A Word Embedding Approach and Sentiment Analysis On Tweets for Social Emotions Domain

Chee Sai Wai
School of Computer Sciences
(of Affiliation)
Universit Sains Malaysia
(of Affiliation)
Penang, Malaysia
callmesaiwai@student.usm.my

Tan Ling Chian
School of Computer Sciences
(of Affiliation)
Universit Sains Malaysia
(of Affiliation)
Penang, Malaysia
lingchianjoyce@student.usm.my

***Abstract*— Datasets are annotated and important for the advancement of classification of emotions. The goal is to abstract the use of different label sets, including happy and sad due to its importance to the domain, and the sentiment analysis problem. These supervised models often only use a limited set of available labelled with one or more sentiment labels. In addition, no previous work has compared emotion corpora even more in a word embedding model. We aim at contributing to this situation with a survey of main dataset, and evaluate them in classification models with some tuned parameters. Based on this signals for emotion prediction accuracy, we perform the first 5-fold cross validation experiments. One result from our analysis is that happiness F-measure and overall accuracy is better classified with models trained on a different parameters. For dimension of the tweets, training averages all the words' embedding vectors together is better than maximum value of all the embedding.**

Keywords—cross validation, classification, sentiment analysis, emotion, word embedding

I. Introduction

This study investigates on dilemma for internet user over Twitter data. With English text itself in the tweets, client in social news domain requires a reliable sentiment annotation to extract emotion. This study, probes on Twitter sentiment analysis related tasks by grouping similar words together that they experienced in, and elements which motivate accurate embedding model to remain with a particular word embedding data set.

Our lecturer, Dr. Gan Keng Hoon, requests that we write either R or Python software. A much more ambitious project and our group wrote Python program to archive emotion for sentiment analysis from massive major events. Happy tweets and sad tweets by main dataset allows tweet collection given a comma separated (csv) file.

We first discussed the objective and major issues by detailing process and design specification. Further, we have discussed implementation process which covers each of the sentiment components. We then discussed how classes of sentiment score such as negative, neutral and positive which builds upon the supervised method. For the outcome that we have received from evaluation train and split sets, we accessed word embedding technique we learned in this semester. We then observed tags without every emotional states in the collection to infer what was through crowd-sourced studies. With the tweet content given, we manually ensure metadata are shortened and converted to their true regular expression. The relevant hashtags set includes user's emotion such as '#happy' and '#sad' while the events against the relevant hashtags set includes user's emotion such as neutral. These labels are important for modelling stage as any uses them to test and train the sentences.

Once we created the amount of posts for testing, Word2vector.py takes a corpus and computes large word vector size will give better performance. The basic statistics of the training data stands for metadata of the word embedding model. We have used name entities from text to find the character cases for each embedding in the vector model. Lastly, we integrated cross-validation and used some classifiers. The models is trained on the Tweet Sentiment Analysis.py data from the previous steps, and is randomly splitting the data into k-equal subsets and testing along the way and tested using a set of external parameters. Once an effective classifier was selected, we provided the classification accuracy between random forests, decision tree and support vector machines with a Python program. The final evaluation contains relevant emotional states from a tweet collection. The final product allows the client to either input a single preloaded emotion or a list of annotated emotions from another dataset.

II. Background and Related Works

A. Related Works

For textual dataset in the digital news industry, Twitter Search API has identified tweets with the keyword when tweets added text sentences. The nature of tweets can be any form of cleaned text problem because increased noisy, short and have different features. This is apparent that the big issue when plain information is about to develop a broader meaning. Tweets have informal abbreviations, different spellings for different words as well as spelling mistakes [1]. Facts are used in news and more words with general emotion semantics.

Sentiment analysis, which is a part of sentiment abstraction, relying a human annotation of sentiment label with a variety from positive and negative posts. The most essential metrics such as polarity of text units and indicators are directly dependent on vocabulary items. Sentiment analysis hardly found the benefits of classification to include the related aspect in modelling. Thus, a supporting literature provides techniques for large volume of positive and negative feelings. First, favourable dataset against the extensive research was created and each tweets was annotated [2]. Each tweets rely on finding the target of opinion expressed is the same as finally whether it conveys positive, negative and neutral sentiments. Second, the modelling is used by some research of neural network when a need to have phrases in tweet corpora learned. Without some discussions in early work, the combinations of predefined lists of words with sentiment values remain uninteresting [3]. Recent progress of neural network do not assign labels hence it leads to imprecise prediction about sentiment values.

Other related works from researchers developed word vector after sentiment analysis on Twitter data. One resilience of applying word embedding models is that it allows data set is much smaller than that for both words and phrases [4]. Syntactic contexts of words are previous study from [5] using C&W model and then word embedding models are similar handling from [5]–[7]. Besides previous one on C&W model, the authors from [6], [7] proposed training objectives using word2vec model to capture large number of syntactic relationships of words. Skip-gram model aims the use cases predicting the surrounding words in a sentence or tweet text. Average log probability is one fastest get close, now that large amount of training data condenses the information into real-valued vector of the dimension size ranged from tens to hundreds [4].

In our study, little progress has been made in main goals of current literature review. For motivation of sentiment analysis, we have explored micro-text with happy tweets and sad tweets. This work [8] focused on the self-tagged to represent the label, allowing our study to obtain a large representative sample from collected tweets. As such, Python program can choose supervised models. However, some author demonstrated machine learning techniques such as random forests classifier [8], Naïve Bayes and decision trees [9], and support vector machines [10], which are the combinations of three supervised methods. Based on general scheme of SenticNet, Chikersal et al. [11] presented Twitter sentiment system to generate polarity classification from its rules as the baseline.

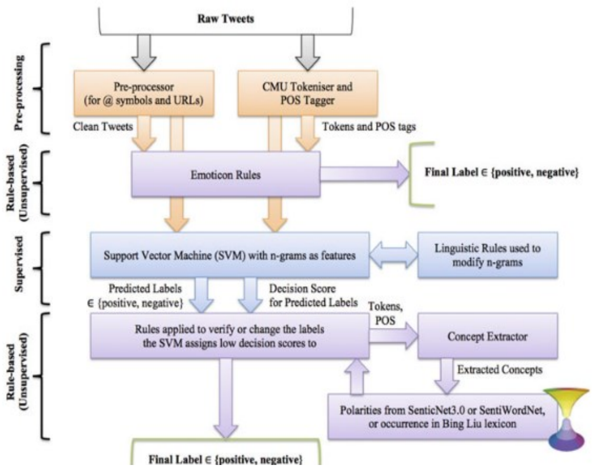
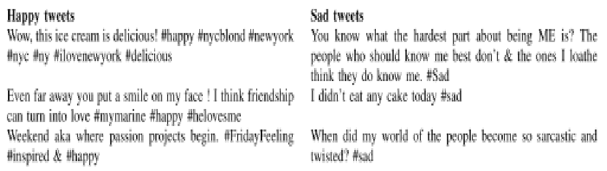

Raw Tweets
Pre-processing
Pre-processor (for @ symbols and URLs)
CMU Tokeniser and POS Tagger
Clean Tweets
Tokens and POS tags
Rule-based (Unsupervised)
Emoticon Rules
Final Label ∈ {positive, negative}
Supervised
Support Vector Machine (SVM) with n-grams as features
Linguistic Rules used to modify n-grams
Predicted Labels ∈ {positive, negative}
Decision Score for Predicted Labels
Tokens, POS
Rule-based (Unsupervised)
Rules applied to verify or change the labels the SVM assigns low decision scores to
Concept Extractor
Extracted Concepts
Polarities from SenticNet3.0 or SentiWordNet, or occurrence in Bing Liu lexicon
Final Label ∈ {positive, negative}

Fig. 1. General schema of SenticNet.



