GUIDELINES FOR MAJOR PROJECT 2020-21

a) The final exam will have <u>*Presentation*</u> followed by Viva-Voce at the department level.
b) Title of project, name of the student and supervisor, year of submission must be printed and embossed on the cover page of the Major project report. The report must be bound using <u>**blue color**</u> paper.
c) The main contents in the report should be typed in Times New Roman 12 font
d) The page no. should be numbered from Abstract in roman (i),(ii),(ii)….
e) The page no. from Chapter 1 should be in numeral 1,2,3,,,,,,,,,,
f) Spacing between consecutive lines should be 1.5.
g) Separate successive paragraphs by 9 points.
h) Margins of each page should be preferably 1.25 inch left side, 0.75 inch right hand side, one inch top and 1.5 inch bottom with page no. at bottom margin of one inch.
i) For each chapter at top right corner chapter no. should be written: for example; for chapter 1, it should be Chapter - 1 (Font 12, Times New Roman).
j) Title of the chapter should be typed in Font 14 (All Capital) and should be centered.
k) References will be arranged in the order in which they are referred in the report. Preferably, the Report should be organized as under:
i) Introduction (Chapter - 1, Essential, Detailing Scope of work with Objectives)
ii) Literature review of the project (Chapter - 2, Essential)
iii) Chapters describing the Work (Tables should be given the number based on their chapters; for example; for chapter 2 they should be Table 2.1, 2.2 and so on. <u>They should be written as per IEEE Syntax as under:</u>

TABLE 1.1
NO. OF LINES, RELAYS, VARIABLES AND CONSTRAINTS
FOR SOME POWER SYSTEMS

<u>Like Tables, Figures should be numbered, and should be titled as under”</u>
Fig. 2.2. Diagram showing near-end and far-end faults for relay

vi) Conclusions (Future Scope should be mentioned at the end of this chapter)

l) References (IEEE Syntax),
m) Appendix
n) The covering page and Candidate’s declaration must be as per pg. (3) and (4) respectively, and the details as mentioned in Page nos. (5) to (9) enclosed herewith must be filled-up and be submitted to the University besides all above. *The dotted blank entries in the covering page and Candidate's declaration {pg. (3) and (4)} must be filled-up, and later instructions described in the dotted lines in the subscripted form for filling-up must be removed.*

1. ARRANGEMENT OF CONTENTS:

The sequence in which the project report material should be arranged and Hard bound should be as follows:

01. Cover Page (no page numbering)
02. Certificate (recent decent color professional photo , no fancy photo) (no page numbering)
03. Acknowledgement (no page numbering)
04. Table of Contents (no page numbering)
05. Abstract (one page only) (Roman page number starts from here (i)
06. Project Summary
07. List of Tables
08. List of Figures (Roman numbering ends here)
09. Matter of project ***(Chapter wise (4-5 Chapters), Numeral page numbering 1,2,3.. starts from chapter 1 and continues till end of report)***
    - i. Chapter I – Introduction
    - ii. Chapter II- Literature Review
    - iii. System Analysis
        - a. Existing Work
        - b. Issues with Existing Work
        - c. Proposed Work
    - iv. System Design (includes use case diagram Data flow diagram, flow charts, Components used, design of front end and backend, Database used etc.)
    - v. Results (Screen shot, detailed results etc,)
    - vi. Conclusion Future Scope (Final conclusion, drawbacks, advantages, future scope of project)
    - vii. References
    - viii. Appendix (Any other supplementary material, like source code, any other detailed diagram)

Submission of Project Report Each Student is required to submit three copies of project report; one for RTU/department, one for supervisor and one for his/her own copy. The range of pages in the project report should be limit to 45-60 pages including every content (i.e from cover page).

The format for report is given in Annexure – I. The abstract, acknowledgement provided in annexure I are just sample. Please write your own matter properly.

A

Project Report

On

Emojify

Submitted
in partial fulfillment
for the award of the Degree of
Bachelor of Technology
in Department of Computer Science and Engineering

Engineering College Bikaner

(Rajasthan Technical University, Kota)
Session 2020-21

Submitted By:
Kartik Kumar

Submitted To:-
Mr. Rishiraj Vyas

CERTIFICATE

This is to certify that the Project Report Titled **"Emojify"** has been submitted by "Kartik Kumar" in partial in fulfilment for the requirement of the degree of Bachelor of Technology, Final Year for the academic Session 2020-2021

This seminar work is carried out under the supervision and guidance of Mr. Rishiraj Vyas and he has undergone the requisite work as prescribed by Rajasthan Technical University, Kota.

Mr Kartik Kumar
Roll No: 17EEBCS302
Branch: CSE
E-mail: mysticfreak1499@gmail.com

ECB, Bikaner

Date: 19/06/2021
Place: Bikaner

ACKNOWLEDGEMENT

This is an opportunity to express my heartfelt words for the people who were part of this project in numerous ways, people who gave me unending support right from the beginning of the training project report.

I am grateful to the project guide **Mr. Rituraj Soni** for giving guidelines to make the training seminar successful.

