Automatic Emoji Recommendation System

Team Members
203059011-Akshay Batheja
20305R002-Shivam Ratnakant Mhaskar

Dec 12, 2020

CONTENTS:

CHAPTER

ONE

SOURCE

1.1 Logistic Regression with BOW module

A Logistic Regression model with Bag of Words to recommend the emojis for a sentence using the sentence's sentiment.

This module defines the following classes:

- *LR* , a class that initializes the Logistic Regression Model with Bag of Words
- *LR_run* , a class that runs the initialized LR over the Emoji Dataset

1.1.1 How To Use This Module

(See the individual classes, methods and attributes for details.)

1. Import it: `import LR`
2. Create an object of `LR_run`:

```
final_model = LR.LR_run()
```

3. Finally run the `run()` method of `LR_run` class:

```
vect,tftdf, LogReg_modellist,mlb,new_emoji_matrix, emoji_set, top =final_model.
↪run()
```

4. **After running the above lines, you should see the model metrics (`Precision`, `Recall`, `F1_score`, `accuray`) for the tw**

    - Without Emoji Matrix
    - With Emoji Matrix

5. If you want to test the system then call following method:

```
final_model.recommend_em(vect,LogReg_modellist,mlb,new_emoji_matrix,emoji_set,top)
```

6. This method will ask you to input a string and will recommend you the respective emojis for the same.

1.1.2 Working of this Module

1. *LR*, This class declares all the methods used for data preprocessing, generating LR model , generating emoji_matrix that are used by *LR_run* class.
2. *LR_run* , This class does following functions:

   a) **Generates a pandas DataFrame from a text file consisting of Tweets. These Tweets consists of text and multiple emoj**

      - *extract_sent*, extracts the sentences from tweets.
      - *extract_emojis*, extracts the emojis from the tweets.

   b) It then cleans the DataFrame by removing stop_words, punctuations, performing lemmatization.

```
def remove_stopwords(self,s):
    return ' '.join(self.lemmatizer.lemmatize(c.lower()) for c in list(s.
↪split()) if c not in self.stop_words)
```

   c) It then keeps only sentences that has atleast one of the Top 100 most frequent used emojis. Look at the function *clean* and *label* for reference.

   d) After cleaning the whole dataset it the vectorize the text sentences and labels(emojis) using Bag Of Words and sklearn's Mulilabel Binarizer respectively:

```
def create_bag_of_words(self, X, y):
'''
:type X: pandas.core.frame.DataFrame
:parameter X: DataFrame consisting of text of whole corpus

:type y: pandas.core.frame.DataFrame
:parameter y: DataFrame consisting of the emojis correspong to the text␣
↪passed as argument1

returns a vectorized form of the whole dataset
'''

print ('Creating bag of words...')
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,␣
↪preprocessor = None, stop_words = None, ngram_range = (1,3), max_features =␣
↪800)

train_data_features = vectorizer.fit_transform(X)
train_data_features = train_data_features.toarray()
tfidf = TfidfTransformer()
tfidf_features = tfidf.fit_transform(train_data_features).toarray()
vocab = vectorizer.get_feature_names()

mlb = MultiLabelBinarizer()
y_train_mlb = mlb.fit_transform(y.apply(lambda x: tuple(x)))

return vectorizer, vocab, train_data_features, tfidf_features, tfidf, y_train_
↪mlb, mlb
```

   e) After vectorizing we split the whole dataset into 80%(train_data) and 20%(test_data):

```
#splitting dataset into train 80% and test 20%
X_train,X_test,y_train,y_test = train_test_split(train_sent_X,train_label_y,␣
↪test_size=0.2)
```

f) After splitting the whole dataset we generate a Pipeline with LR clf

```
def make_model(self):

'''
returns a pipeline of onevsrestclassifier with logistic regression
'''

LogReg_pipeline = Pipeline([
            ('clf', OneVsRestClassifier(LogisticRegression(solver='sag'), n_
↪jobs=-1)),
        ])

return LogReg_pipeline
```

