = df['sentiment'].map({'pos': 1, 'neg': 0})
```

Next, we move on to building the model. The model is built carefully by choosing the right parameters and layers. This is done to ensure maximum efficiency and minimum loss. The model is trained over 3 epochs.

```
model = Sequential()
model.add(Embedding(len(dictionary), embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

batch_size = 100
epochs = 3
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.2)
```

The output looks like this.

```
Train on 60000 samples, validate on 15000 samples
Epoch 1/3
60000/60000 [==============================] - 271s 5ms/sample - loss:
0.3670 - acc: 0.8293 - val_loss: 0.2173 - val_acc: 0.9251
Epoch 2/3
60000/60000 [==============================] - 271s 5ms/sample - loss:
0.1643 - acc: 0.9402 - val_loss: 0.1007 - val_acc: 0.9669
Epoch 3/3
60000/60000 [==============================] - 268s 4ms/sample - loss:
0.0728 - acc: 0.9759 - val_loss: 0.0648 - val_acc: 0.9782
```

The other 2 models are quite similar, only the embed_size and the dictionary size changes.

6.1.4. Model Evaluation

Once the model is trained, it is saved using

```
model.save('sentiment_model-v2.h5')
```

The test set is used to make predictions and evaluate the model. It looks something like this.

```
prediction = model.predict(x_test)

y_pred = (prediction > 0.5)

from sklearn.metrics import f1_score

print('F1-score: {0}'.format(f1_score(y_pred, y_test)))
F1-score: 0.9861099959855479
```

This covers the entire project in brief. However there is still one tiny, yet important part to be done. The tensorflow.js code can not import .h5 files. It can only import .json models. So the model.h5 files have to be converted into model.json files. This is done easily in just one line.

```
$ tensorflowjs_converter --input_format=keras /tmp/model.h5 /tmp/tfjs_model
```

6.2. Screenshots

1. Home

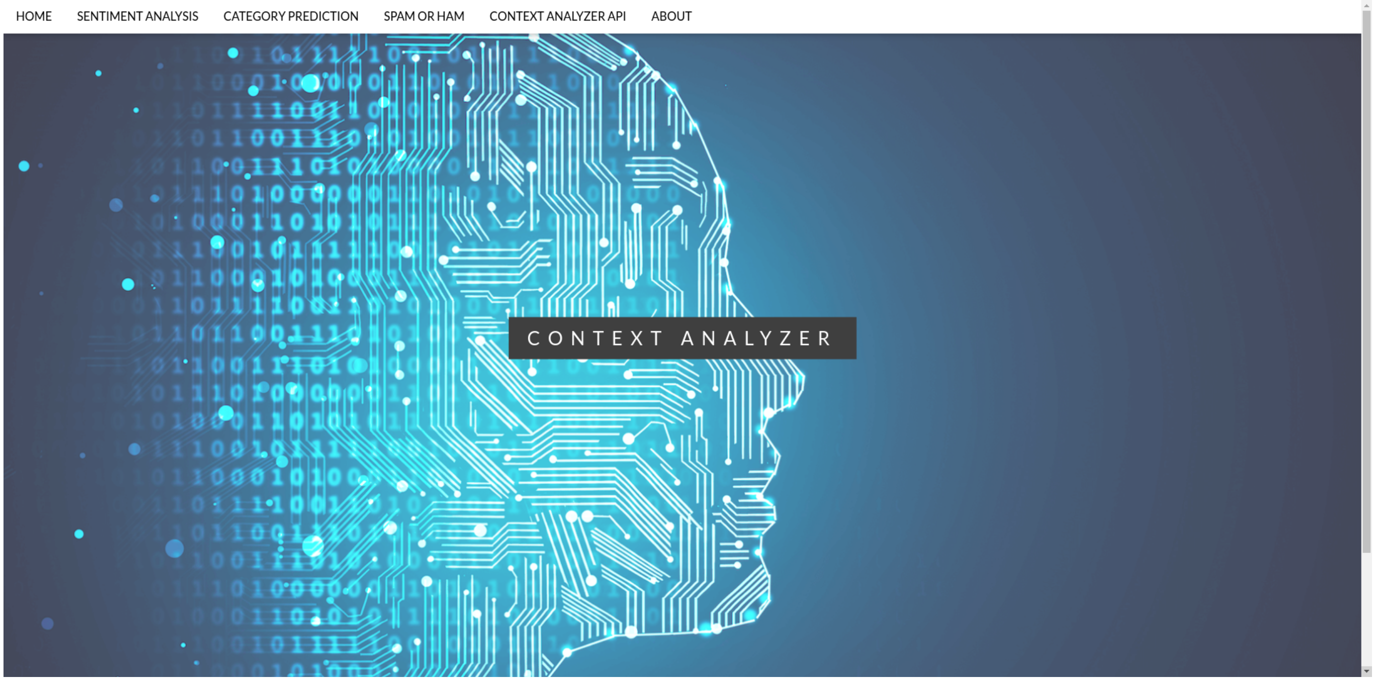
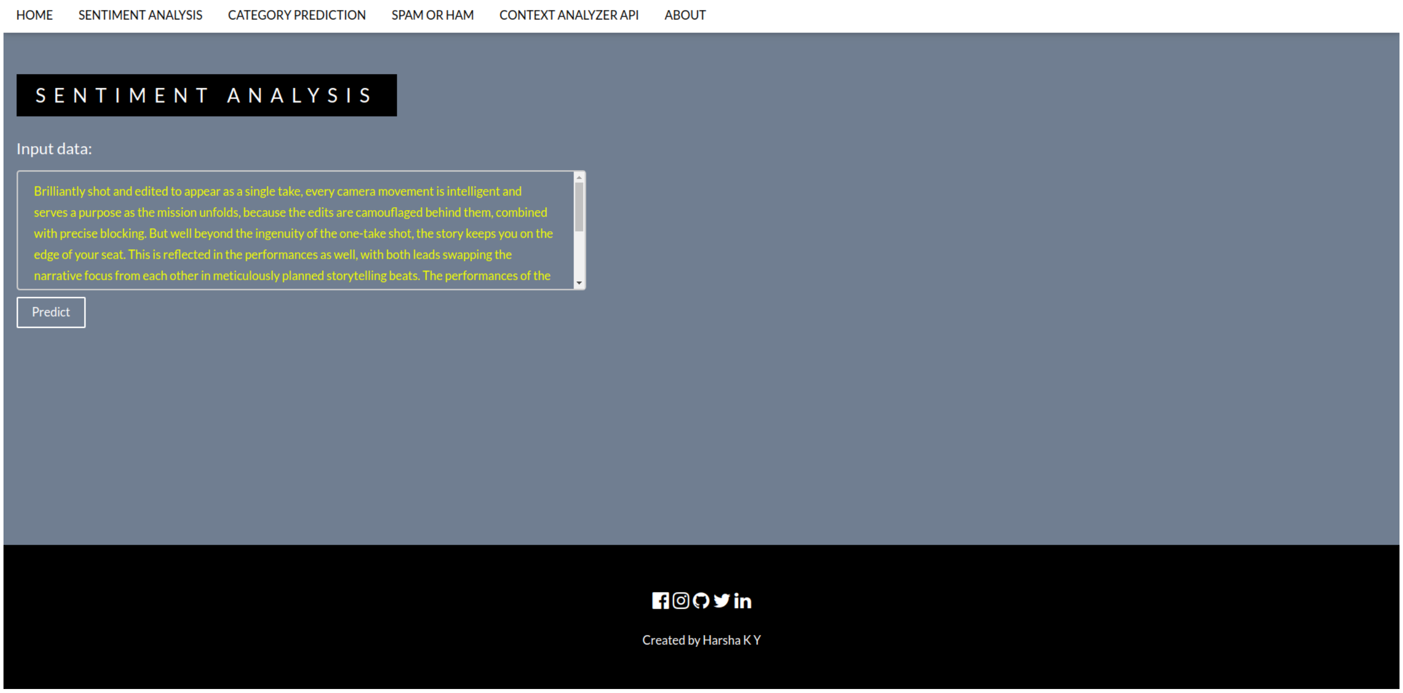


Figure 6.2.a – Home Page

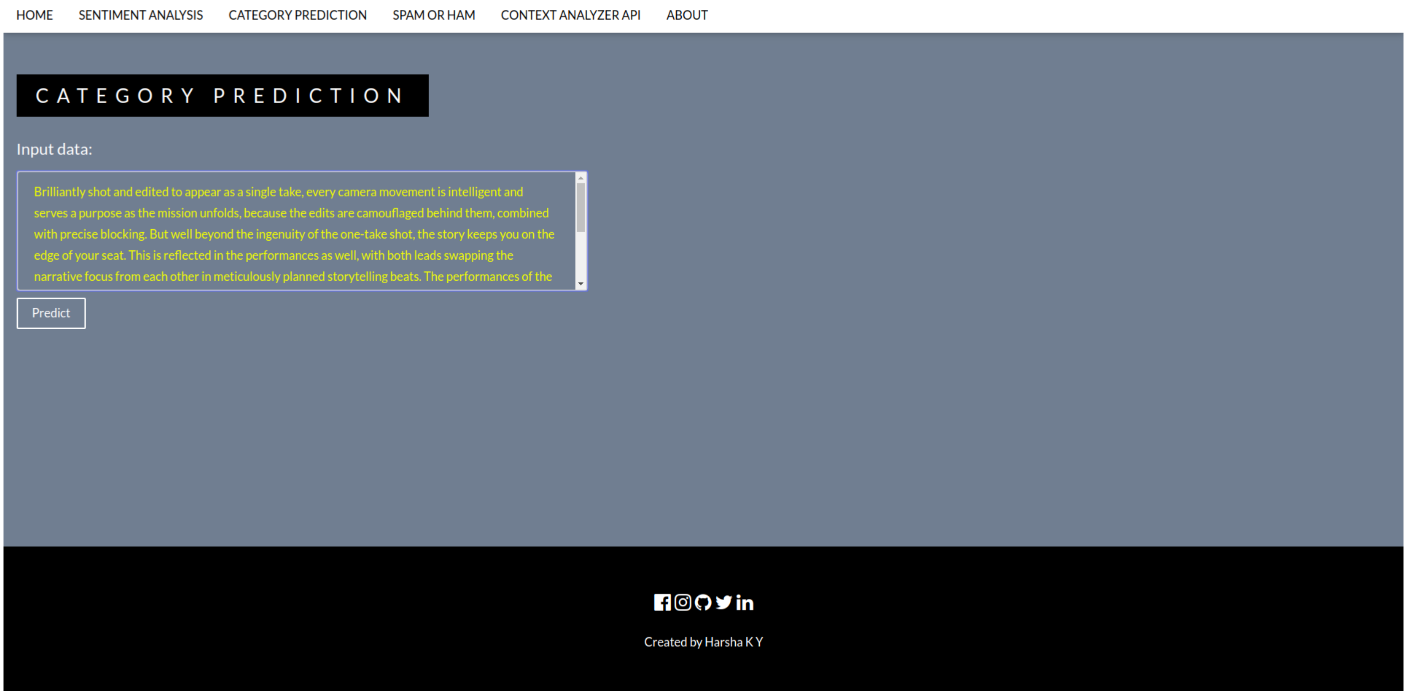
2. Sentiment Analysis



Brilliantly shot and edited to appear as a single take, every camera movement is intelligent and serves a purpose as the mission unfolds, because the edits are camouflaged behind them, combined with precise blocking. But well beyond the ingenuity of the one-take shot, the story keeps