I want to give sincere thanks to the principal, **Prof. J.P. Bhamu** for his valuable support.

I extend my thanks to **Mr. Rishiraj Vyas,** CSE Department for his cooperation and guidance.

Yours

Sincerely,

Kartik Kumar

Table of Contents

ABSTRACT

Emojis are small images that are commonly included in social media text messages. The combination of visual and textual content in the same message builds up a modern way of communication. Despite being widely used in social media, emojis' underlying semantics have received little attention from a Natural Language Processing (NLP) standpoint. In this project, I investigate the relation between words and emojis, studying the novel task of predicting which emojis are evoked by text-based tweet messages. I experimented with variants of word embedding techniques, and trained several models based on Sentiment analysis using CNNs and classification using RNNs respectively. My experimental results show that the model can predict reasonable emoji from tweets.

Project Summary

Project Title	Emojify web application	
Project Team Members (Name with Register No)	Kartik Kumar	
Guide Name/Designation	Mr RITURAJ SONI Assistant Professor, Department of Computer Science and Engineering	
Program Concentration Area	Deep Learning	
Technical Requirements	Python - Numpy, Pandas, Scikit-learn, Tensorflow, Keras	
Engineering standards and realistic constraints in these areas		
Area	**Codes & Standards / Realistic Constraints**	**Tick ✓**
Economic	This project is developed using open source software free of cost	✔
Sustainability	The project ensures sustainability as after deployment it doesn't need much change.	✔
Social	This project is useful for a general audience of any age	✔
Ethical	This project is designed keeping in mind the need for people of all age groups	✔

LIST OF FIGURES

LIST OF ABBREVIATIONS

SVM	Support Vector Machine
NB	Naive Bayes
CFM	Confusion Matrix
LSTM	Long Short Term Memory
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
NLP	Natural Language Processing

CHAPTER 1

INTRODUCTION

People use emojis every day??. Emojis have become a new language that can more effectively express an idea or emotion. This visual language is now a standard for online communication, available not only in Twitter, but also in other large online platforms such as Facebook and Instagram. Right now, the keyboard on iOS can predict emojis but only based on certain keywords and tags that are associated with emojis.

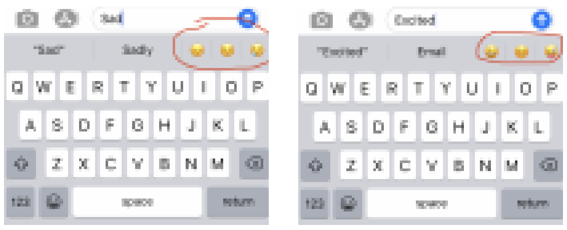


Figure 1.1: Examples of emoji prediction in iOS13 keyboard

Emoji prediction is a fun variant of sentiment analysis. When texting your friends, emoji can make your text messages more expressive. It would be nice if the keyboard can predict emojis based on the emotion and meaning of the whole sentence you typed out. Moreover, despite its status as a language form, emojis have been so far scarcely studied from a Natural Language Processing (NLP) standpoint. The interplay between text based messages and emojis remains virtually unexplored.

In this application project, we aim to fill this gap by investigating the relation between words and emojis, studying the problem of predicting which emojis are evoked by text based tweet message. We build classifiers that learn to associate emojis with sentences. The models we explore here are Multinomial Naive Bayes and Bidirectional LSTM. Standard Bag of Word TF-IDF and pre-trained GLoVe model are used as word embedding, respectively. We train a large dataset of sentences with emojis labels aggregated from Twitter messages. In the Inference stage, the trained classifier takes as input a sentence and finds the most appropriate emoji to be used with this sentence.

CHAPTER 2

LITERATURE REVIEW

2.1. Emojis

Emojis, also known as ideograms or smileys, can be used as compact expressions of objects, topics and emotions. Being encoded in Unicode, they have no language barriers and are diffused on the Internet rapidly. The prevalence of emo jis has attracted researchers from various research communities such as NLP, multimedia and data mining [6]. The various non-verbal functions of emojis play an important role in their wide adoption to the extent that they can have their unique linguistic purpose alongside written text [5]. Generating emojis automatically from a sentence is essentially a text classification problem. In [1], the author use millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm.

2.2. Sentiment Analysis

sentiment analysis refers to using the NLP techniques to study the subjective information from a sentence. Currently, sentiment analysis is a topic of great interest and de

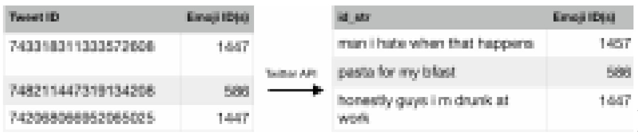
4321

velopment, since it can be widely used for getting insights from social media comments, survey responses, product reviews, and making data-driven decisions. Some applications, such as recommender systems employ binary classification to determine if a user expresses like or dislike of some product or service or even polarity [4] of a statement. Other forms of sentiment analysis use unique, categorical labels similar to the emoji tags to predict on those distinct emotions, such as happy, confused, tired, surprised, etc. Since emojis have multi-contextual representation and are readily used across all languages [2], it serves as a great sentiment label that can encapsulate the nuances in a sentence.