g) Now we train the model for each emoji as class (OneVsRest classifier):

```
#Training LR for each emoji as a class
for i in range(y_train.shape[1]):
    print('**Processing class {} comments...**'.format(i))

    # Training logistic regression model on train data
    LogReg_pipeline.fit(X_train, y_train[:,i])

    # calculating test accuracy
    prediction = LogReg_pipeline.predict(X_test)

    pred[:,i] = prediction
    print(pred.shape)
    if 1 in pred[:,i]:
        print('yes')
    print('Test accuracy is {}'.format(accuracy_score(y_test[:,i],␣
↪prediction)))
print('Model Metrics without grouping / clustering')
```

h) **Now we follow two approaches for evaluating our model:**

- *Without Emoji Matrix*, directly find the model metrics from the model trained by predicting it over the test dataset.
- *With Emoji Matrix*, generate an emoji_matrix from emoji_net. Look at `generate_emoji_matrix` for reference. Then transform the test labels and predicted labels into their respective keywords using emoji_matrix and the find the model metrics. Look at `find_sem` method for reference.

i) After running all above steps you should be able to see the model's test metrics with and without emoji matrix.

class LR.**LR**(*stop_words*, *lemmatizer*)

Bases: `object`

A Logistic Regression model with Bag of Words.

It declares all the methods for data preprocessing, cnn model generation, emoji_matrix generation etc.

clean(*label*)

function to keep only 100 most frequent as target labels, removing empty labels

Parameters **label** (*set*) – set of emojis

returns set consisting of only the 100 most frequent emojis

create_bag_of_words(*X*, *y*)

Parameters

- **X** (*pandas.core.frame.DataFrame*) – DataFrame consisting of text of whole corpus
- **y** (*pandas.core.frame.DataFrame*) – DataFrame consisting of the emojis correspong to the text passed as argument1

returns a vectorized form of the whole dataset

data_preprocessing(*data*, *all_emojis*)

Generating data dict from the emoji-twitter text file , performing lemmatization and removing stop_words, punctuations and converting the sentence to lower caps

Parameters

- **data** (*dict*) – dict consisting of the complete dataset with text and emojis
- **all_emojis** (*dict*) – dict consisting of all the emojis and their frequency

returns the preprocessed dataset i.e removing stop_words, lemmatization, extracting emojis and text

extract_emojis(*s*)

Extracting emojis out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns a set of emojis in the respective sentence

extract_sent(*s*)

Extracting text sentences out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns the text and remove the emoji from the sentence

find_sem(*y*, *y_hat*, *emoji_matrix*, *emoji_set*, *top*)

converting predicted multilabels into cluters/groups using emoji_matrix

Parameters

- **y** (*numpy.ndarray*) – Actual vector of the test instance
- **emoji_matrix** (*pandas.core.frame.DataFrame*) – emoji matrix constructed from the Emojinet

Paramter y_hat Predicted vector of the test instance

returns the keywords corresponding to the y and y_hat

generate_emoji_matrix()

Generate Emoji_matrix using Emojinet , emoji_matrix consists of the emojis as its columns and keywords as its index

make_model()

returns a pipeline of onevsrestclassifier with logistic regression

new_label(*label*)

function to keep those labels with atleast one of the most frequent emojis

Parameters **label** (*set*) – set of emojis

returns the set if atleast one of the emojis in set is in emoji_set

remove_stopwords(*s*)
removing stop words and performing lemmatization

:type s : string :parameter s : tweet containing text and emoji

returns a lemmatized sentence free from stop_words

set_emo(*emo_list*, *emoji_set*)
function to set emo_list and emoji_set as its instance members

Parameters

- **emo_list** (*list*) – list of emojis
- **emoji_set** – set of emojis

Emoji_set set

vectorizer(*data_df*)
vectorizing the whole dataset - text and labels

Parameters **data_df** (*pandas.core.frame.DataFrame*) – DataFrame consisting all the training and testing dataset with text and labels

returns the vectorized sentences and labels respective to all the complete dataset

class LR.**LR_run**
Bases: object

It is a class that is responsible for the execution of all the methods declared in the LR class and training and tesing the model and finally generating the model metrics with and without the emoji matrix.