Happy tweets	Sad tweets
Wow, this ice cream is delicious! #happy #nycblond #newyork #nyc #ny #ilovenewyork #delicious	You know what the hardest part about being ME is? The people who should know me best don't & the ones I loathe think they do know me. #Sad
Even far away you put a smile on my face ! I think friendship can turn into love #mymarine #happy #helovesme	I didn't eat any cake today #sad
Weekend aka where passion projects begin. #FridayFeeling #inspired & #happy	When did my world of the people become so sarcastic and twisted? #sad

Fig. 2. Example of tagged dataset of happy and sad emotion.

B. *Problem Statement*

Despite small recommendation of emotion prediction pipeline, we will demonstrate data preprocessing techniques. To tackle cleaned text, moreover, tweets often contain URLs, elongations, repeated punctuations, emoticons, abbreviations and hashtags. Thus, it points out comparison of coding expressed within regular expressions. Each program has unique challenges of its own such as R and Python.

The challenges beyond the application of machine learning tools are considered. With a reference to supporting literature in [8], we will consider Stanford CoreNLP sentiment classifier. This system will output final label based on polarity of text units as both positive emoticon and negative emoticon and otherwise all tweets are labelled into unknown. After handling precaution, we will find false annotation outputted by the system. Thus, it points out impact and correlation to tweets above three emoticons. For example, there are threshold between strongly positive and strongly negative instances.

C. *Scope and Limitation*

The scope of our study now represents limited source of datasets. However, these datasets can be monitored as they are publicly visible because each sources have a huge volume of way people express their views. Due to tremendous sentiment values for practical applications, most of our works rely on three emotions, employing positive, negative and neutral. For word embedding model of our study, we focuses the existing approach and can be similar as literature in [4]. Again, our dataset is yet for training dataset and followed by a test set. It is possible to create a test set for word embedding model before using them. As natural language processing (NLP) tasks in different domain and different text format, our study demonstrated discovery about micro-text and created basic data preprocessing to run models with good classification report. Thus, our output is far from a way in human capability and we served guides for analysis of social data.

III. PROPOSED SOLUTION

This section has outlined several data preprocessing and distant supervision of both happy tweets and sad tweets such as in Python program. This section is organized as importing dataset called Grounded.csv. Then, the implementation of each use case is detailed following the components of the provided source codes. In tweet sentiment analysis.py file, we proposed suitable machine learning model to predict sentiment labels. Next, each 5-fold cross validation and confusion matrix of testing data varying with distributions is presented. Next section is organized as results to compare the F-measure and accuracy on the labelled dataset. Finally, the comparison of word embedding parameters are analyzed.

A. *Sentiment Analysis Framework*

Initially, our framework and few components are summarized in Figure 3. The flow of data starts from the individual data collection and moves with arrows indicating connected items. Since the csv file handles metadata, most field columns were filtered except column with the texts from txt file. To do this, we need to remove mentions and URLs and apply regular expressions to clean the data. For short, regular expressions are descriptions for a pattern of text. Regular expressions sometimes use backslashes in them. It is convenient to pass raw strings to the re.compile () function instead of adding extra backslashes. Next, we classify emotion-related hashtag such as '#happy' and '#sad' to obtain the use of hashtag. The dimension size of the dataset can be reduced from millions to thousands instances. This data will be used later to train the machine learning model to predict sentiment labels. We will be testing different classifiers such as decision trees classifier, gradient boosting tree and random forests tree classifier. Tweets often contains preceding hashtag such as '#not', therefore we will handle word vector when sentiment whether or not expresses "#happy'.

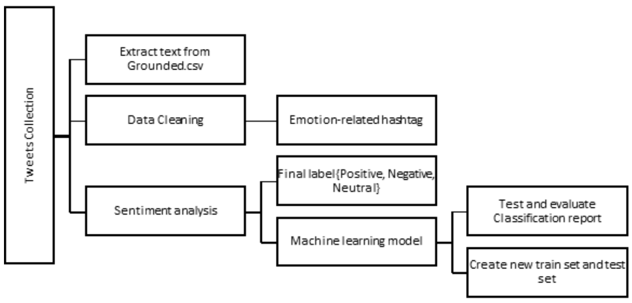

Tweets Collection
Extract text from Grounded.csv
Data Cleaning
Emotion-related hashtag
Sentiment analysis
Final label{Positive, Negative, Neutral}
Machine learning model
Test and evaluate Classification report
Create new train set and test set

Fig. 3. Customized framework for sentiment analysis on tweets for social emotions domain.

B. *Word Embedding Framework*

After we loaded the EMOJISET dataset, we have to represent split as 70/30 as distribution and create for each as a train set and a test set. We will use input of the model is a string corresponding to a sentence. The output will be the probability vector of shape. After first step is done, we will convert each word in the input sentence into their word vector representations. We have then take an average of the word vectors. This will fed into softmax() function in Python to train the parameters as shown in Equation 1. We will be testing word ordering, evaluating and analyzing test performance using confusion matrix whose label is one class is mislabeled by the algorithm with different class.

$$\sigma(We + \mathrm{b}) = \gamma \qquad (1)$$

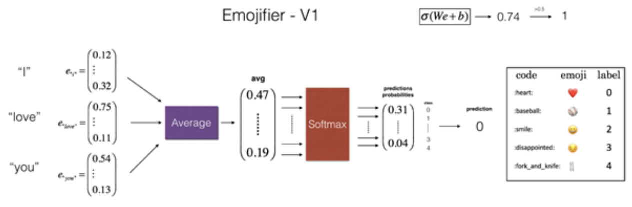


Fig. 4. Customized framework of training word embeddings.

IV. IMPLEMENTATION

All components created as part of our project are separate scripts written in Python 3. There is a detailed description for each file.

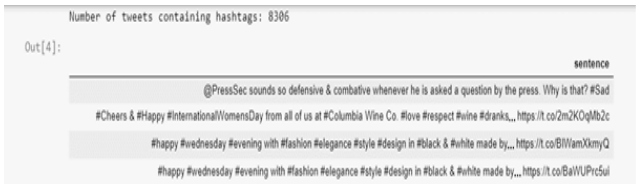
A. *Tweet Sentiment Analysis.py*

Tweet collections come in between January 18, 2017 and April 14, 2017 via the download at http://lit.eecs.umich.edu/downloads.html [8]. When working on the code with 2,557 single labelled examples, we encountered emotion tagged tweets contained within both TXT and CSV files. As such, we decided to make our remaining content, as well as metadata, such as extracting the time it was published, its author, and its location from both formats. Shown below in table 1, is a summary of the total number of labeled instances collected and attributes dropped. The set of labels is happy and sad. The tweets are annotated by the authors [8].

Total number of tweets	Total number of tweets with hashtags	Final total number of happy tweets	Final total number of sad tweets
22,890	8306	726	267

Fig. 5. Example of number of instances from Grounded-Emotion dataset.