3. Dataset and Features

3.1. Dataset

Given the prevalence of emoji usage and digital opinions on Twitter, it was obvious to extract a rich dataset from Twitter. The Twemoji dataset provides an extensive set of almost 13 million tweet status ID and emoji annotation per tweet. We fetched the corresponding tweets through the twitter API with the provided tweet ID to aggregate our dataset of sentence-emoji ID pairs. The data collection process is shown in Fig 2.



Tweet ID	Emoji ID(s)
743318311333572608	1447
748211447319134208	586
742068088952065025	1447

Twitter API

id_str	Emoji ID(s)
man i hate when that happens	1457
pasta for my bfast	586
honestly guys i m drunk at work	1447

Figure 2.1: Twitter data API

3.2. Pre-processing

From the Twemoji dataset, we extracted 167,685 sentence - emoji pairs for our training. However, the dataset is heavily unevenly distributed, as shown in Fig 3, where emoji has over 50% representation power of the whole dataset. So we have done several data preprocessing steps before training.

- noise removal, filter out emojis which have less than 10000 correspondence sentences.
- stop emojis, like the stopwords technique in NLP, remove high frequent emojis which are everywhere and do not have specific semantic meaning.
- data un-bias, equalize the number of sentences for each emoji.

After the data pre-processing, the dataset is reduced to 13251 valid examples which belong to 13 emoji categories. The emoji we used in our project is shown in Fig 4.

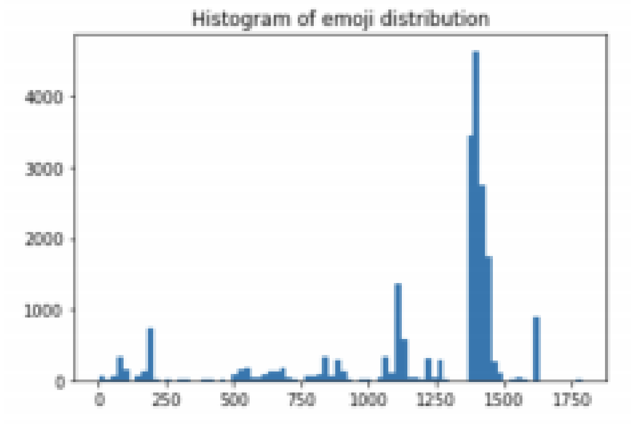
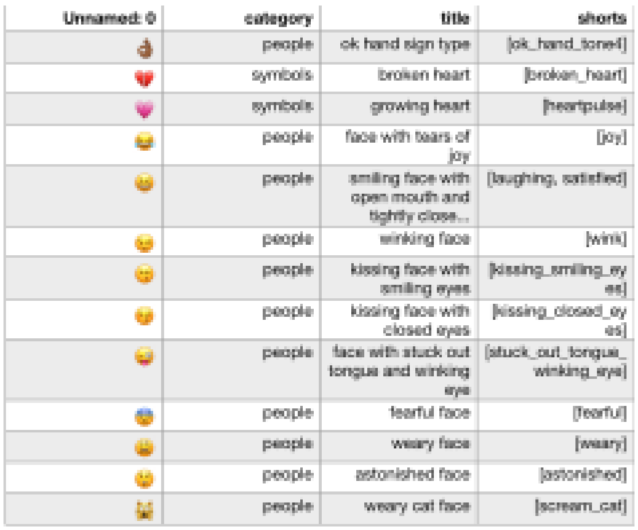


Figure 2.2: Raw data class distribution



Unnamed: 0	category	title	shorts
👌🏾	people	ok hand sign type	[ok_hand_tone4]
💔	symbols	broken heart	[broken_heart]
💗	symbols	growing heart	[heartpulse]
😂	people	face with tears of joy	[joy]
😆	people	smiling face with open mouth and tightly close...	[laughing, satisfied]
😉	people	winking face	[wink]
😙	people	kissing face with smiling eyes	[kissing_smiling_eyes]
😚	people	kissing face with closed eyes	[kissing_closed_eyes]
😜	people	face with stuck out tongue and winking eye	[stuck_out_tongue_winking_eye]
😨	people	fearful face	[fearful]
😩	people	weary face	[weary]
😲	people	astonished face	[astonished]
🙀	people	weary cat face	[scream_cat]

Figure 2.3 : Emoji category after data pre-processing

4. Method

Predicting emojis is more like a sentence classification problem. We first investigate the word embedding techniques to use, then try machine learning algorithms we learned in the class, such as Multinomial Naive Bayes, SVM and Bi-LSTM to do the emoji prediction.

4.1. Word Embedding

BoW-TFIDF Instead of representing the sentences with matrix of corpus and counts like we've seen in class, we can employ Bag of Words representation with TF-IDF trying to capture more indicative keywords in a sentence. The dictionary size is 1834 after stop words and stemming. We can feed this into our traditional methods for Naive Bayes and SVM.

$$W_{i,j} = tf_{i,j} \times \log(^{N}df_i) \quad (1)$$

where $tf_{i,j}$ is the number of occurrences of i in j, dfi *is* the number of documents containing i, while N is the total number of documents in corpus.