find_s(*y*, *emoji_matrix*, *emoji_set*, *top*)
It converts the predicted set of emojis into its keyword using emoji matrix

Parameters

- **y** (*numpy.ndarray*) – numpy input vector
- **emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis

returns a keyword corresponding to the label vector

recommend_em(*vect*, *LogReg_modellist*, *mlb*, *new_emoji_matrix*, *emoji_set*, *top*)
This function generates the emojis for the input sentence

Parameters

- **vect** (*sklearn.CountVectorizer*) – tokenize the given input sentence
- **LogReg_modellist** (*list*) – list of trained pipeline of Logistic Regression classifiers for different classes(emojis)
- **mlb** (*sklearn.preprocessing._label.MultiLabelBinarizer*) – To vectorize the labels
- **new_emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis

run()
This method creates an object of cnn class and perform all the required operation for generation of model metrics like data preprocessing, training , testing.

1.2 Convolution Neural Network module

A Convolution Neural Network to recommend the emojis for a sentence using the sentence's sentiment.

This module defines the following classes:

- *cnn* , a class that initializes the Convolution Neural Network
- *cnn_run* , a class that runs the initialized CNN over the Emoji Dataset

1.2.1 How To Use This Module

(See the individual classes, methods and attributes for details.)

1. Import it: `import cnn`
2. Create an object of `cnn_run`:

```
final_model = cnn.cnn_run()
```

3. Finally run the `run()` method of `cnn_run` class:

```
vect , model ,mlb, new_emoji_matrix,emoji_set,top, max_len = final_model.run()
```

4. **After running the above lines, you should see the model metrics (`Precision`, `Recall`, `F1_score`, `accuray`) for the tw**

    - Without Emoji Matrix
    - With Emoji Matrix

5. If you want to test the system then call following method:

```
final_model.recommend_em(vect,model,mlb,new_emoji_matrix,emoji_set,top, max_len)
```

6. This method will ask you to input a string and will recommend you the respective emojis for the same.

1.2.2 Working of this Module

1. *cnn*, This class declares all the methods used for data preprocessing, generating cnn model , generating emoji_matrix that are used by *cnn_run* class.
2. *cnn_run* , This class does following functions:
    a) **Generates a pandas DataFrame from a text file consisting of Tweets. These Tweets consists of text and multiple emoj**

        - *extract_sent*, extracts the sentences from tweets.
        - *extract_emojis*, extracts the emojis from the tweets.

    b) It then cleans the DataFrame by removing stop_words, punctuations, performing lemmatization.

```
def remove_stopwords(self,s):
    return ' '.join(self.lemmatizer.lemmatize(c.lower()) for c in list(s.
↪split()) if c not in self.stop_words)
```

c) It then keeps only sentences that has atleast one of the Top 100 most frequent used emojis. Look at the function *clean* and *label* for reference.

d) After cleaning the whole dataset it the vectorize the text sentences and labels(emojis) using keras vectorizer and sklearn's Mulilabel Binarizer respectively:

```
def vectorizer(self,data_df):
'''
vectorizing the whole dataset -  text and labels

:type data_df: pandas.core.frame.DataFrame
:paramter data_df: DataFrame consisting all the training and testing dataset␣
↪with text and labels

returns the vectorized sentences and labels respective to all the complete␣
↪dataset
'''

#vectorizing
vect = Tokenizer()
vect.fit_on_texts(data_df['text'])
train_sent = vect.texts_to_sequences(data_df['text'])
vocab_size = len(vect.word_index) + 1

#set max_len of sentence
max_len = 50
train_sent_X = pad_sequences(train_sent, padding='post', maxlen=max_len)
mlb = MultiLabelBinarizer()
train_label_y = mlb.fit_transform(data_df['label'].apply(lambda x: tuple(x)))

return vect,train_sent,vocab_size,train_sent_X,train_label_y,mlb,max_len
```