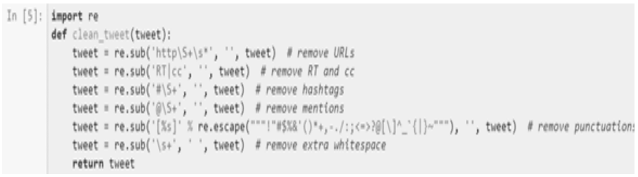
We started with a final set of about 726 happy tweets and about 267 sad tweets had mentions and URLs. After all the mentions, URLs with punctuations have been removed, we were left with about 22,000 labeled instances. Lastly, we generated a total of about 8,000 emotion-related hashtags. For hashtags, the first problem that we had to solve was how to extract all the character cases '#' from a collection of tweets. Figure 6 shows a sample of what a raw data frame of tweet collections looked like. The panda library in Python also ignored user ID, date and time as other columns, except sentences as only column.



```
Number of tweets containing hashtags: 8306

Out[4]:
                                                                                     sentence
                      @PressSec sounds so defensive & combative whenever he is asked a question by the press. Why is that? #Sad
    #Cheers & #Happy #InternationalWomensDay from all of us at #Columbia Wine Co. #love #respect #wine #dranks,... https://t.co/2m2KOqMb2c
        #happy #wednesday #evening with #fashion #elegance #style #design in #black & #white made by... https://t.co/BIWamXkmyQ
        #happy #wednesday #evening with #fashion #elegance #style #design in #black & #white made by... https://t.co/BaWUPrc5ui
```

Fig. 6. Source code output for tweets containing hashtags.

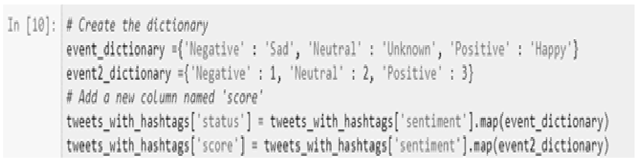
As we can see, every tweet submitted each words with a list of inferred emotions. We had to write a Python script to parse through the CSV file and extract only '#happy' and '#sad' from each tweet. A regular expression pattern had to look up the format of a hashtag and then generated a new column as sentiment score. The "tqdm" library in Python also helped to clean corpus after regular expression.



```
In [5]: import re
        def clean_tweet(tweet):
            tweet = re.sub('http\S+\s*', '', tweet)  # remove URLs
            tweet = re.sub('RT|cc', '', tweet)  # remove RT and cc
            tweet = re.sub('#\S+', '', tweet)  # remove hashtags
            tweet = re.sub('@\S+', '', tweet)  # remove mentions
            tweet = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), '', tweet)  # remove punctuation
            tweet = re.sub('\s+', ' ', tweet)  # remove extra whitespace
            return tweet
```

Fig. 7. Source code output for tweets containing cleaned corpus.

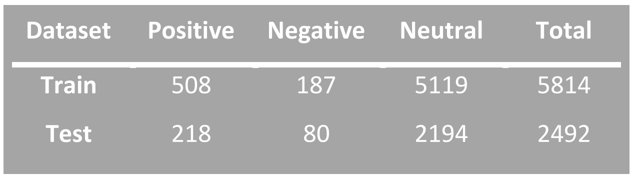
The process of regular expression was much more straightforward. The main problem with hashtag exported from the CSV file is that token with hashtag is separate object. Keeping tokens that is not useful will increase the vector space size and compute cost [4]. Then, the size of labelled corpus in target is assumed to be much smaller than training set. Therefore, we manually assigned sentiment ordering. For example, while negative was ‘1’, neutral was ‘2’ and positive was ‘3’. We also manually assigned status (emotional status) ordering. For example, while negative was ‘sad’, neutral was ‘unknown’ and positive was ‘happy’.



```
In [10]: # Create the dictionary
         event_dictionary ={'Negative' : 'Sad', 'Neutral' : 'Unknown', 'Positive' : 'Happy'}
         event2_dictionary ={'Negative' : 1, 'Neutral' : 2, 'Positive' : 3}
         # Add a new column named 'score'
         tweets_with_hashtags['status'] = tweets_with_hashtags['sentiment'].map(event_dictionary)
         tweets_with_hashtags['score'] = tweets_with_hashtags['sentiment'].map(event2_dictionary)
```

Fig. 8. Source code output for sentiments containing postive and negative.

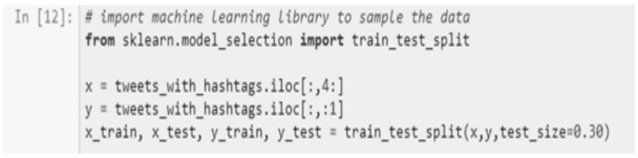
Once the sentiment label of tweets was computed, each statistic had to be summed and plotted using the “matplotlib” library in Python. Shown below in Figure 9, using the cumulative frequency table, sentiment analysis can be benefited greatly from distributions of the downloaded tweets.



Dataset	Positive	Negative	Neutral	Total
Train	508	187	5119	5814
Test	218	80	2194	2492

Fig. 9. Example of tweets sentiment statistics.

The next part of our pipeline was to model sentiment analysis as classification problem. We wanted to implement and train some classifiers. We first had to feed the extracted text corpora into different supervised learning classifiers. This was model that represents the prediction of each label as one among different labels. From the model learned in the previous decision tree classifier, we use the scikit learn class called “model selection” to train surrounding polarities of tweets into a labelled dataset. The development set we shall look for model development, and fine-tuning model parameters. For tweet sentiment analysis, we shall evaluate precision, recall, F measure and accuracy. We had to generate results later for two polarities: positive and negative and then they would yield a result of neutral.

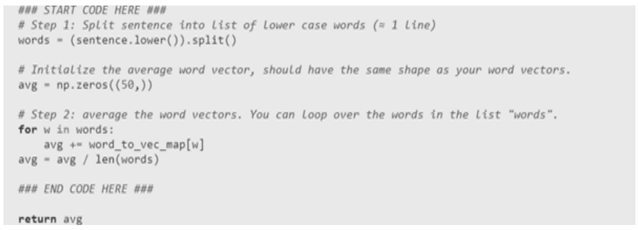


```
In [12]: # import machine learning library to sample the data
         from sklearn.model_selection import train_test_split

         x = tweets_with_hashtags.iloc[:,4:]
         y = tweets_with_hashtags.iloc[:,:1]
         x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.30)
```

Fig. 10. Source code output for splitting of train data and test data.

B. *Word2vector.py*

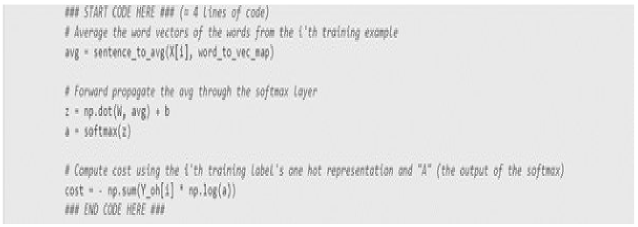
We first had to get our labels and consolidated all the steps as described in proposed solution about word embedding model. This one hot representation can take each row is one-hot vector. Once we have the index of the most likely emoji output, we can feed all the data into the word embedding model. As shown in Figure 11, the first step is to convert an input sentence into the word vector representation, which then get averaged together. Similar to the previous setup, we will use pretrained 50-dimensional GloVe embeddings. We had to convert every sentence to lower-case, then split the sentence into a list of words. Once we have each word in the sentence, this GloVe representation can average all these values.



```
### START CODE HERE ###
# Step 1: Split sentence into list of lower case words (≈ 1 line)
words = (sentence.lower()).split()

# Initialize the average word vector, should have the same shape as your word vectors.
avg = np.zeros((50,))

# Step 2: average the word vectors. You can loop over the words in the list "words".
for w in words:
    avg += word_to_vec_map[w]
avg = avg / len(words)

### END CODE HERE ###

return avg
```

Fig. 11. Source code output for word embedding modelling.

