# Introduction

I am performing sentiment analysis on data available on TripAdvisor using python. The sentiment analysis is performed multi-layered fully connected Neural Network. In past several years, Neural Networks (NNs) have become the state-of-the-art solution for many applications.

*Problem Statement*

*“TripAdvisor is the world’s largest travel site where you can compare and book hotels, flights, restaurants etc. The data set I am utilizing consists of a sample of hotel reviews provided by the customers. Analyzing customers reviews will help them understand about the hotels listed on their website i.e. if they are treating customers well or if they are providing hospitality services as expected. In this challenge, you have to predict if a customer is happy or not happy.”*

*Data-set Description*

There are two files incorporated as data after downloading: train.csv, test.csv. The training data has 38932 rows, while the test data has 29404 rows. The dataset can be downloaded from [here](https://www.dropbox.com/s/o8jeborvubfs7fn/trip_advisor_hackerearth_data.zip?dl=0).



To solve the problem I am interested only in 2 columns. ‘Description’ which contains hotel reviews given by different users and ‘Is\_Response’ which keeps the record of ‘happy’ or ‘not\_happy’. So, in essence, this is a 2-class sentiment analysis problem.

The steps followed can be summarized as follows:

1. Prepare data.
2. Feature Extraction.
3. Build the Model.
4. Train the Model.
5. Evaluating Performance.

# Data Preparation

Training data is provided in .csv format which can be ingested easily with pandas as shown in the code below. ‘Is\_Response’ field of data carries strings, i.e. ‘happy’ and ‘not\_happy’, which needs to be encoded in integer format, i.e. 0 and 1. Here, it is done by LabelEncoder class of scikit-learn library. The function returns list of a hotel reviews and their respective happiness labels.

|  |
| --- |
| **def** data\_prepare(training\_file\_path): |
| dataset **=** pd.read\_csv(training\_file\_path) reviews **=** [] labels **=** []    # Enconding Categorical Data labelencoder\_y **=** LabelEncoder()  dataset['Is\_Response'] **=** labelencoder\_y.fit\_transform(dataset['Is\_Response']) cLen **=** len(dataset['Description'])  **for** i **in** range(0,cLen):  review **=** dataset['Description'][i] reviews.append(review)  label **=** dataset["Is\_Response"][i] labels.append(label) labels **=** np.asarray(labels) **return** reviews,labels |

# Feature Extraction

In this task, words are features, hence the bag-of-words model can be used to create a feature vector. It can be done in following steps:

1. Make a dictionary: We create a dictionary containing word-index tuples of all the distinct words in training text reviews. We assume that the ordering of words is not important.
2. Convert words of each text review into word index array and store the index array of each review in global array. Example of a text review –

The room was kind of clean but had a VERY strong smell of dogs. Generally below average but ok for a overnight stay if you're not too fussy. Would consider staying again if the price was right. Breakfast was free and just about better than nothing.

[1, 14, 5, 436, 9, 52, 17, 25, 3, 22, 1735, 628, 9, 1727, 1109, 943, 492, 17, 322, 11, 3, 1010, 34, 42, 411, 24, 131, 3754, 40, 941, 181, 72, 42, 1, 126, 5, 117, 60, 5, 89, 2, 56, 64, 172, 100, 268]

1. Convert the global array of index into a feature matrix. Each text review is represented by a sparse vector of the size of the vocabulary, with 1 in the entries representing the word and 0 in all other entries. We use the maximum number of features as 10,000. Thus the final feature matrix will be of shape (38392,10000).