GLoVe For the deep learning and SVM methods, we tried to use the pre-trained GLoVe to see if adding in global statistics of our Twitter corpus would help with neural net performance. GLoVe is an unsupervised trained model which maps words into a meaningful space where the distance between words is related to semantic similarity. In this project we used the pre-trained model GLoVe-50 and GLoVe-300 from Stanford. From there, we split our dataset into 90% training and 10% test.

4.2. Multinomial Naive Bayes Classifier

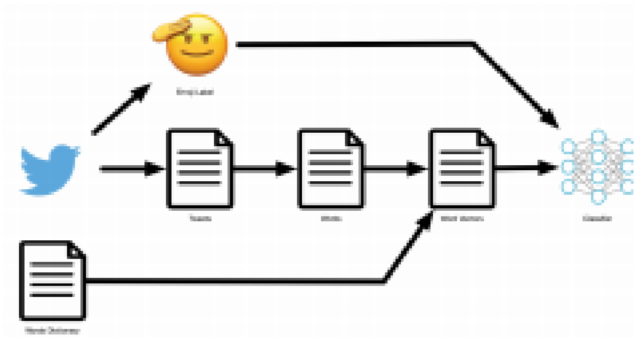


Figure 2.4: Overall pipeline

The whole pipeline for NB classifiers is shown as Fig 5. From twitter we can collect data which contains tweets, messages and its corresponding emoji as a label. Then by using a word dictionary we transform texts into word vectors via a bag of words with tf-idf as described above. Finally here we apply a Multinomial Naive Bayes Classifier to train the model for text classification. Since Naive Bayes is a simple classifier to use, it acts as a good baseline and starting point to tackle this problem.

Similarly, we also employed the pipeline to SVM. 4.3. Bi-LSTM Classifier

For the RNN model, the first layer is the embedding layer, embedding layer will represent each word as a matrix. The embedding layer is loaded from a Pre-trained GLoVe model and the weight is fixed during training. The Glove will map similar words to close vectors in high-dimensional feature space. The next layers are two 1-D convolution layers, followed by a bidirectional LSTM layer. LSTM is a type of RNN network. It allows previous outputs to be used as inputs while having hidden states, so it is suitable for analyzing sentences. RNN suffers from the vanishing gradient problem which can be handled by LSTM. Lastly, we use a softmax layer which outputs a vector that represents the probability distribution of a list of potential emoji. The network architecture is shown in Fig 6

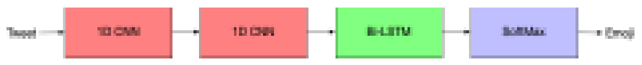


Figure2.5: LSTM Architecture

The code is implemented in Keras. After each convolution and LSTM layer, ReLU activation and batch normalization is used. We use Adam as an optimizer, the learning rate is 0.001. The LSTM layers have a L2 regularization with rate to be 0.01 to prevent over-fitting. The training batch size is 128 and we trained 100 epochs.

5. Experiments

5.1. overall accuracy

Right now the twitter dataset we collected contains 12,997,220 tweets, and there are 1,791 emojis associated with these tweets. From our preprocessing, we reduced it to 167,685 tweets and 13 emojis. To evaluate the model, we split 90% tweets into a training set and 10% into a testing set. The results from test sets will be assessed using accuracy scores. Our current accuracy on these dataset is shown in Tab 1.

Word Embedding	BoW	GLoVe-50	GLoVe-300
Multinomial NB	19.53 %	N/A	N/A
SVM	9.195 %	16.376%	14.966%
Deep CNN	N/A	15.168%	15.906%
Deep Bi-LSTM	N/A	15.705%	15.57%

Table 1: Emoji prediction accuracy

5.2. Result Analysis

Here we did some further analysis on each method.

5.2.1 Naive Bayes and SVM Classifiers:

Here is some analysis on the performance of Naive Bayes Classifier. The model we choose is the multinomial Naive Bayes model because it can take word frequency into account. The word dictionary's size is 2791, which is created from just a training set.

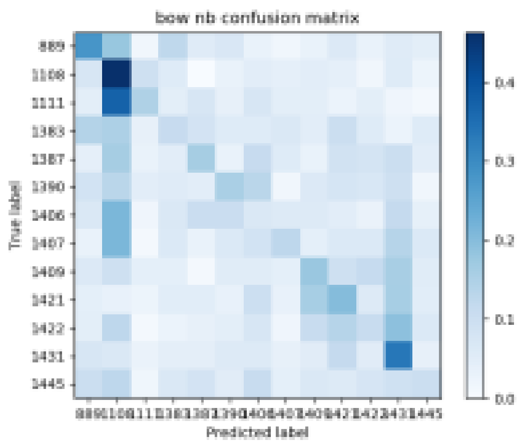


Figure 2.6: Confusion Matrix for Naive Bayes

The overall NB classification accuracy is 19.53%, while SVM did quite poorly with only 9.1% on bag of word embeddings and much better on GLoVe embeddings, as shown in Fig 7 and 8.