As shown in Figure 12, we had to pass the average through forward propagation, compute the cost, and then backpropagate to update the softmax's parameters. It is possible to come up with a more efficient vectorized implementation. To train a model, stochastic gradient descent via backpropagation can be extended in previous work [12]. The algorithm has generalized well to minimize the cross-entropy loss, which is a loss function for softmax output. After gradients of the loss function with respect to weights was done, parameter updates have a high variance [13].



```
### START CODE HERE ### (≈ 4 lines of code)
# Average the word vectors of the words from the i'th training example
avg = sentence_to_avg(X[i], word_to_vec_map)

# Forward propagate the avg through the softmax layer
z = np.dot(W, avg) + b
a = softmax(z)

# Compute cost using the i'th training label's one hot representation and "A" (the output of the softmax)
cost = - np.sum(Y_oh[i] * np.log(a))
### END CODE HERE ###
```

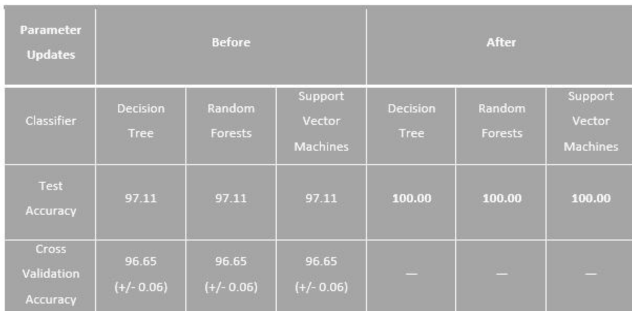
Fig. 12. Source code output for softmax parameter tuning.

Next, we had to decide on which 127 examples would be most accurate on the training set in situation. Therefore, we tested sentence “not feeling happy” which it did not appear in the training set. After testing phrases “not happy” on our labelled data set, we found that this algorithm ignored word ordering and yielded pretty good accuracy when it came to predicting a word it have not seen before. More extensive confusion matrix and performance gains are shown in next section.

V. RESULTS

In this experiment, this task performed differently which there were labelled 726 happy tweets and 267 sad tweets, was comparable with the labelled dataset was from [8]. We found that the author was using another program to create the output of tweets sentiment statistics. For tweets pre-processing approach, our experiment was based on steps we thought were necessary, since hashtags symbol provided important context and most of the token normalized. Stop words and stemming were not applied, since we were not wanting to build word cloud from our study. For our testing purposes, from our 8,306 labelled sentiments, about 5,814 scores were used for training and 2,492 scores were used for testing. A summary of the test accuracy results for each classifier is displayed in table 2 below.

As we can see, for a single test with no cross-validation, parameter updates before different classifiers yielded the 97.11% accuracy. After we thoroughly tested max_depth for both decision tree classifier and random forests classifier, parameters would increase our accuracy even more. Therefore, we also tested these parameters, if there was any influence using kernel such as linear and rbf for support vector machines classifier. Once we printed out accuracy for a single test. We chose the score with the highest accuracy, shown in Figure 13, and set the 5-fold cross validation of our classifier were used for 96.65% accuracy for accuracy accordingly.



Parameter Updates	Before			After		
Classifier	Decision Tree	Random Forests	Support Vector Machines	Decision Tree	Random Forests	Support Vector Machines
Test Accuracy	97.11	97.11	97.11	**100.00**	**100.00**	**100.00**
Cross Validation Accuracy	96.65 (+/- 0.06)	96.65 (+/- 0.06)	96.65 (+/- 0.06)	—	—	—

Fig. 13. Prediction output with example of parameter updates from three supervised classification algorithms.

Learning from the best three classifiers allowed our various experiment to predict emotions. As we can see, the precision by one point (from 93% to 94%) in an overall lower accuracy whereas F-measure by the same amount. However, we pointed out the better performance at an individual level. For example, mapping feature types using our confusion matrix, we was able to achieve all negative predictions to be sad and all positive predictions to be happy. Since neutral score in a given emotion class, we tested macro average F-measure of 56% and an overall accuracy of 97.0%, was among the lowest on this dataset. While the sentiment classifier was better tuned, the emotion of a tweet can be predicted with a 100.0% accuracy from overall accuracy.

Method	Precision	Recall	F-measure	Accuracy
Decision Tree	94.0	97.0	95.0	97.0
Random Forests	93.0	97.0	95.0	97.0
Support Vector Machines	93.0	97.0	95.0	97.0

Fig. 14. F-measre, precision, recall and overall accuracy obtained.

While in the previous outcomes in literature [8], the models estimated do not perform well on other sets. Results obtained via 10-fold cross validation in precision, recall, and F1 micro-averaged of the sentiment rule-based approach only achieved F-measure for happy of 20.5%, F-measure for sad of 53.3%, and an overall accuracy of 47.2%. After sentiment annotator was done, predicted score were used in earlier work to achieve a 64.8% using random forests classifier. The performance of random forests was significantly better. Comparing this performance trained on the sentiment score derived from the labelled tweets, MaxEnt classfier with bag-of-words (BOW) features was able to outperform it across all metrics. Training on them and classifying Grounded-Emotion datasets lead to F-measure for happy (from 20.5% to 57%) and F-measure for sad (53.3% to 44%) given outcomes in [2]. The reason was that previous methods displaying dissimilar classes in the training data. We could observe emotions than the happy and sad occurring frequently and have comparably less noisy on tweets.

Our decision tree classifier and random forests classifier as long as could be helpful to choosing best parameters. While Grounded-Emotion dataset has different labels and its max_depth was designed for different estimators, visualization of classifiers trained on their approach can be seen in Figure 15. Upon further lending evidence, the accuracy obtained by both classifier was relatively higher for max-depth of 5.

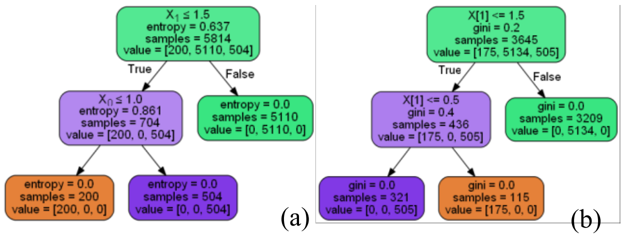
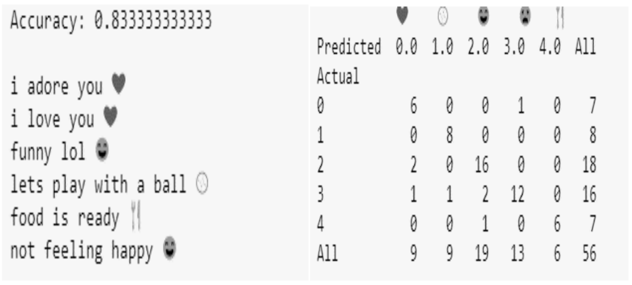


Fig. 15. Parameters output by (a) Decision Trees and (b) Random Forests.