# Build the Model

It is very easy to build a NN model using Keras. In this solution, I have used a fully connected, 2-hidden layered neural network. We need the Sequential module for initializing a NN and the Dense module to add the hidden Layers. In the output layer, there are 2 nodes, one for the positive and another for the negative sentiment class. The Python code for building the model is shown below:

|  |
| --- |
| # Creating a Dense Neural Network Model |
| model **=** Sequential()  model.add(Dense(256, input\_shape**=**(max\_words,), activation**=**'elu')) model.add(Dropout(0.5)) |
| model.add(Dense(128, activation**=**'elu')) |
| model.add(Dropout(0.5))  model.add(Dense(2, activation**=**'softmax')) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

==================================================================

dense\_1 (Dense) (None, 256) 2560256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_1 (Dropout) (None, 256) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_2 (Dense) (None, 128) 32896 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_2 (Dropout) (None, 128) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 2) 258

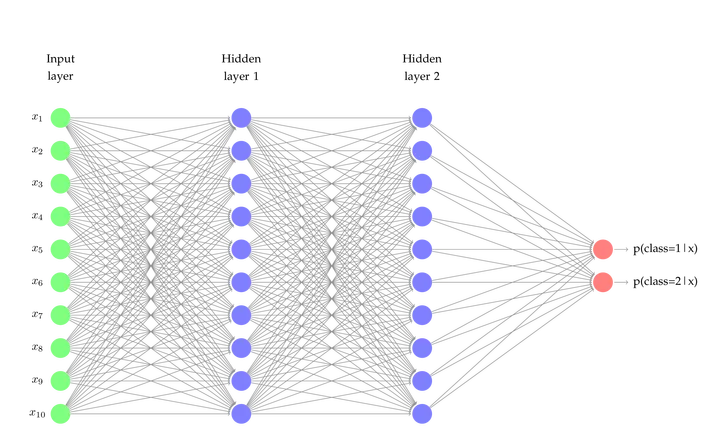
==================================================================

Total params: 2,593,410

Trainable params: 2,593,410

Non-trainable params: 0

# Training the Model



A typical 2-hidden layered fully connected NN

Now, we have completed both feature extraction and model building. For training the model, it is required to first compile the model with categorical cross entropy loss function and stochastic gradient descent learning algorithm. Once compiled, one can train the model by utilizing the GPU of their system. In Keras, it can be done as :

|  |  |
| --- | --- |
| 1 | model.compile(loss**=**'categorical\_crossentropy', optimizer**=**'sgd', metrics**=**['accuracy' |
| 2 | model.fit(train\_X, labels, batch\_size**=**32, epochs**=**5, verbose**=**1, validation\_split**=**0.1 |

While training the model, I pass the feature matrix, the labels, input batch size to process, the number of iterations etc. as parameters. I also save the dictionary and the NN model in order to use them later while performing predictions on the test data. Once the NN model has been trained, I can check the performance of the model on test .csv data.

Below is the entire code for training the NN model for sentiment analysis application.

|  |
| --- |
| **import** numpy as np |
| **import** pandas as pd  **from** sklearn.preprocessing **import** LabelEncoder **import** json **import** keras  **import** keras.preprocessing.text as kpt **from** keras.preprocessing.text **import** Tokenizer  **from** keras.models **import** Sequential **from** keras.layers **import** Dense, Dropout  **def** convert\_text\_to\_index\_array(text):  **return** [dictionary[word] **for** word **in** kpt.text\_to\_word\_sequence(text)]    train\_file\_path **=** "./train.csv"  [reviews,labels] **=** data\_prepare(train\_file\_path)  # Create Dictionary of words and their indices max\_words **=** 10000  tokenizer **=** Tokenizer(num\_words**=**max\_words) tokenizer.fit\_on\_texts(reviews) dictionary **=** tokenizer.word\_index  # save dictionary with open('dictionary.json','w') as dictionary\_file:  json.dump(dictionary,dictionary\_file)  # Replace words of each text review to indices allWordIndices **=** [] **for** num,text **in** enumerate(reviews):  wordIndices **=** convert\_text\_to\_index\_array(text) allWordIndices.append(wordIndices)    # Convert the index sequences into binary bag of words vector (one hot encoding) allWordIndices **=** np.asarray(allWordIndices)  train\_X **=** tokenizer.sequences\_to\_matrix(allWordIndices, mode**=**'binary') labels **=** keras.utils.to\_categorical(labels,num\_classes**=**2)  # Creating Dense Neural Network Model model **=** Sequential()  model.add(Dense(256, input\_shape**=**(max\_words,), activation**=**'elu')) model.add(Dropout(0.5))  model.add(Dense(128, activation**=**'elu'))  model.add(Dropout(0.5))  model.add(Dense(2, activation**=**'softmax'))    model.compile(loss**=**'categorical\_crossentropy', optimizer**=**'sgd', metrics**=**['accuracy'])    # Training the Model model.fit(train\_X, labels, batch\_size**=**32, epochs**=**5,  verbose**=**1,  validation\_split**=**0.1, shuffle**=**True)    # Save model to disk model\_json **=** model.to\_json() with open('model.json', 'w') as json\_file:  json\_file.write(model\_json) model.save\_weights('model.h5') |