e) After vectorizing we split the whole dataset into 80%(train_data) and 20%(test_data):

```
#splitting dataset into train 80% and test 20%
X_train,X_test,y_train,y_test = train_test_split(train_sent_X,train_label_y,␣
↪test_size=0.2)
```

f) After splitting the whole dataset we generate a tf.keras cnn model:

```
def make_model(self,vocab_size,top,max_len):
'''
making cnn model using tf.keras

:type vocab_size: int
:parameter vocab_size: count of unique words in the corpus

:type top: int
:parameter top: count of most frequent emojis

:type max_len: int
:paramter max_len: fixed max length of the sentence
```

(continues on next page)

(continued from previous page)

```
    returns the cnn model configured using tf.keras
    '''

    model = keras.Sequential()
    model.add(tensorflow.keras.layers.Embedding(vocab_size,top, input_length=max_
    ↪len))
    model.add(tensorflow.keras.layers.Conv1D(32, 5, padding='valid', activation=
    ↪'relu'))
    model.add(tensorflow.keras.layers.Conv1D(64, 3, padding='valid', activation=
    ↪'relu'))
    model.add(tensorflow.keras.layers.Dense(128, activation='relu'))
    model.add(tensorflow.keras.layers.LSTM(32,return_sequences=True))
    model.add(tensorflow.keras.layers.Flatten())
    model.add(tensorflow.keras.layers.Dense(top, activation = 'sigmoid'))
    print(model.summary())
    return model
```

g) **Now we train the model::** #compiling and training the model model.compile(optimizer='adam', loss='binary_crossentropy', metrics = ['accuracy']) model.fit(X_train, y_train, epochs = 1, validation_data=(X_test, y_test), batch_size = 64)

h) **Now we follow two approaches for evaluating our model:**

- *Without Emoji Matrix*, directly find the model metrics from the model trained by predicting it over the test dataset.
- *With Emoji Matrix*, generate an emoji_matrix from emoji_net. Look at `generate_emoji_matrix` for reference. Then transform the test labels and predicted labels into their respective keywords using emoji_matrix and the find the model metrics. Look at `find_sem` method for reference.

i) After running all above steps you should be able to see the model's test metrics with and without emoji matrix.

class cnn.**cnn**(*stop_words*, *lemmatizer*)

Bases: object

A Convolution Neural Network.

It declares all the methods for data preprocessing, cnn model generation, emoji_matrix generation etc.

clean(*label*)

function to keep only 100 most frequent as target labels, removing empty labels

Parameters **label** (*set*) – set of emojis

returns set consisting of only the 100 most frequent emojis

data_preprocessing(*data*, *all_emojis*)

Generating data dict from the emoji-twitter text file , performing lemmatization and removing stop_words, punctuations and converting the sentence to lower caps

Parameters

- **data** (*dict*) – dict consisting of the complete dataset with text and emojis
- **all_emojis** (*dict*) – dict consisting of all the emojis and their frequency

returns the preprocessed dataset i.e removing stop_words, lemmatization, extracting emojis and text

extract_emojis(*s*)

Extracting emojis out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns a set of emojis in the respective sentence

extract_sent(*s*)
Extracting text sentences out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns the text and remove the emoji from the sentence

find_sem(*y*, *y_hat*, *emoji_matrix*, *emoji_set*, *top*)
converting predicted multilabels into cluters/groups using emoji_matrix

Parameters

- **y** (*numpy.ndarray*) – Actual vector of the test instance
- **emoji_matrix** (*pandas.core.frame.DataFrame*) – emoji matrix constructed from the Emojinet