you on the edge of your seat. This is reflected in the performances as well, with both leads swapping the narrative focus from each other in meticulously planned storytelling beats. The performances of the

Figure 6.2.b – Sentiment Analysis

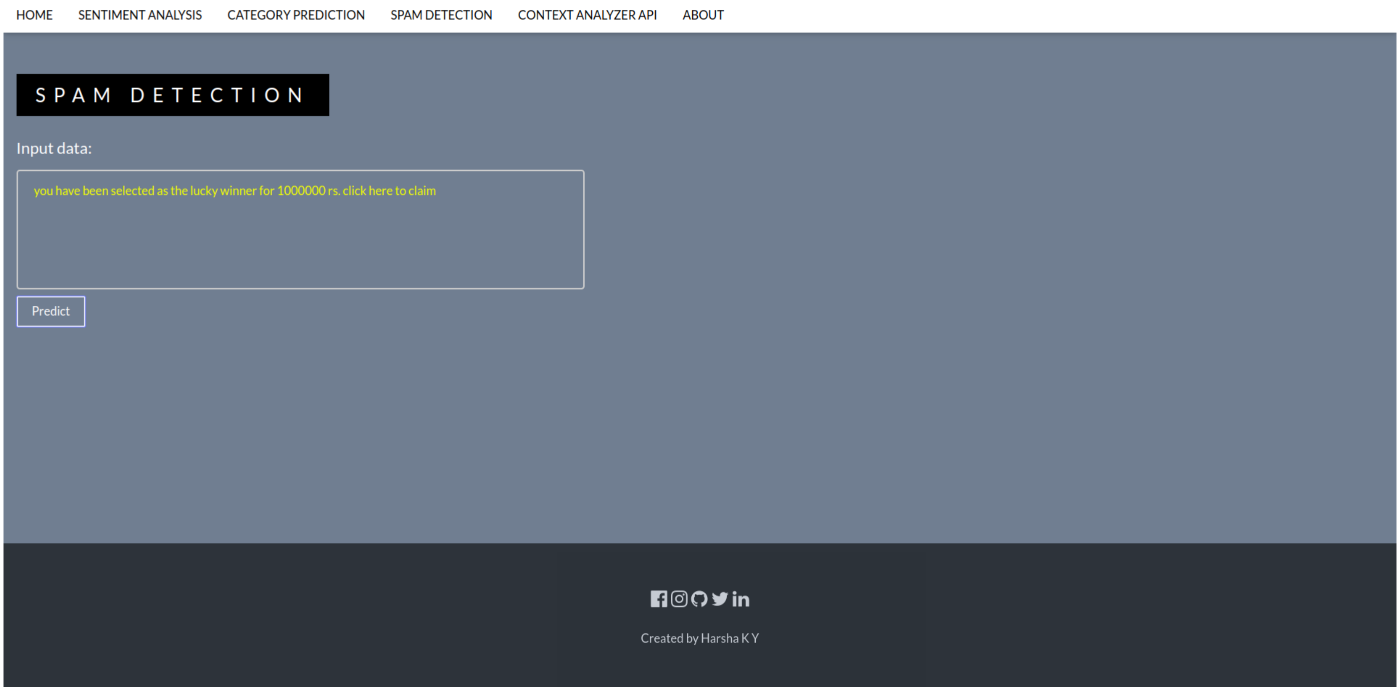
3. Category Prediction



Brilliantly shot and edited to appear as a single take, every camera movement is intelligent and serves a purpose as the mission unfolds, because the edits are camouflaged behind them, combined with precise blocking. But well beyond the ingenuity of the one-take shot, the story keeps you on the edge of your seat. This is reflected in the performances as well, with both leads swapping the narrative focus from each other in meticulously planned storytelling beats. The performances of the

Figure 6.2.c – Category Prediction

4. Spam Detection

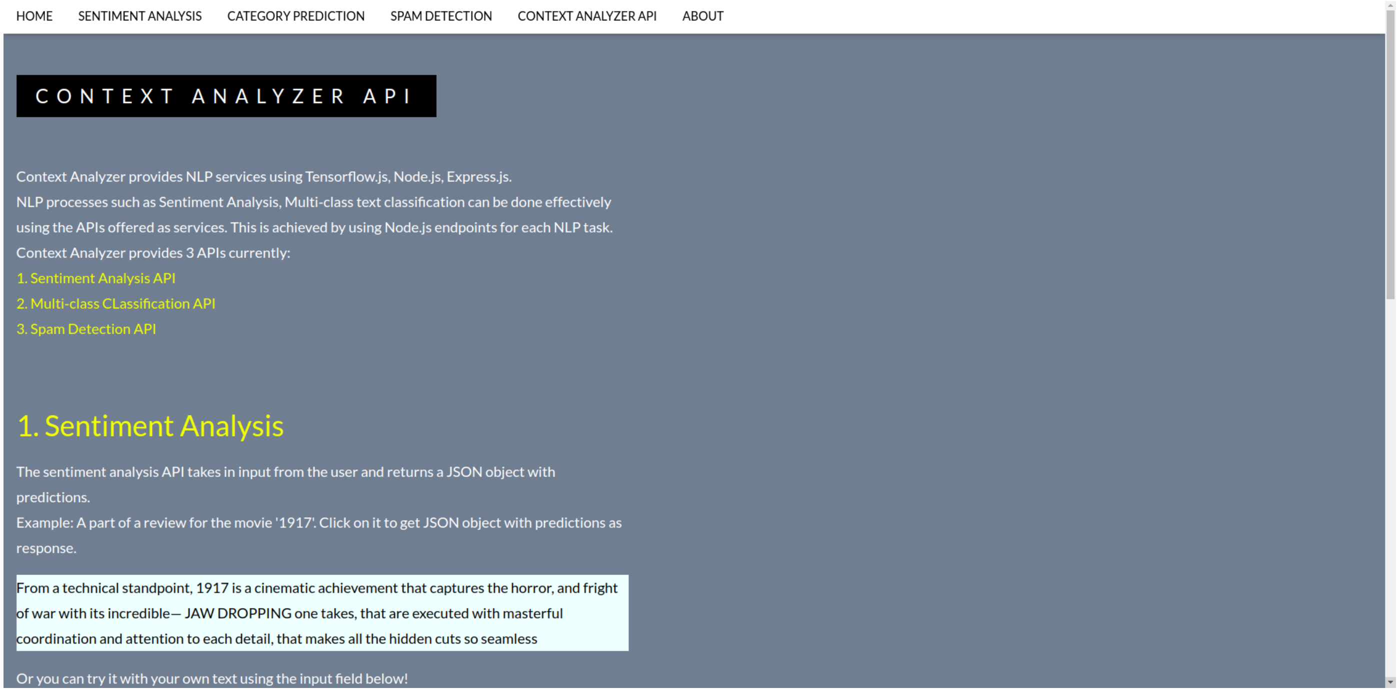


Input data:

you have been selected as the lucky winner for 1000000 rs. click here to claim

Figure 6.2.d – Spam Detection

5. Context Analyzer API (Documentation)



CONTEXT ANALYZER API

Context Analyzer provides NLP services using Tensorflow.js, Node.js, Express.js.