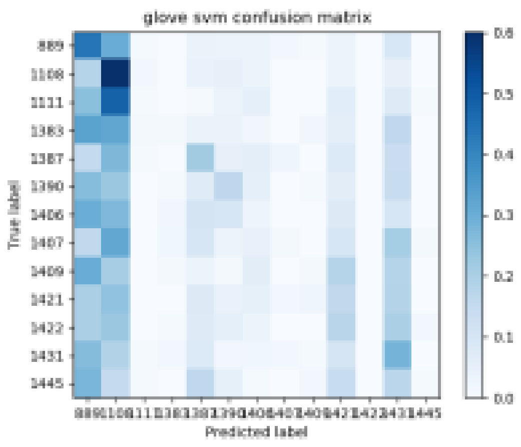
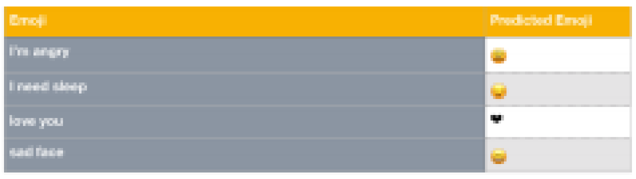


Figure 2.7: Confusion Matrix for SVM

Here are some prediction examples using this trained model. Some make sense and some are just wrong.



Emoji	Predicted Emoji
I'm angry	😀
I need sleep	😀
love you	♥
sad face	😀

Figure 2.8: prediction results from naive bayes classifier

In addition, we also know that if we further reduce the number of emojis to classify with naive bayes and thereby
create an even more balanced distribution of the dataset, say to top 5 balanced classes with sufficient data, we see that the test accuracy can improve up to 41%.

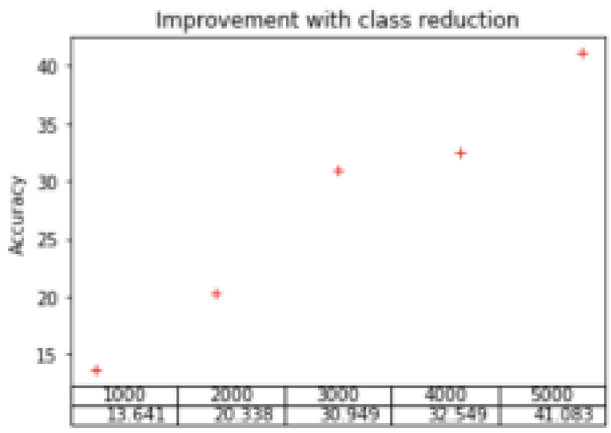
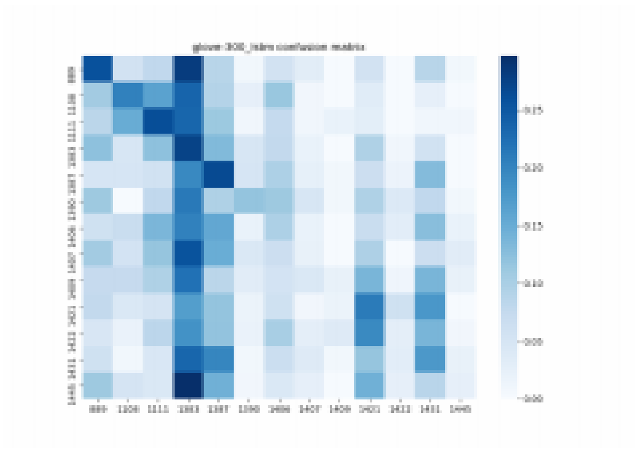


Figure 2.9: Accuracy improvement after reduce categories

5.2.2 Bi-LSTM Classifiers



We also show the results of Bi-LSTM with the GloVe-300 confusion matrix in Fit 11.

Figure 2.10: Confusion Matrix for LSTM with

CHAPTER 3

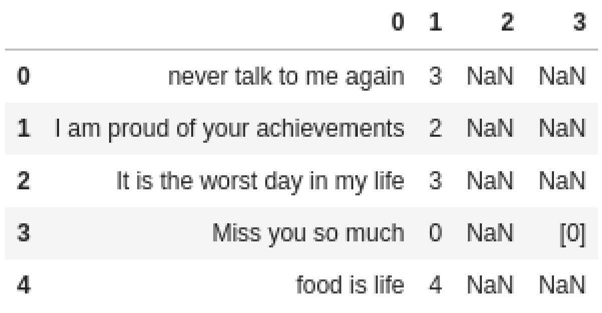
SYSTEM ANALYSIS

Emogify has performed well on a number of platforms and has proved on par with expectations. We shall now discuss the current prototype and the problems with this version, following which, which will discuss future expectations.

3.1 EXISTING WORK

We have used Keras as the principal deep learning library while taking occasional help of numpy, pandas and sklearn for preprocessing and matplotlib for visualization. Also, for the dataset preparation a special library called **emoji** has been used which forms the base of all the classification.