In our analysis for emoji prediction, we explored classification problem when there were total of single labelled instances with 127 examples for word vector representations. In contrast, we tried common methods that use the maximum (max), minimum (min), or average (ave) of the embeddings of all words in a tweet [14], [15]. We were not going to compare our approach to other methods with F-measure, precision and recall derived from algorithm since that is two-way classification task.

We also performed and printed the confusion matrix to check how good performance after training on only 127 examples. A confusion matrix found whether each class determined to the training examples are often whose actual class mislabeled by the algorithm with different class was predicted class. For word embedding model trained on this dataset, our performance was slightly better and an overall accuracy of 83.0%. Similarly in [4], tweet sentiment analysis performance presented outcome which ave better than max approach. This classification algorithm lead to 84.1% accuracy for ave approach and 80.4% accuracy for max approach. We chose the algorithm with poorly on sentences such as "This movie is not good and not feeling happy" because it did not understand combinations of words. As such, we obtained averages all the words' embedding vectors together, without paying attention to the ordering of words.



```
Accuracy: 0.833333333333

i adore you ♥
i love you ♥
funny lol 😀
lets play with a ball ⚽
food is ready 🍴
not feeling happy 😀
```

```
             ♥    ⚽   😀   😞   🍴
Predicted  0.0  1.0  2.0  3.0  4.0  All
Actual
0            6    0    0    1    0    7
1            0    8    0    0    0    8
2            2    0   16    0    0   18
3            1    1    2   12    0   16
4            0    0    1    0    6    7
All          9    9   19   13    6   56
```

Fig. 16. Prediction output and confusion matrix by embedding model.

VI. Conclusion and Future Work

Amount of data play a role in sentiment classification task. Yet, we will open for diverse collection of datasets to support research on labelled data. With this paper, we present amount of work such as tweets preprocessing steps and transfer the needed data such as emotion-related hashtags. One could learn how to select the suitable classification models for a given domain and evaluate them across different labels, different F-measure, precision and recall and annotation procedures. In addition to this related work, sentiment analysis opens up for Standford CoreNLP classifier to perform annotation tasks. However, other experiments with more than three emotion prediction are added in future research. In addition, this work we provide results neutral score in a given emotion class, we tested macro average F-measure of 56% and an overall accuracy of 97.0%, was among the lowest on this dataset. While the sentiment classifier was better tuned, the emotion of a tweet can be predicted with a 100.0% accuracy from overall accuracy. While max_depth of 5 for both decision trees and random forests classifier was better than max_depth of 1, linear and RBF for support vector machines classifier were better performance. Parameter updates before different classifiers yielded the 97.11% accuracy. 5-fold cross validation of our classifier were used for 96.65% accuracy accordingly. For word embedding model trained on this dataset, our performance was slightly better and an overall accuracy of 83.0%. As future work, we plan to use another datasets, domains and annotation procedures. Furthermore, we also recommended to perform different deep learning experiment between words and phrases to analyze word embedding such as Long Short Term Memory (LSTM) model since it does usually not pay attention to the ordering of words.

Acknowledgment

The main author wishes to thank Dr. Gan Keng Hoon from School of Computer Sciences whose help was essential. He thanks supporting author to making poster presentation.

References

[1] A. Joshi, P. Bhattacharyya, and S. Ahire, *Sentiment Resources: Lexicons and Datasets*. 2017.

[2] L. A. M. Bostan and R. Klinger, "An Analysis of Annotated Corpora for Emotion Classification in Text Title and Abstract in German," *Proc. 27th Int. Conf. Comput. Linguist.*, pp. 2104–2119, 2018.

[3] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques," *arXiv Prepr. cs/0205070*, 2002.

[4] Q. Li, S. Shah, X. Liu, and A. Nourbakhsh, "Data Sets : Word Embeddings Learned from Tweets and General Data," pp. 428–436.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv Prepr. arXiv1301.3781*, 2013.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Adv. Neural Inf. Process. Syst.*, vol. 26, pp. 3111–3119, 2013.

[8] V. Liu, C. Banea, and R. Mihalcea, "Grounded emotions," *2017 7th Int. Conf. Affect. Comput. Intell. Interact. ACII 2017*, vol. 2018-Janua, pp. 477–483, 2017.

[9] R. Quinlan, "C4. 5: Programs for Machine Learning. Morgan Kaufmann Publishers," *San Mateo, CA*, 1993.

[10] J. C. Platt, "Advances in Kernel Methods of Support Vector Machines: Fast training of support vector machines using sequential minimal optimization." MIT Press, Cambridge, 1998.

[11] P. Chikersal, S. Poria, E. Cambria, A. Gelbukh, and C. E. Siong, "Modelling public sentiment in Twitter: using linguistic patterns to enhance supervised learning," in *International Conference on Intelligent Text Processing and Computational Linguistics*, 2015, pp. 49–65.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "others. 1988," *Learn. Represent. by back-propagating errors. Cogn. Model.*, vol. 5, no. 3, p. 1, 1988.

[13] L. Zhang and L. Corporation, "Deep Learning for Sentiment Analysis : A Survey."

[14] R. Socher *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[15] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, "Learning sentiment-specific word embedding for twitter sentiment classification," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 1555–1565.

Appendices

A1. Source code of Tweet Sentiment Analysis.py

```
#!/usr/bin/env python
# coding: utf-8

# #### 1 Load library package and import Grounded Emotion Dataset
# * In this project "A Word Embedding Approach and Sentiment Analysis on Social News Em
otion Data", we can download the dataset online [here](http://web.eecs.umich.edu/~mihal
cea/downloads.html#GroundedEmotions) <sup>1</sup>.
#
# #### Grounded Emotion Data Description
# * A dataset consisting of external factors associated with emotions expressed in twee
ts, including weather, news events, social network, user predisposition, and timing, us
ed in experiments aiming to show the role played by these factors in predicting emotion
s.
# > References <br>[1] <i> Grounded emotion dataset </i>, University of Michigan, Dec.
2020. [Online]. Available: http://web.eecs.umich.edu/~mihalcea/downloads.html#GroundedE
motions

# In[2]:


import pandas as pd
from IPython.display import display
tweets = pd.read_csv('data/grounded.csv', header=None, encoding='latin1')
tweets.head(5)


# In[2]:


# adding column name to the respective columns
tweets.columns =['id1', 'id2', 'sentence', 'created_at', 'tf1', 'fl1', 'fl2', 'tf2']

# displaying the DataFrame
display(tweets.head(5))

# Remove column name except 'sentence'
tweets2 = tweets.drop(['id1', 'id2', 'created_at', 'tf1', 'fl1', 'fl2', 'tf2'],
                       axis = 1)
# show the dataframe
display(tweets2.head(5))


# In[3]:


# view the shape of the data (the number of rows and columns)
print(f"The shape of the data is: {tweets2.shape}")

# view the data with the "tweet" column widened to 800px
# so that the full tweet is displayed,
# and hide the index column
tweets2.style.set_properties(subset=['sentence'], **{'width': '800px'}).hide_index()


# #### 2 Preprocess emotion-related hashtag
# * Extract each tweets contain a hashtag.
# * Clean tweets corpora using tqdm package in Python instead using regular expression.

#     * Remove URLs
#     * Remove RT and cc
#     * Remove mentions
#     * Remove punctuations
# * Reduce dimension size by selecting only instances with hashtags.

# In[4]:

```

```
# let's find out how many tweets contain a hashtag
tweets_with_hashtags = tweets2.loc[tweets2["sentence"].str.contains("#")]

# view the number of tweets that contain a hashtag
print(f"Number of tweets containing hashtags: {len(tweets_with_hashtags)}")

# view the tweets that contain a hashtag
tweets_with_hashtags.style.set_properties(subset=['sentence'], **{'width': '800px'}).hi
de_index()


# In[5]:


import re
def clean_tweet(tweet):
    tweet = re.sub('http\S+\s*', '', tweet)  # remove URLs
    tweet = re.sub('RT|cc', '', tweet)  # remove RT and cc
    tweet = re.sub('#\S+', '', tweet)  # remove hashtags
    tweet = re.sub('@\S+', '', tweet)  # remove mentions
    tweet = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), '', twee
t)  # remove punctuations
    tweet = re.sub('\s+', ' ', tweet)  # remove extra whitespace
    return tweet


# In[6]:


tweets_with_hashtags = tweets_with_hashtags.reset_index().drop(['index'], axis = 1)
print(tweets_with_hashtags)


# #### 3 Implement Sentiment score and labelled dataset
# * Manually assigned sentiment ordering. For example, while negative was ‘1’, neutral
was ‘2’ and positive was ‘3’.
# * Manually assigned status (emotional status) ordering. For example, while negative w
as ‘sad’, neutral was ‘unknown’ and positive was ‘happy’.
# * Once the sentiment label of tweets was computed, each statistic had to be summed.


# In[7]:


import numpy as np
tweets_with_hashtags['sentiment'] = np.NaN
for i in range(len(tweets_with_hashtags)):
    if (str.lower(tweets_with_hashtags['sentence'][i])).find('#happy') != -1:
        tweets_with_hashtags['sentiment'][i] = 'Positive'
    elif (str.lower(tweets_with_hashtags['sentence'][i])).find('#sad') != -1:
        tweets_with_hashtags['sentiment'][i] = 'Negative'
    else:
        tweets_with_hashtags['sentiment'][i] = 'Neutral'


# In[8]:


import seaborn as sns
sns.set(font_scale=1.4)
import matplotlib.pyplot as plt
tweets_with_hashtags['sentiment'].value_counts().plot(kind='barh', facecolor='skyblue',
figsize=(8,5)).invert_yaxis() # Plot Histogram
tweets_with_hashtags['sentiment'].value_counts() # Count rows of each sentiments
```

```
# In[9]:


from tqdm import tqdm # Clean corpus as regexes do not work

for i in tqdm(range(len(tweets_with_hashtags))):
    tweets_with_hashtags['sentence'][i] = clean_tweet(tweets_with_hashtags['sentence'][
i])


# In[10]:


# Create the dictionary
event_dictionary ={'Negative' : 'Sad', 'Neutral' : 'Unknown', 'Positive' : 'Happy'}
event2_dictionary ={'Negative' : 1, 'Neutral' : 2, 'Positive' : 3}
# Add a new column named 'score'
tweets_with_hashtags['status'] = tweets_with_hashtags['sentiment'].map(event_dictionary
)
tweets_with_hashtags['score'] = tweets_with_hashtags['sentiment'].map(event2_dictionary
)
# Move 'score' to first column
col_name = 'score'
first_col = tweets_with_hashtags.pop(col_name)
tweets_with_hashtags.insert(0, col_name, first_col)
# Print the DataFrame
print(tweets_with_hashtags)


# In[11]:


# converting type of columns to 'category'
tweets_with_hashtags['sentiment'] = tweets_with_hashtags['sentiment'].astype('category'
)
tweets_with_hashtags['status'] = tweets_with_hashtags['status'].astype('category')
# Assigning numerical values and storing in another column
tweets_with_hashtags['sentiment_Cat'] = tweets_with_hashtags['sentiment'].cat.codes
tweets_with_hashtags['status_Cat'] = tweets_with_hashtags['status'].cat.codes
# Print the DataFrame
print(tweets_with_hashtags)


# #### 4 Modeling
# * Train set (sentiment,e.g negative/neutral/positive and status, e.g. sad/unknown/hap
py)
# * Test set (score, e.g 1/2/3)
# * Supervised learning methods: <i>Decision Tree</i> <sup>2</sup>, <i>Random Forest</i
> <sup>3</sup>, <i>Support Vector Machine</i> <sup>4</sup>
#     * Decision Tree/Random Forest Parameters: `Max depth=1`,`Max depth=5`
#     * Support Vector Machine Parameters: `kernel='linear'`, `kernel='rbf'`, `kernel='
sigmoid'`
#
# > References <br> [2] R. Quinlan, "C4. 5: Programs for Machine Learning. Morgan Kaufm
ann Publishers," San Mateo, CA, 1993. <br> [3] V. Liu, C. Banea, and R. Mihalcea, "Grou
nded emotions," 2017 7th Int. Conf. Affect. Comput. Intell. Interact. ACII 2017, vol. 2
018-Janua, pp. 477-
483, 2017.<br> [4] J. C. Platt, "Advances in Kernel Methods of Support Vector Machines:
Fast training of support vector machines using sequential minimal optimization." MIT Pr
ess, Cambridge, 1998.

# In[12]:


# import machine learning library to sample the data
```

```python
from sklearn.model_selection import train_test_split

x = tweets_with_hashtags.iloc[:,4:]
y = tweets_with_hashtags.iloc[:,:1]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.30)

# import machine learning library to standardize the data
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit_transform(x_train,y_train)
sc.fit(x_test,y_test)

print('-------- x axis test ----------')
print(x_test)
print('-------- x axis train ---------')
print(x_train)
print('-------- y axis test ----------')
print(y_test)
print('-------- y axis train ---------')
print(y_train)
print('*******************************')


# In[13]:


from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

# input the decision tree classifier using "entropy" & train the model
dtree = DecisionTreeClassifier(criterion='entropy', max_depth=1).fit(x_train, y_train)

# predict the classes of new, unseen data
predict = dtree.predict(x_test)
print("The prediction accuracy is: {0:2.2f}{1:s}".format(dtree.score(x_test,y_test)*100
,"%"))
# Creates a confusion matrix
print(confusion_matrix(y_test, predict))
print(classification_report(y_test, predict))


# In[14]:


from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,
filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())


# In[15]:


from sklearn.ensemble import RandomForestClassifier
num_classifiers = 500

# input the random forest classifier using "gini" & train the model
```

```
rf = RandomForestClassifier(n_estimators=num_classifiers,
criterion='gini', max_depth=1)
rf.fit(x_train, y_train)
# predict the classes of new, unseen data
y_pred_rf = rf.predict(x_test)
print("The prediction accuracy is: {0:2.2f}{1:s}".format(rf.score(x_test,y_test)*100,"%
"))
# Creates a confusion matrix
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))


# In[16]:


from sklearn.tree import export_graphviz
import pydot
# Extract single tree
estimator = rf.estimators_[9]

# Export as dot file
dot_data = StringIO()
export_graphviz(estimator, out_file=dot_data,
rounded = True, proportion = False,
precision = 1, filled = True)

# Use dot file to create a graph
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

# Display in jupyter notebook
from IPython.display import Image
Image(graph.create_png())


# In[17]:


from sklearn.svm import SVC # "Support vector classifier"

# input the support vector classifier using "kernel" & train the model
SvmLinear = SVC(kernel='linear')
SvmRbf = SVC(kernel='rbf')
SvmSigmoid = SVC(kernel='sigmoid')
SvmLinear.fit(x_train, y_train)
SvmRbf.fit(x_train, y_train)
SvmSigmoid.fit(x_train, y_train)

# predict the classes of new, unseen data
y_predictSvm=SvmSigmoid.predict(x_test)
print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmLinear.score(x_test,y_test)
*100,"%"))
print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmRbf.score(x_test,y_test)*10
0,"%"))
print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmSigmoid.score(x_test,y_test
)*100,"%"))

# Creates a confusion matrix
print(confusion_matrix(y_test, y_predictSvm))
print(classification_report(y_test, y_predictSvm))


# #### 5 Testing and Validation

# In[18]:

```

```
# get score for the 5 fold cross validation
scoredtree = cross_val_score(dtree, x_train, y_train, cv=5, scoring='accuracy')
scorerf = cross_val_score(rf, x_train, y_train, cv=5, scoring='accuracy')
scoresvm = cross_val_score(SvmSigmoid, x_train, y_train, cv=5, scoring='accuracy')
print("The 5-fold cross-
validation score for Decision Tree Classifier is: {0:2.2f}{1:s}".format(scoredtree.mean
()*100,"%"))
print("The 5-fold cross-
validation score for Random Forest is : {0:2.2f}{1:s}".format(scorerf.mean()*100,"%"))

print("The 5-fold cross-
validation score for Support Vector Machine is : {0:2.2f}{1:s}".format(scoresvm.mean()*
100,"%"))
```

A2. Source code of Word2vector.py

```
#!/usr/bin/env python
# coding: utf-8

# #### The original source code was written in Coursera program.
#
# * First of all, none of all the codes were given to Coursera paticipant. There is som
e indicators like `##START CODE HERE##` and `##END CODE HERE##` which requires student
to complete the assignment called "Deep Learning Specialization Courses". This course c
an be found [here](https://www.coursera.org/specializations/deep-learning).
#
# * Please acknowledge contribution of P-
COM0145/19 who has used the source code with its originality.
#
# * Let's get started! Run the following cell to load the package you are going to use.


# In[1]:


import numpy as np
from emo_utils import *
import emoji
import matplotlib.pyplot as plt

get_ipython().run_line_magic('matplotlib', 'inline')


# ## 1 - Baseline word embedding model: Emojifier-V1
#
# ### 1.1 - Dataset EMOJISET
#
# Let's start by building a simple baseline classifier.
#
# You have a tiny dataset (X, Y) where:
# - X contains 127 sentences (strings).
# - Y contains an integer label between 0 and 4 corresponding to an emoji for each sent
ence.
#
# Let's load the dataset using the code below. We split the dataset between training (1
27 examples) and testing (56 examples).

# In[2]:


X_train, Y_train = read_csv('data/train_emoji.csv')
X_test, Y_test = read_csv('data/tesss.csv')


# In[3]:


maxLen = len(max(X_train, key=len).split())


# Run the following cell to print sentences from X_train and corresponding labels from
Y_train.
# * Change `idx` to see different examples.
# * Note that due to the font used by iPython notebook, the heart emoji may be colored
black rather than red.

# In[4]:


for idx in range(10):
    print(X_train[idx], label_to_emoji(Y_train[idx]))
```

```
# ### 1.2 - Overview of the Emojifier-V1
#
# In this part, you are going to implement a baseline model called "Emojifier-v1".
#
#
# #### Inputs and outputs
# * The input of the model is a string corresponding to a sentence (e.g. "I love you).

# * The output will be a probability vector of shape (1,5), (there are 5 emojis to choo
se from).
# * The (1,5) probability vector is passed to an argmax layer, which extracts the index
of the emoji with the highest probability.

# #### One-hot encoding
# * To get our labels into a format suitable for training a softmax classifier, lets co
nvert $Y$ from its current shape  $(m, 1)$ into a "one-hot representation" $(m, 5)$,
#     * Each row is a one-hot vector giving the label of one example.
#     * Here, `Y_oh` stands for "Y-one-
hot" in the variable names `Y_oh_train` and `Y_oh_test`:

# In[5]:


Y_oh_train = convert_to_one_hot(Y_train, C = 5)
Y_oh_test = convert_to_one_hot(Y_test, C = 5)


# Let's see what `convert_to_one_hot()` did. Feel free to change `index` to print out d
ifferent values.

# In[6]:


idx = 50
print(f"Sentence '{X_train[50]}' has label index {Y_train[idx]}, which is emoji {label_
to_emoji(Y_train[idx])}", )
print(f"Label index {Y_train[idx]} in one-hot encoding format is {Y_oh_train[idx]}")


# All the data is now ready to be fed into the Emojify-
V1 model. Let's implement the model!

# ### 1.3 - Implementing Emojifier-V1
#
# As shown in Figure 2 (above), the first step is to:
# * Convert each word in the input sentence into their word vector representations.
# * Then take an average of the word vectors.
# * Similar to the previous exercise, we will use pre-trained 50-
dimensional GloVe embeddings.
#
# Run the following cell to load the `word_to_vec_map`, which contains all the vector r
epresentations.

# In[7]:


word_to_index, index_to_word, word_to_vec_map = read_glove_vecs('../../readonly/glove.6
B.50d.txt')


# You've loaded:
# - `word_to_index`: dictionary mapping from words to their indices in the vocabulary

#     - (400,001 words, with the valid indices ranging from 0 to 400,000)
# - `index_to_word`: dictionary mapping from indices to their corresponding words in th
e vocabulary
# - `word_to_vec_map`: dictionary mapping words to their GloVe vector representation.
#
# Run the following cell to check if it works.

# In[8]:


word = "cucumber"
idx = 289846
print("the index of", word, "in the vocabulary is", word_to_index[word])
print("the", str(idx) + "th word in the vocabulary is", index_to_word[idx])


# **Exercise**: Implement `sentence_to_avg()`. You will need to carry out two steps:
# 1. Convert every sentence to lower-
case, then split the sentence into a list of words.
#     * `X.lower()` and `X.split()` might be useful.
# 2. For each word in the sentence, access its GloVe representation.
#     * Then take the average of all of these word vectors.
#     * You might use `numpy.zeros()`.
#
#
# #### Additional Hints
# * When creating the `avg` array of zeros, you'll want it to be a vector of the same s
hape as the other word vectors in the `word_to_vec_map`.
#     * You can choose a word that exists in the `word_to_vec_map` and access its `.sha
pe` field.
#     * Be careful not to hard code the word that you access.  In other words, don't as
sume that if you see the word 'the' in the `word_to_vec_map` within this notebook, that
this word will be in the `word_to_vec_map` when the function is being called by the aut
omatic grader.
#     * Hint: you can use any one of the word vectors that you retrieved from the input
`sentence` to find the shape of a word vector.

# In[9]:


# GRADED FUNCTION: sentence_to_avg

def sentence_to_avg(sentence, word_to_vec_map):
    """
    Converts a sentence (string) into a list of words (strings). Extracts the GloVe rep
resentation of each word
    and averages its value into a single vector encoding the meaning of the sentence.

    Arguments:
    sentence -- string, one training example from X
    word_to_vec_map -- dictionary mapping every word in a vocabulary into its 50-
dimensional vector representation

    Returns:
    avg -- average vector encoding information about the sentence, numpy-
array of shape (50,)
    """

    ### START CODE HERE ###
    # Step 1: Split sentence into list of lower case words (≈ 1 line)
    words = [i.lower() for i in sentence.split()]

    # Initialize the average word vector, should have the same shape as your word vecto
rs.
    avg = np.zeros((50,))
    # Step 2: average the word vectors. You can loop over the words in the list "words"
.

    for w in words:
        avg += word_to_vec_map[w]
    avg = avg / len(words)

    ### END CODE HERE ###

    return avg


# In[10]:


avg = sentence_to_avg("Morrocan couscous is my favorite dish", word_to_vec_map)
print("avg = ", avg)


# **Expected Output**:
#
# ```Python
# avg =
# [-0.008005    0.56370833 -0.50427333  0.258865    0.55131103  0.03104983
#  -0.21013718  0.16893933 -0.09590267  0.141784   -0.15708967  0.18525867
#   0.6495785   0.38371117  0.21102167  0.11301667  0.02613967  0.26037767
#   0.05820667 -0.01578167 -0.12078833 -0.02471267  0.4128455   0.5152061
#   0.38756167 -0.898661   -0.535145    0.33501167  0.68806933 -0.2156265
#   1.797155    0.10476933 -0.36775333  0.750785    0.10282583  0.348925
#  -0.27262833  0.66768    -0.10706167 -0.283635    0.59580117  0.28747333
#  -0.3366635   0.23393817  0.34349183  0.178405    0.1166155  -0.076433
#   0.1445417   0.09808667]
# ```

# #### Model
#
# You now have all the pieces to finish implementing the `model()` function.
# After using `sentence_to_avg()` you need to:
# * Pass the average through forward propagation
# * Compute the cost
# * Backpropagate to update the softmax parameters
#
# **Exercise**: Implement the `model()` function described in Figure (2).
#
# * The equations you need to implement in the forward pass and to compute the cross-
entropy cost are below:
# * The variable $Y_{oh}$ ("Y one hot") is the one-
hot encoding of the output labels.
#
# $$ z^{(i)} = W . avg^{(i)} + b$$
#
# $$ a^{(i)} = softmax(z^{(i)})$$
#
# $$ \mathcal{L}^{(i)} = - \sum_{k = 0}^{n_y - 1} Y_{oh,k}^{(i)} * log(a^{(i)}_k)$$
#
# **Note** It is possible to come up with a more efficient vectorized implementation. F
or now, let's use nested for loops to better understand the algorithm, and for easier d
ebugging.
#
# We provided the function `softmax()`, which was imported earlier.

# In[11]:


# GRADED FUNCTION: model