The IPython console will show the training accuracy and the validation accuracy in iterations like this:

Train on 35038 samples, validate on 3894 samples

Epoch 1/5

35038/35038 [==============================] - 7s 207us/step - loss: 0.3985 - acc: 0.8233 - val\_loss: 0.3168 - val\_acc: 0.8675

Epoch 2/5

35038/35038 [==============================] - 6s 161us/step - loss: 0.3162 - acc: 0.8701 - val\_loss: 0.3019 - val\_acc: 0.8760

Epoch 3/5

35038/35038 [==============================] - 6s 163us/step - loss: 0.2982 - acc: 0.8780 - val\_loss: 0.2934 - val\_acc: 0.8770

Epoch 4/5

35038/35038 [==============================] - 6s 163us/step - loss: 0.2849 - acc: 0.8853 - val\_loss: 0.2960 - val\_acc: 0.8837

Epoch 5/5

35038/35038 [==============================] - 6s 163us/step - loss: 0.2776 - acc: 0.8877 - val\_loss: 0.2971 - val\_acc: 0.8765

# Performance Evaluation

The test.csv file provided consists of 29404 hotel reviews. We will now predict the sentiment for all the hotel reviews.

To check the performance of the “Predict the Happiness” system, the trained dictionary and the NN model is loaded. For each of the hotel reviews, I extract the bag of word features in a similar way as in training. The softmax scores of the output layer are calculated by feedforwarding the input features to the trained NN model. A higher score shows more probability of that sentiment. Finally, the prediction csv file is written with User\_ID and the predicted response. The Python code for performing predictions on the test data is shown below.

1 **import** json

|  |  |
| --- | --- |
| 2 | **import** numpy as np |
| 3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | **import** keras.preprocessing.text as kpt **from** keras.preprocessing.text **import** Tokenizer **from** keras.models **import** model\_from\_json  **import** pandas as pd  **def** convert\_text\_to\_index\_array(text):  words **=** kpt.text\_to\_word\_sequence(text) wordIndices **=** [] **for** word **in** words: **if** word **in** dictionary:  wordIndices.append(dictionary[word]) **return** wordIndices    # Load the dictionary labels **=** ['happy','not\_happy'] with open('dictionary.json', 'r') as dictionary\_file:  dictionary **=** json.load(dictionary\_file)  # Load trained model  json\_file **=** open('model.json', 'r') loaded\_model\_json **=** json\_file.read() json\_file.close()  model **=** model\_from\_json(loaded\_model\_json)  model.load\_weights('model.h5')    testset **=** pd.read\_csv("./test.csv") cLen **=** len(testset['Description']) tokenizer **=** Tokenizer(num\_words**=**10000)  # Predict happiness for each review in test.csv y\_pred **=** [] **for** i **in** range(0,cLen):  review **=** testset['Description'][i]  testArr **=** convert\_text\_to\_index\_array(review)  input **=** tokenizer.sequences\_to\_matrix([testArr], mode**=**'binary') pred **=** model.predict(input)  #print pred[0][np.argmax(pred)] \* 100, labels[np.argmax(pred)]  y\_pred.append(labels[np.argmax(pred)])    # Write the results in submission csv file raw\_data **=** {'User\_ID': testset['User\_ID'],  'Is\_Response': y\_pred}  df **=** pd.DataFrame(raw\_data, columns **=** ['User\_ID', 'Is\_Response']) df.to\_csv('submission\_model1.csv', sep**=**',',index**=**False) |

*Test Results*

After evaluating the prediction file, the achieved accuracy score was 87.79.