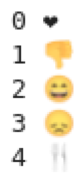
After importing the dataset and splitting into training and testing batches, this is how the first five rows look:



	0	1	2	3
0	never talk to me again	3	NaN	NaN
1	I am proud of your achievements	2	NaN	NaN
2	It is the worst day in my life	3	NaN	NaN
3	Miss you so much	0	NaN	[0]
4	food is life	4	NaN	NaN

Figure 3.1 head() of dataset

The first column [0] represents the sentences and the second column [1] represents the index of emoji. Below is a small set of emojis against their indices for example:



```
0 ❤
1 👎
2 😄
3 😞
4 🍴
```

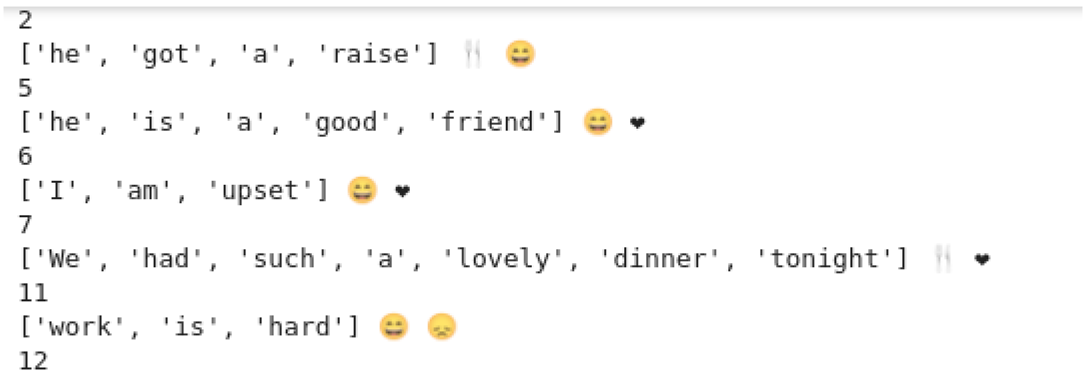
Figure 3.2 Emojis

We have then converted the sentences into arrays of words with help of sklearn's OneHotEncoder. after a slight look-at at the data for instance finding cosine distances between different labels etc, we have the final fully prepared data which is then ready to be fed into our RNN.

3.1.1 APPROACH 1 : USING RNN

Next we use tensorflow keras for the training. We have run 50 epochs of our prepared RNN model on each of the 132 training examples.
The final predictions were not very accurate though. We achieved around **60% accuracy** with RNNs.

Below are some sample predictions:

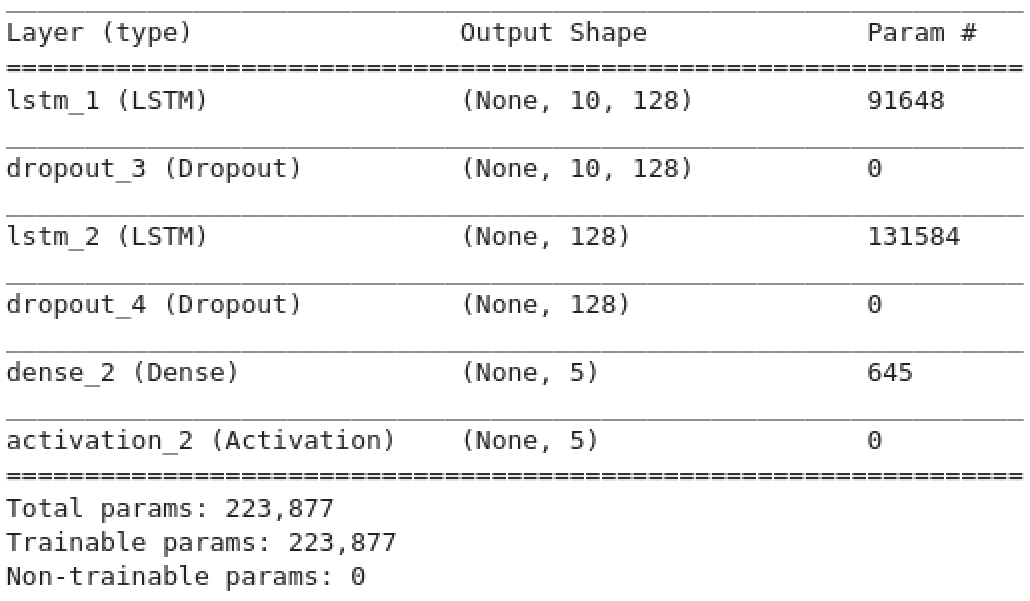


```
2
['he', 'got', 'a', 'raise'] 🍴 😄
5
['he', 'is', 'a', 'good', 'friend'] 😄 ❤
6
['I', 'am', 'upset'] 😄 ❤
7
['We', 'had', 'such', 'a', 'lovely', 'dinner', 'tonight'] 🍴 ❤
11
['work', 'is', 'hard'] 😄 😞
12
```

Figure 3.3 RNN predictions

3.1.1 APPROACH 1 : USING LSTM

Again, using keras Sequential model, but for LSTM this time, we achieved improved accuracies. The LSTM model we prepared looks like this :