NLP processes such as Sentiment Analysis, Multi-class text classification can be done effectively using the APIs offered as services. This is achieved by using Node.js endpoints for each NLP task.
Context Analyzer provides 3 APIs currently:
1. Sentiment Analysis API
2. Multi-class CLassification API
3. Spam Detection API

1. Sentiment Analysis

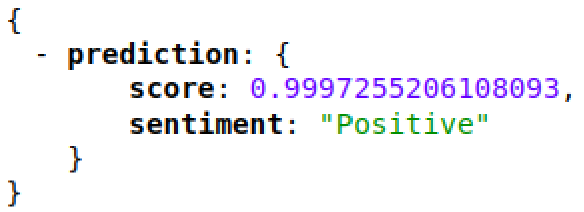
The sentiment analysis API takes in input from the user and returns a JSON object with predictions.
Example: A part of a review for the movie '1917'. Click on it to get JSON object with predictions as response.

From a technical standpoint, 1917 is a cinematic achievement that captures the horror, and fright of war with its incredible— JAW DROPPING one takes, that are executed with masterful coordination and attention to each detail, that makes all the hidden cuts so seamless

Or you can try it with your own text using the input field below!

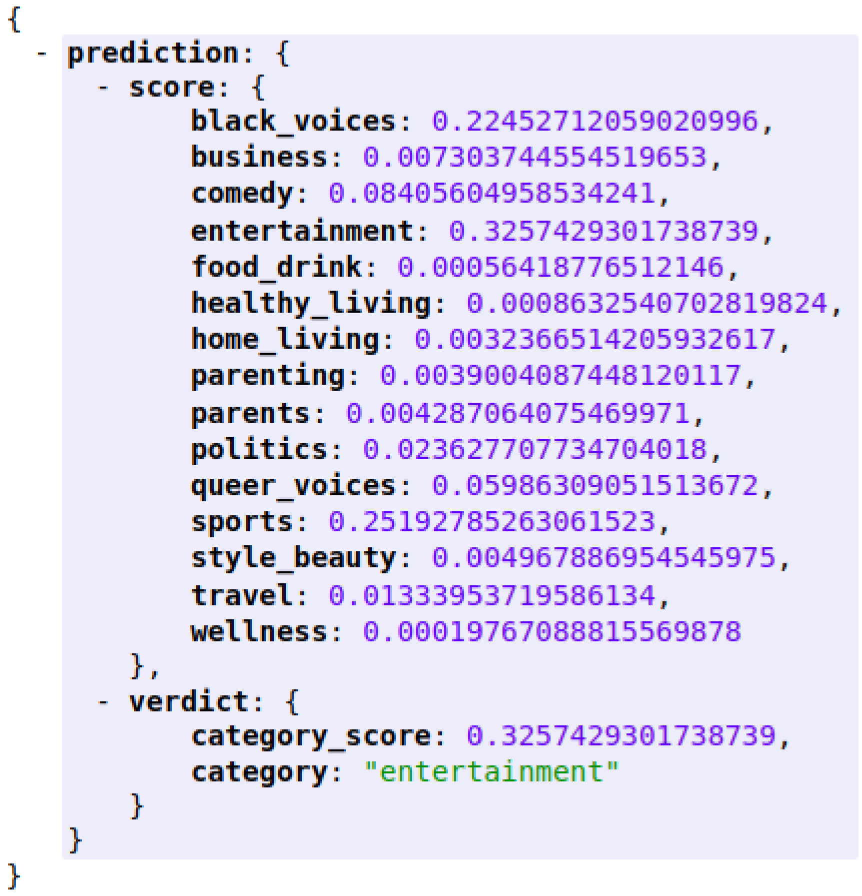
Figure 6.2.e – API Documentation

6. JSON responses



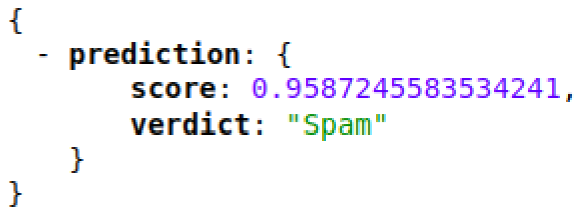
```
{
  - prediction: {
        score: 0.9997255206108093,
        sentiment: "Positive"
    }
}
```

Figure 6.2.f – Sentiment Response



```
{
  - prediction: {
      - score: {
            black_voices: 0.22452712059020996,