Paramter y_hat Predicted vector of the test instance

returns the keywords corresponding to the y and y_hat

generate_emoji_matrix()
Generate Emoji_matrix using Emojinet , emoji_matrix consists of the emojis as its columns and keywords as its index

make_model(*vocab_size*, *top*, *max_len*)
making cnn model using tf.keras

Parameters

- **vocab_size** (*int*) – count of unique words in the corpus
- **top** (*int*) – count of most frequent emojis
- **max_len** (*int*) – fixed max length of the sentence

returns the cnn model configured using tf.keras

new_label(*label*)
function to keep those labels with atleast one of the most frequent emojis

Parameters **label** (*set*) – set of emojis

returns the set if atleast one of the emojis in set is in emoji_set

remove_stopwords(*s*)
removing stop words and performing lemmatization

:type s : string :parameter s : tweet containing text and emoji

returns a lemmatized sentence free from stop_words

set_emo(*emo_list*, *emoji_set*)
function to set emo_list and emoji_set as its instance members

Parameters

- **emo_list** (*list*) – list of emojis
- **emoji_set** – set of emojis

Emoji_set set

vectorizer(*data_df*)
vectorizing the whole dataset - text and labels

Parameters **data_df** (*pandas.core.frame.DataFrame*) – DataFrame consisting all the training and testing dataset with text and labels

returns the vectorized sentences and labels respective to all the complete dataset

class cnn.**cnn_run**
Bases: object

It is a class that is responsible for the execution of all the methods declared in the cnn class and training and tesing the model and finally generating the model metrics with and without the emoji matrix.

find_s(*y*, *emoji_matrix*, *emoji_set*, *top*)
It converts the predicted set of emojis into its keyword using emoji matrix

Parameters

- **y** (*numpy.ndarray*) – numpy input vector
- **emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis

returns a keyword corresponding to the label vector

recommend_em(*vect*, *model*, *mlb*, *new_emoji_matrix*, *emoji_set*, *top*, *max_len*)
This function generates the emojis for the input sentence

Parameters

- **vect** (*keras_preprocessing.text.Tokenizer*) – tokenize the given input sentence
- **model** (*tensorflow.python.keras.engine.sequential.Sequential*) – trained CNN model
- **mlb** (*sklearn.preprocessing._label.MultiLabelBinarizer*) – To vectorize the labels
- **new_emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis
- **max_len** (*int*) – maximum length of sentence after padding

run()
This method creates an object of cnn class and perform all the required operation for generation of model metrics like data preprocessing, training , testing.

1.3 CNN-Long Short Term Memory module

A Convolution Neural Network - Long Short term memory to recommend the emojis for a sentence using the sentence's sentiment.

This module defines the following classes:

- *cnn_lstm* , a class that initializes the Convolution Neural Network
- *cnn_lstm_run* , a class that runs the initialized cnn_lstm over the Emoji Dataset

1.3.1 How To Use This Module

(See the individual classes, methods and attributes for details.)

1. Import it: `import cnn_lstm`
2. Create an object of `cnn_lstm_run`:

```
final_model = cnn_lstm.cnn_lstm_run()
```

3. Finally run the `run()` method of `cnn_lstm_run` class:

```
vect , model ,mlb, new_emoji_matrix,emoji_set,top, max_len = final_model.run()
```

4. **After running the above lines, you should see the model metrics (`Precision`, `Recall`, `F1_score`, `accuray`) for the tw**

    - Without Emoji Matrix
    - With Emoji Matrix

5. If you want to test the system then call following method:

```
final_model.recommend_em(vect,model,mlb,new_emoji_matrix,emoji_set,top, max_len)
```

6. This method will ask you to input a string and will recommend you the respective emojis for the same.

1.3.2 Working of this Module

1. *cnn_lstm*, This class declares all the methods used for data preprocessing, generating cnn_lstm model , generating emoji_matrix that are used by *cnn_lstm_run* class.
2. *cnn_lstm_run* , This class does following functions:
    a) **Generates a pandas DataFrame from a text file consisting of Tweets. These Tweets consists of text and multiple emoj**

        - *extract_sent*, extracts the sentences from tweets.
        - *extract_emojis*, extracts the emojis from the tweets.