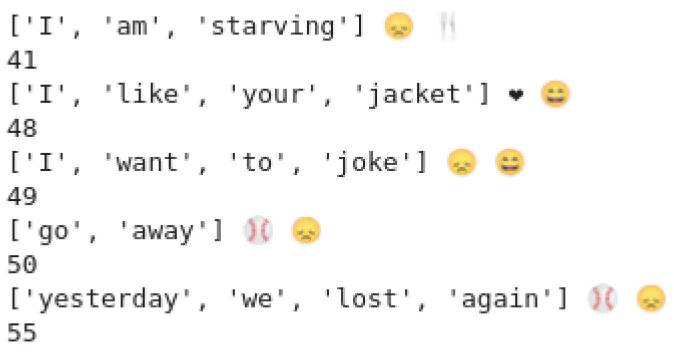


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 10, 128)           91648
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 128)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
_________________________________________________________________
activation_2 (Activation)    (None, 5)                 0
=================================================================
Total params: 223,877
Trainable params: 223,877
Non-trainable params: 0
```

Figure 3.4 LSTM Architecture

After 50 epochs we achieved around a fabulous **70 % accuracy** with minor adjustments. Clearly RNNs didn’t stand by the promise they showed in other projects. LSTM has proven efficiently powerful.

Here are sample predictions using LSTM:



```
['I', 'am', 'starving'] 😞 🍴
41
['I', 'like', 'your', 'jacket'] ❤ 😄
48
['I', 'want', 'to', 'joke'] 😞 😄
49
['go', 'away'] ⚾ 😞
50
['yesterday', 'we', 'lost', 'again'] ⚾ 😞
55
```

Figure 3.5 LSTM predictions

3.2 ISSUES WITH EXISTING SYSTEM

One profound issue with the existing version 1.0.0 of emojify is that most of the training happens on local machines. We are trying to expand it to the cloud by deploying the prediction and testing through a web app.

The choice of language seems a major issue. Although there are a plenty platforms for web apps like,

- streamlit Python library, but it’s not scalable and powerful enough.
- Django, needs a larger support team
- Java, Jython are needed , which we are not willing to deploy at this stage of project since it will be cumbersome to manage for such a small project.

CHAPTER 4

SYSTEM DESIGN

4.1 COMPONENTS USED

4.1.1 Software Requirements

The target systems for our project reach a vast span and thus requirements had to be kept minimal. Software requirements include:

- Python 3.5 or higher.

Figure 4.1 Python Logo

- All the necessary libraries: Numpy, Pandas, Matplotlib, Tensorflow, Keras.

Figure 4.2 Keras Logo

- Jupyter Notebook (only for accessing source code)

4.1.2 Hardware Requirements

Hardware requirements, just like the software requirements, are not a big deal for running Emogify. The following are the requirements:

- OS : Ubuntu 16.0 or later / Windows 7 or later
- CPU : Intel Pentium 4 or higher
- RAM : 2 GB or higher
- Hard Disk Space : Negligible (a few kilobytes)
- GPU : Not required, but preferred.

4.1.3 Component Description

Components used, as the reader can see, are of minimal nature. The following is the description of each.

- Python 3.5 or higher is required because Tensorflow does not support the previous versions of Python. Since the majority of the work is done in keras, higher versions of python are preferable.
- Tensorflow and keras are both version 2.2 and 2.0-tf respectively.
- We have made use of Jupyter Notebook, so if the reader seeks access to the source code, either Jupyter Notebook or similar Notebook environments such as Google Colab should be used. However, any other IDE or text editor will suffice.
- Again, the operating system requirements are actually the requirements of Tensorflow, otherwise earlier OSs would have also proved efficient.
- GPU is not required, but for future versions of Emojify, we advise you to use a GPU as Tensorflow works well with GPUS(only Nvidia cards).

4.2 IMPLEMENTATION OF THE PROJECT

Most of the project is on front end and is console based. However, the future updates might include backend work as well which we wish to carry out in Django. As of now, the project is at the front end with minimal to no network requirement. There was a specific reason to omit the backend work for now. First, I wanted to produce a flexible code that could be fed into any other app by any organisation using my code. For these obvious reasons, the GUI had to be omitted. Secondly, most of the work had to be console based requiring no internet so that users can get results offline on their keyboards as soon as they type, hence diminishing network lag.

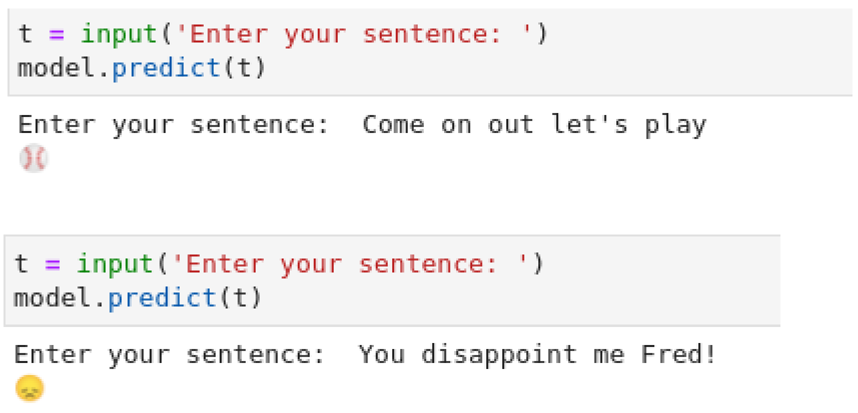
The front end work includes carrying out LSTM and RNN model training on an emoji dataset. Also, gloves are used for improved results as it provides an augmented dataset with thousands of additional sentences. For the backend, there is no support right now. However, I am looking for backend support to be added soon so as to impart a finish.

CHAPTER 5

RESULTS AND CONCLUSION

5.1 RESULTS

The project has shown promising results so far. And constant improvements are being embedded through regular updates. Below is the implementation and working with results and conclusions.