            business: 0.007303744554519653,
            comedy: 0.08405604958534241,
            entertainment: 0.3257429301738739,
            food_drink: 0.00056418776512146,
            healthy_living: 0.0008632540702819824,
            home_living: 0.0032366514205932617,
            parenting: 0.0039004087448120117,
            parents: 0.004287064075469971,
            politics: 0.023627707734704018,
            queer_voices: 0.05986309051513672,
            sports: 0.25192785263061523,
            style_beauty: 0.004967886954545975,
            travel: 0.01333953719586134,
            wellness: 0.00019767088815569878
        },
      - verdict: {
            category_score: 0.3257429301738739,
            category: "entertainment"
        }
    }
}
```

Figure 6.2.g – Category Response



```
{
  - prediction: {
        score: 0.9587245583534241,
        verdict: "Spam"
    }
}
```

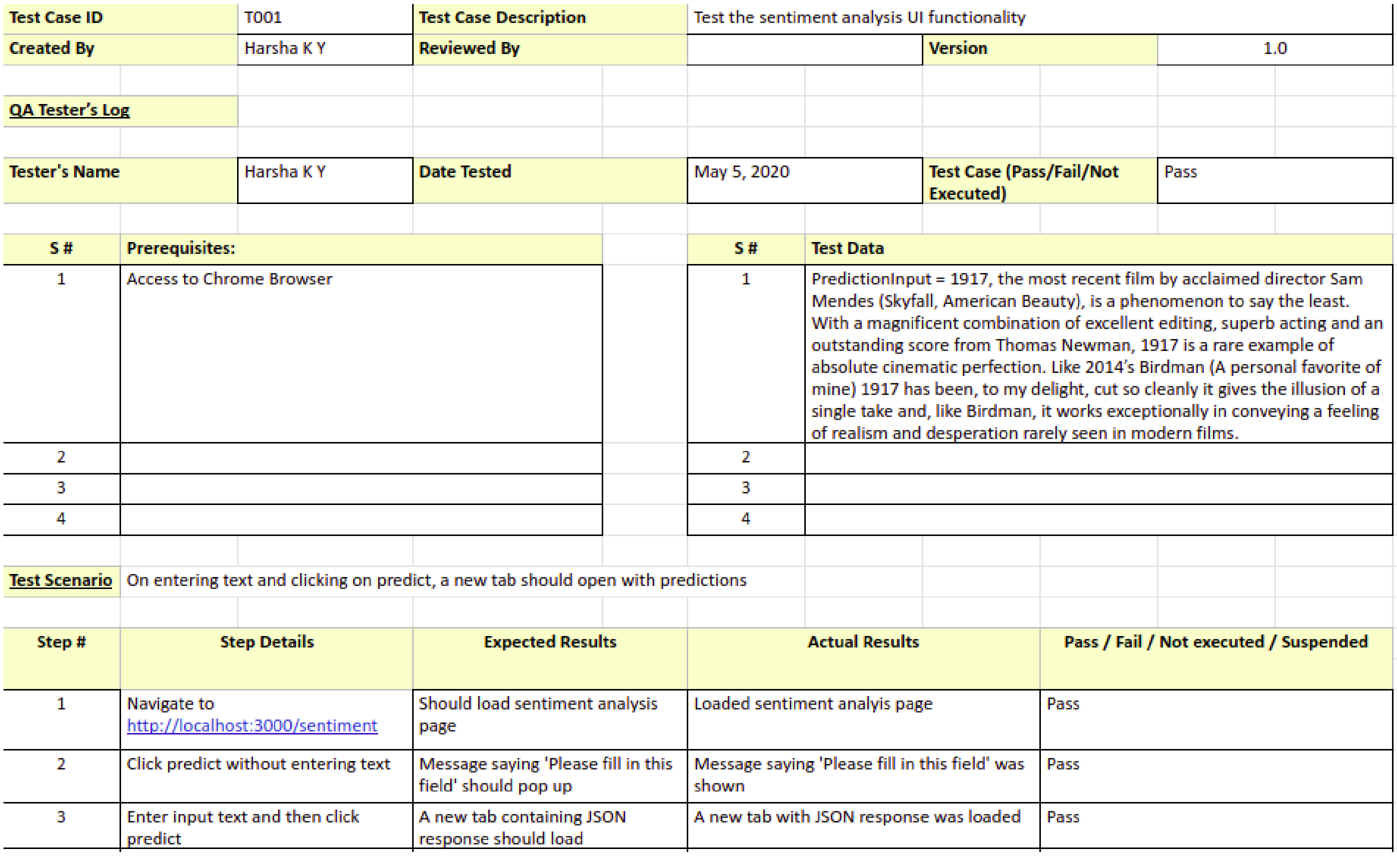
Figure 6.2.h – Spam Response

7. SOFTWARE TESTING

Software testing is the process of checking whether the actual results of using the product meet the expected results. It can also be used to evaluate the functionality of the the application. This lets us find faults and identify potential fixes if any mistakes are found.