    b) It then cleans the DataFrame by removing stop_words, punctuations, performing lemmatization.

```
def remove_stopwords(self,s):
    return ' '.join(self.lemmatizer.lemmatize(c.lower()) for c in list(s.
↪split()) if c not in self.stop_words)
```

c) It then keeps only sentences that has atleast one of the Top 100 most frequent used emojis. Look at the function *clean* and *label* for reference.

d) After cleaning the whole dataset it the vectorize the text sentences and labels(emojis) using keras vectorizer and sklearn's Mulilabel Binarizer respectively:

```
def vectorizer(self,data_df):
'''
vectorizing the whole dataset -  text and labels

:type data_df: pandas.core.frame.DataFrame
:paramter data_df: DataFrame consisting all the training and testing dataset␣
↪with text and labels

returns the vectorized sentences and labels respective to all the complete␣
↪dataset
'''

#vectorizing
vect = Tokenizer()
vect.fit_on_texts(data_df['text'])
train_sent = vect.texts_to_sequences(data_df['text'])
vocab_size = len(vect.word_index) + 1

#set max_len of sentence
max_len = 50
train_sent_X = pad_sequences(train_sent, padding='post', maxlen=max_len)
mlb = MultiLabelBinarizer()
train_label_y = mlb.fit_transform(data_df['label'].apply(lambda x: tuple(x)))

return vect,train_sent,vocab_size,train_sent_X,train_label_y,mlb,max_len
```

e) After vectorizing we split the whole dataset into 80%(train_data) and 20%(test_data):

```
#splitting dataset into train 80% and test 20%
X_train,X_test,y_train,y_test = train_test_split(train_sent_X,train_label_y,␣
↪test_size=0.2)
```

f) After splitting the whole dataset we generate a tf.keras cnn_lstm model:

```
def make_model(self,vocab_size,top,max_len):
'''
making cnn_lstm model using tf.keras

:type vocab_size: int
:parameter vocab_size: count of unique words in the corpus

:type top: int
:parameter top: count of most frequent emojis

:type max_len: int
:paramter max_len: fixed max length of the sentence

returns the cnn_lstm model configured using tf.keras
'''

model = keras.Sequential()
model.add(tensorflow.keras.layers.Embedding(vocab_size,top, input_length=max_
↪len))
```

(continues on next page)

(continued from previous page)

```
model.add(tensorflow.keras.layers.Conv1D(32, 5, padding='valid', activation=
↪'relu'))
model.add(tensorflow.keras.layers.Conv1D(64, 3, padding='valid', activation=
↪'relu'))
model.add(tensorflow.keras.layers.Dense(128, activation='relu'))
model.add(tensorflow.keras.layers.Flatten())
model.add(tensorflow.keras.layers.Dense(top, activation = 'sigmoid'))
print(model.summary())
return model
```

g) **Now we train the model::** #compiling and training the model model.compile(optimizer='adam', loss='binary_crossentropy', metrics = ['accuracy']) model.fit(X_train, y_train, epochs = 1, validation_data=(X_test, y_test), batch_size = 64)

h) **Now we follow two approaches for evaluating our model:**

- *Without Emoji Matrix*, directly find the model metrics from the model trained by predicting it over the test dataset.
- *With Emoji Matrix*, generate an emoji_matrix from emoji_net. Look at `generate_emoji_matrix` for reference. Then transform the test labels and predicted labels into their respective keywords using emoji_matrix and the find the model metrics. Look at `find_sem` method for reference.

i) After running all above steps you should be able to see the model's test metrics with and without emoji matrix.

class cnnlstm.**cnn_lstm**(*stop_words*, *lemmatizer*)

Bases: object

A Convolution Neural Network.