```
t = input('Enter your sentence: ')
model.predict(t)
```

```
Enter your sentence:  Come on out let's play
⚾
```

```
t = input('Enter your sentence: ')
model.predict(t)
```

```
Enter your sentence:  You disappoint me Fred!
😞
```

Figure 5.1 Sample tests

The above outputs prove the optimal results we obtained and the model is constantly improving.

The first line takes a string input and the second line outputs the suggested emoji using the keras model that we trained. In these examples, it is 100 percent accurate, however, I have tested emojify for a 70 percent accuracy over a large range of examples. It is supposed to work even better after embedding into a keyboard that I hope to do soon.

5.2 FUTURE PROSPECTS

As of now, the project dwells on local machines with zero backend support. However, I plan to move the system to the cloud for training and testing with only predictions taking place on the user's local machine. I also intend to add a bit of reinforcement learning so that the system learns user behavior after being embedded into a keyboard. A complete mobile application in Kotlin and Java with Django backend is under progress in which I can finally embed this machine learning code to make it a foolproof self learning keyboard application.

CHAPTER 6

REFERENCES

6.1 Textbooks

- Hands on Machine Learning with Scikit learn, Tensorflow and Keras - Aurelion Geron

  https://www.amazon.in/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1491962291

- Deep Learning Cookbook - Douwe Osinga

  https://www.amazon.in/Deep-Learning-Cookbook-Practical-Recipes/dp/9352137574/ref=sr_1_3?dchild=1&keywords=deep+learning+cookbook&qid=1624264945&s=books&sr=1-3

- Python - Lutz M.

  https://www.amazon.in/Learning-Python-Powerful-Object-Oriented-Programming/dp/9351102017/ref=sr_1_3?dchild=1&keywords=python+oreilly&qid=1624264996&s=books&sr=1-3

- Machine Learning : A Probabilistic Perspective - Kevin P. Murphy

  https://www.amazon.in/dp/0262018020/?coliid=I11M08I9KTQQ3O&colid=2M7L2OO4FUKC&psc=1&ref_=lv_ov_lig_dp_it

- Practical Natural Language Processing - Gupta and Majumber

  https://www.amazon.in/dp/1492054054/?coliid=ITRPZQK6TF8K3&colid=2M7L2OO4FUKC&psc=1&ref_=lv_ov_lig_dp_it

- Deep Learning - MIT Press (Goodfellow, Bengio and Courville)

  https://www.deeplearningbook.org/

6.1 Other Resources

- Machine Learning - Andrew Ng on Coursera

  https://www.coursera.org/learn/machine-learning?utm_source=gg&utm_medium=sem&utm_campaign=07-StanfordML-IN&utm_content=07-StanfordML-IN&campaignid=1950458127&adgroupid=71501032500&device=c&keyword=coursera%20machine%20learning&matchtype=p&network=g&devicemodel=&adpostion=&creativeid=415449761695&hide_mobile_promo&gclid=CjwKCAjw8cCGBhB6EiwAgORey8Fv-naa1qY61s9zcLj4cBj2T3uGEguyQaBmE59-OvG2UgkY833N7xoCM20QAvD_BwE

- Python Predictive Analysis - University of Edinberg

  https://www.edx.org/course/introduction-to-predictive-analytics-using-python

- Natural Language Processing - Lawrence Muroney

  https://www.coursera.org/learn/natural-language-processing-tensorflow?specialization=tensorflow-in-practice&utm_source=gg&utm_medium=sem&utm_campaign=33-DeepLearningAI-TensorFlow-IN&utm_content=33-DeepLearningAI-TensorFlow-IN&campaignid=12462557662&adgroupid=120411989496&device=c&keyword=&matchtype=b&network=g&devicemodel=&adpostion=&creativeid=510017701427&hide_mobile_promo&gclid=CjwKCAjw8cCGBhB6EiwAgORey8TTs4u6peEXmAZNDMlkZNGQtijvf1HmCilVXPENJfGJwHtp5EUT4BoCT_0QAvD_BwE

- Statistics with Python - University of Michigan

  https://www.coursera.org/specializations/statistics-with-python