7.1 Test Cases

Table 7.1.a – Test Case T001



Test Case ID	T001	**Test Case Description**	Test the sentiment analysis UI functionality		
Created By	Harsha K Y	**Reviewed By**		**Version**	1.0

QA Tester's Log

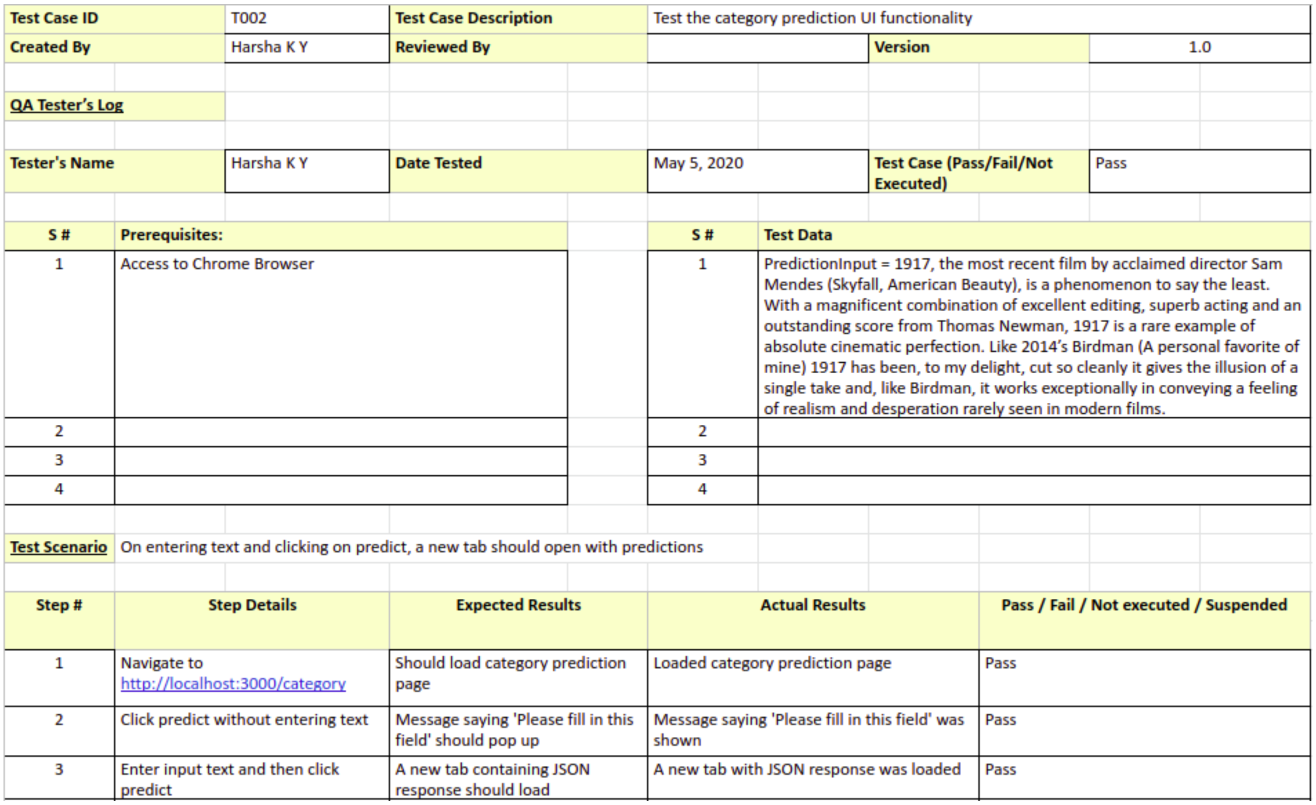
Tester's Name	Harsha K Y	**Date Tested**	May 5, 2020	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**	**S #**	**Test Data**
1	Access to Chrome Browser	1	PredictionInput = 1917, the most recent film by acclaimed director Sam Mendes (Skyfall, American Beauty), is a phenomenon to say the least. With a magnificent combination of excellent editing, superb acting and an outstanding score from Thomas Newman, 1917 is a rare example of absolute cinematic perfection. Like 2014's Birdman (A personal favorite of mine) 1917 has been, to my delight, cut so cleanly it gives the illusion of a single take and, like Birdman, it works exceptionally in conveying a feeling of realism and desperation rarely seen in modern films.
2		2	
3		3	
4		4	

Test Scenario On entering text and clicking on predict, a new tab should open with predictions

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to http://localhost:3000/sentiment	Should load sentiment analysis page	Loaded sentiment analyis page	Pass
2	Click predict without entering text	Message saying 'Please fill in this field' should pop up	Message saying 'Please fill in this field' was shown	Pass
3	Enter input text and then click predict	A new tab containing JSON response should load	A new tab with JSON response was loaded	Pass

Table 7.1.b – Test Case T002



Test Case ID	T002	**Test Case Description**	Test the category prediction UI functionality		
Created By	Harsha K Y	**Reviewed By**		**Version**	1.0

QA Tester's Log

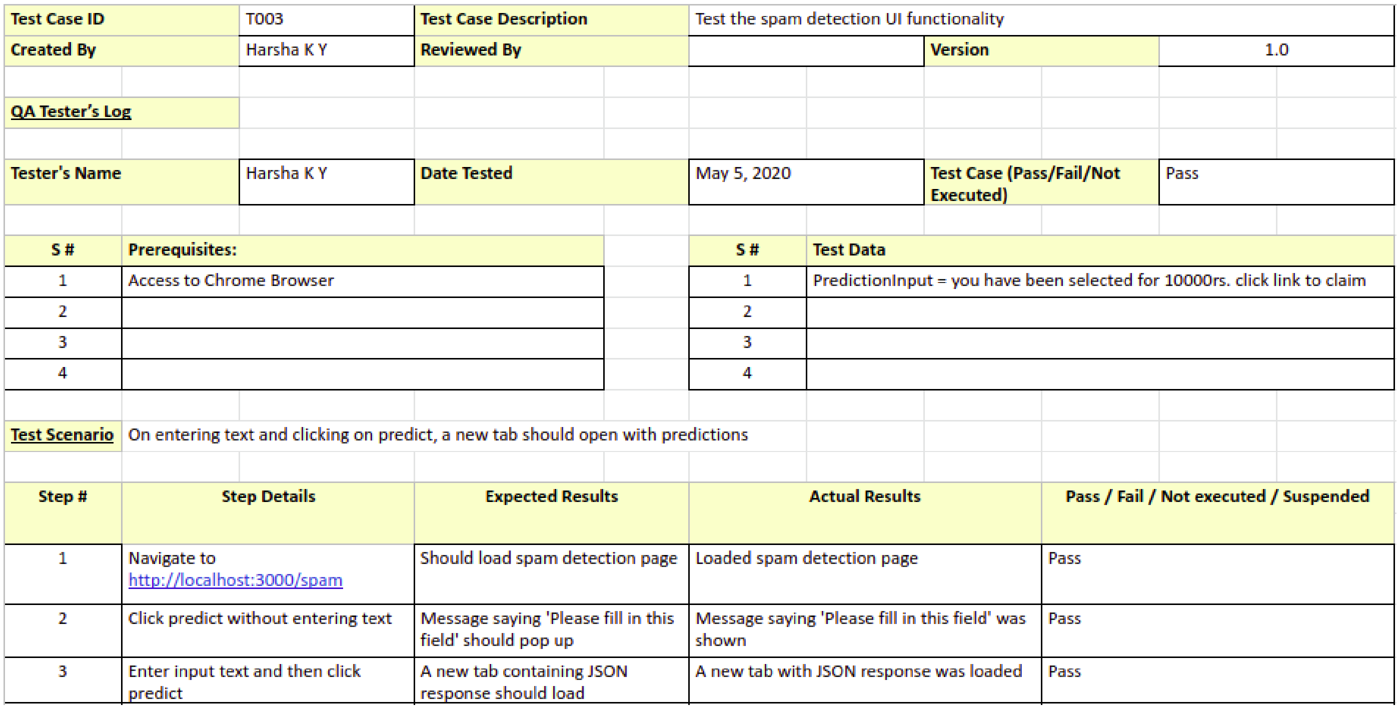
Tester's Name	Harsha K Y	**Date Tested**	May 5, 2020	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**	**S #**	**Test Data**
1	Access to Chrome Browser	1	PredictionInput = 1917, the most recent film by acclaimed director Sam Mendes (Skyfall, American Beauty), is a phenomenon to say the least. With a magnificent combination of excellent editing, superb acting and an outstanding score from Thomas Newman, 1917 is a rare example of absolute cinematic perfection. Like 2014's Birdman (A personal favorite of mine) 1917 has been, to my delight, cut so cleanly it gives the illusion of a single take and, like Birdman, it works exceptionally in conveying a feeling of realism and desperation rarely seen in modern films.
2		2	
3		3	
4		4	

Test Scenario On entering text and clicking on predict, a new tab should open with predictions

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to http://localhost:3000/category	Should load category prediction page	Loaded category prediction page	Pass
2	Click predict without entering text	Message saying 'Please fill in this field' should pop up	Message saying 'Please fill in this field' was shown	Pass
3	Enter input text and then click predict	A new tab containing JSON response should load	A new tab with JSON response was loaded	Pass