It declares all the methods for data preprocessing, cnn_lstm model generation, emoji_matrix generation etc.

clean(*label*)

function to keep only 100 most frequent as target labels, removing empty labels

Parameters **label** (*set*) – set of emojis

returns set consisting of only the 100 most frequent emojis

data_preprocessing(*data*, *all_emojis*)

Generating data dict from the emoji-twitter text file , performing lemmatization and removing stop_words, punctuations and converting the sentence to lower caps

Parameters

- **data** (*dict*) – dict consisting of the complete dataset with text and emojis
- **all_emojis** (*dict*) – dict consisting of all the emojis and their frequency

returns the preprocessed dataset i.e removing stop_words, lemmatization, extracting emojis and text

extract_emojis(*s*)

Extracting emojis out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns a set of emojis in the respective sentence

extract_sent(*s*)

Extracting text sentences out off the tweet

Parameters **s** (*string*) – tweet containing text and emoji

returns the text and remove the emoji from the sentence

find_sem(*y*, *y_hat*, *emoji_matrix*, *emoji_set*, *top*)
converting predicted multilabels into cluters/groups using emoji_matrix

Parameters

- **y** (*numpy.ndarray*) – Actual vector of the test instance
- **emoji_matrix** (*pandas.core.frame.DataFrame*) – emoji matrix constructed from the Emojinet

Paramter y_hat Predicted vector of the test instance

returns the keywords corresponding to the y and y_hat

generate_emoji_matrix()
Generate Emoji_matrix using Emojinet , emoji_matrix consists of the emojis as its columns and keywords as its index

make_model(*vocab_size*, *top*, *max_len*)
making cnn_lstm model using tf.keras

Parameters

- **vocab_size** (*int*) – count of unique words in the corpus
- **top** (*int*) – count of most frequent emojis
- **max_len** (*int*) – fixed max length of the sentence

returns the cnn_lstm model configured using tf.keras

new_label(*label*)
function to keep those labels with atleast one of the most frequent emojis

Parameters **label** (*set*) – set of emojis

returns the set if atleast one of the emojis in set is in emoji_set

remove_stopwords(*s*)
removing stop words and performing lemmatization

:type s : string :parameter s : tweet containing text and emoji

returns a lemmatized sentence free from stop_words

set_emo(*emo_list*, *emoji_set*)
function to set emo_list and emoji_set as its instance members

Parameters

- **emo_list** (*list*) – list of emojis
- **emoji_set** – set of emojis

Emoji_set set

vectorizer(*data_df*)
vectorizing the whole dataset - text and labels

Parameters **data_df** (*pandas.core.frame.DataFrame*) – DataFrame consisting all the training and testing dataset with text and labels

returns the vectorized sentences and labels respective to all the complete dataset

class cnnlstm.**cnn_lstm_run**

Bases: object

It is a class that is responsible for the execution of all the methods declared in the cnn_lstm class and training and tesing the model and finally generating the model metrics with and without the emoji matrix.

find_s(*y*, *emoji_matrix*, *emoji_set*, *top*)

It converts the predicted set of emojis into its keyword using emoji matrix

Parameters

- **y** (*numpy.ndarray*) – numpy input vector
- **emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis

returns a keyword corresponding to the label vector

recommend_em(*vect*, *model*, *mlb*, *new_emoji_matrix*, *emoji_set*, *top*, *max_len*)

This function generates the emojis for the input sentence

Parameters

- **vect** (*keras_preprocessing.text.Tokenizer*) – tokenize the given input sentence
- **model** (*tensorflow.python.keras.engine.sequential.Sequential*) – trained CNN-LSTM model
- **mlb** (*sklearn.preprocessing._label.MultiLabelBinarizer*) – To vectorize the labels
- **new_emoji_matrix** (*pandas.frame.DataFrame*) – DataFrame consisting of keywords as its index and emojis as its columns
- **emoji_set** (*set*) – set of "top" most frequent emojis
- **top** (*int*) – "top" most frequent emojis
- **max_len** (*int*) – maximum length of sentence after padding

run()

This method creates an object of cnn_lstm class and perform all the required operation for generation of model metrics like data preprocessing, training , testing.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

L

INDEX