Table 7.1.c – Test Case T003



Test Case ID	T003	**Test Case Description**	Test the spam detection UI functionality		
Created By	Harsha K Y	**Reviewed By**		**Version**	1.0

QA Tester's Log

Tester's Name	Harsha K Y	**Date Tested**	May 5, 2020	**Test Case (Pass/Fail/Not Executed)**	Pass

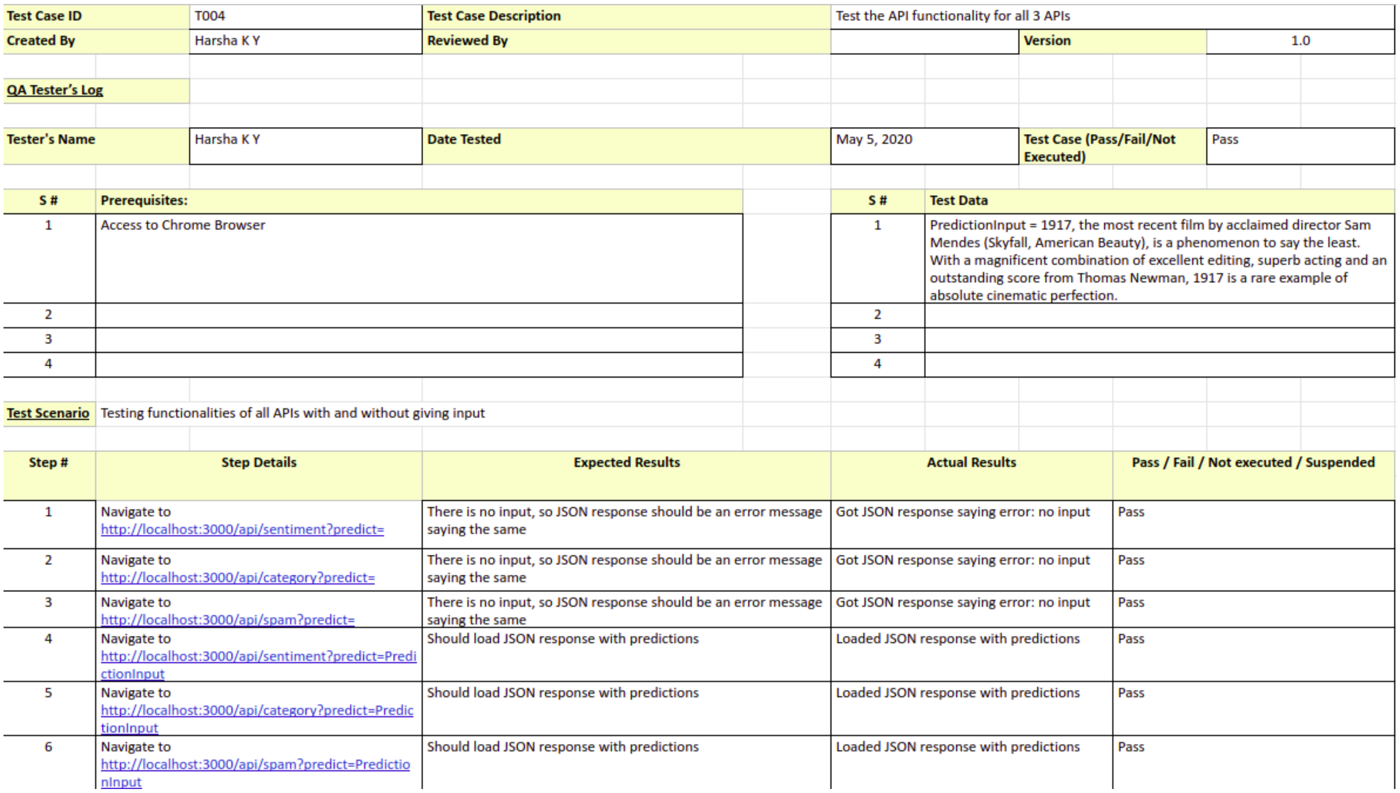
S #	**Prerequisites:**
1	Access to Chrome Browser
2	
3	
4	

S #	**Test Data**
1	PredictionInput = you have been selected for 10000rs. click link to claim
2	
3	
4	

Test Scenario On entering text and clicking on predict, a new tab should open with predictions

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to http://localhost:3000/spam	Should load spam detection page	Loaded spam detection page	Pass
2	Click predict without entering text	Message saying 'Please fill in this field' should pop up	Message saying 'Please fill in this field' was shown	Pass
3	Enter input text and then click predict	A new tab containing JSON response should load	A new tab with JSON response was loaded	Pass

Table 7.1.d – Test Case T004



Test Case ID	T004	**Test Case Description**	Test the API functionality for all 3 APIs		
Created By	Harsha K Y	**Reviewed By**		**Version**	1.0

QA Tester's Log

Tester's Name	Harsha K Y	**Date Tested**	May 5, 2020	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	
3	
4	

S #	Test Data
1	PredictionInput = 1917, the most recent film by acclaimed director Sam Mendes (Skyfall, American Beauty), is a phenomenon to say the least. With a magnificent combination of excellent editing, superb acting and an outstanding score from Thomas Newman, 1917 is a rare example of absolute cinematic perfection.
2	
3	
4	

Test Scenario Testing functionalities of all APIs with and without giving input

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://localhost:3000/api/sentiment?predict=	There is no input, so JSON response should be an error message saying the same	Got JSON response saying error: no input	Pass
2	Navigate to http://localhost:3000/api/category?predict=	There is no input, so JSON response should be an error message saying the same	Got JSON response saying error: no input	Pass
3	Navigate to http://localhost:3000/api/spam?predict=	There is no input, so JSON response should be an error message saying the same	Got JSON response saying error: no input	Pass
4	Navigate to http://localhost:3000/api/sentiment?predict=PredictionInput	Should load JSON response with predictions	Loaded JSON response with predictions	Pass
5	Navigate to http://localhost:3000/api/category?predict=PredictionInput	Should load JSON response with predictions	Loaded JSON response with predictions	Pass
6	Navigate to http://localhost:3000/api/spam?predict=PredictionInput	Should load JSON response with predictions	Loaded JSON response with predictions	Pass

8. CONCLUSION

- The goal of this project from the start was to build reliable, accurate APIs for NLP tasks.
- Having achieved that for 3 different tasks; Sentiment Analysis, Category Prediction, Spam Detection, it is safe to say that ML APIs are going to dominate in the near future.
- It is very easy for users to just integrate APIs for NLP tasks than build them from scratch.
- The tools and technologies used to build this application allow for easy integration of new features and new APIs.
- Having provided a page with documentation for the API, users are going to find it easy to make use of my APIs.
- Since most of the tasks are done on the browser itself, it requires very less computation power. This means that the APIs can be easily used for predictions on mobile devices as well.

9. FUTURE ENHANCEMENT

- There is a lot of potential still left for the project to fulfill.
- The accuracy can and should increase.
- The API itself should provide more features. Features like text labeling, summary generation, etc. should be implemented.
- Users and API keys specific to the user must be set up. As of now, the APIs are free for all and not specific to the users.
- Develop APIs for more NLP tasks.

APPENDIX A: BIBLIOGRAPHY

1. Marc Moreno Lopez, Jugal Kalita – **Deep Learning Applied to NLP** – ArXiv 2017
2. Pengfei Liu, Xipeng Qiu, Xuanjing Huang - **Recurrent Neural Network for Text Classification with Multi-Task Learning –** ArXiv 2016
3. Tensorflow.js - https://www.tensorflow.org/js
4. Node.js - https://nodejs.org/en/docs/
5. hbs - https://nodejs.org/en/docs/
6. Express.js - https://expressjs.com/en/5x/api.html
7. Keras - https://keras.io/api/
8. Gensim - https://radimrehurek.com/gensim/auto_examples/index.html
9. The imdb reviews dataset - https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
10. The news dataset - https://www.kaggle.com/rmisra/news-category-dataset
11. The spam detection dataset - https://www.kaggle.com/uciml/sms-spam-collection-dataset

APPENDIX B: USER MANUAL

1. The header contains links for Sentiment Analysis, Category Prediction, Spam Detection. Users can click on any one of these and the app will redirect the user to the appropriate page.
2. Every page has a text area where the user can input their text data. For example in the category prediction page, the UI contains a text area where the users can type in some text for which they want predictions.
3. After typing, the user can click on predict, and a new tab opens with JSON data which has predictions and probability scores.
4. The 'Context Analyzer API' section has documentation for the API. The users can use this documentation to find out how to integrate the APIs for their own use.
5. The code for the project can be found at https://github.com/HarshaKy/text-classifier
6. The project can be downloaded and run in development mode.
7. After downloading, navigate to the root of the project and run the following commands in the same order to run the project in development mode.

   i. npm install

   ii. npm start
8. Once the app is running, open a browser and navigate to https://localhost:3000/