```

```python
def model(X, Y, word_to_vec_map, learning_rate = 0.01, num_iterations = 400):
    """
    Model to train word vector representations in numpy.

    Arguments:
    X -- input data, numpy array of sentences as strings, of shape (m, 1)
    Y -- labels, numpy array of integers between 0 and 7, numpy-array of shape (m, 1)
    word_to_vec_map -- dictionary mapping every word in a vocabulary into its 50-
dimensional vector representation
    learning_rate -- learning_rate for the stochastic gradient descent algorithm
    num_iterations -- number of iterations

    Returns:
    pred -- vector of predictions, numpy-array of shape (m, 1)
    W -- weight matrix of the softmax layer, of shape (n_y, n_h)
    b -- bias of the softmax layer, of shape (n_y,)
    """

    np.random.seed(1)

    # Define number of training examples
    m = Y.shape[0]                          # number of training examples
    n_y = 5                                 # number of classes
    n_h = 50                                # dimensions of the GloVe vectors

    # Initialize parameters using Xavier initialization
    W = np.random.randn(n_y, n_h) / np.sqrt(n_h)
    b = np.zeros((n_y,))

    # Convert Y to Y_onehot with n_y classes
    Y_oh = convert_to_one_hot(Y, C = n_y)

    # Optimization loop
    for t in range(num_iterations): # Loop over the number of iterations
        for i in range(m):          # Loop over the training examples

            ### START CODE HERE ### (≈ 4 lines of code)
            # Average the word vectors of the words from the i'th training example
            avg = sentence_to_avg(X[i], word_to_vec_map)

            # Forward propagate the avg through the softmax layer
            z = np.dot(W, avg) + b
            a = softmax(z)

            # Compute cost using the i'th training label's one hot representation and "
A" (the output of the softmax)
            cost = -np.sum(np.multiply(Y_oh[i], np.log(a)))
            ### END CODE HERE ###

            # Compute gradients
            dz = a - Y_oh[i]
            dW = np.dot(dz.reshape(n_y,1), avg.reshape(1, n_h))
            db = dz

            # Update parameters with Stochastic Gradient Descent
            W = W - learning_rate * dW
            b = b - learning_rate * db

        if t % 100 == 0:
            print("Epoch: " + str(t) + " --- cost = " + str(cost))
            pred = predict(X, Y, W, b, word_to_vec_map) #predict is defined in emo_util
s.py

    return pred, W, b
```

```
# In[12]:


print(X_train.shape)
print(Y_train.shape)
print(np.eye(5)[Y_train.reshape(-1)].shape)
print(X_train[0])
print(type(X_train))
Y = np.asarray([5,0,0,5, 4, 4, 4, 6, 6, 4, 1, 1, 5, 6, 6, 3, 6, 3, 4, 4])
print(Y.shape)

X = np.asarray(['I am going to the bar tonight', 'I love you', 'miss you my dear',
 'Lets go party and drinks','Congrats on the new job','Congratulations',
 'I am so happy for you', 'Why are you feeling bad', 'What is wrong with you',
 'You totally deserve this prize', 'Let us go play football',
 'Are you down for football this afternoon', 'Work hard play harder',
 'It is suprising how people can be dumb sometimes',
 'I am very disappointed','It is the best day in my life',
 'I think I will end up alone','My life is so boring','Good job',
 'Great so awesome'])

print(X.shape)
print(np.eye(5)[Y_train.reshape(-1)].shape)
print(type(X_train))


# Run the next cell to train your model and learn the softmax parameters (W,b).

# In[13]:


pred, W, b = model(X_train, Y_train, word_to_vec_map)
print(pred)


# **Expected Output** (on a subset of iterations):
#
# <table>
#     <tr>
#         <td>
#             **Epoch: 0**
#         </td>
#         <td>
#            cost = 1.95204988128
#         </td>
#         <td>
#            Accuracy: 0.348484848485
#         </td>
#     </tr>
#
#
# <tr>
#         <td>
#             **Epoch: 100**
#         </td>
#         <td>
#            cost = 0.0797181872601
#         </td>
#         <td>
#            Accuracy: 0.931818181818
#         </td>
#     </tr>
#
# <tr>
#         <td>
#             **Epoch: 200**
#         </td>
#         <td>
#            cost = 0.0445636924368
#         </td>
#         <td>
#            Accuracy: 0.954545454545
#         </td>
#     </tr>
# 
#     <tr>
#         <td>
#             **Epoch: 300**
#         </td>
#         <td>
#            cost = 0.0343226737879
#         </td>
#         <td>
#            Accuracy: 0.969696969697
#         </td>
#     </tr>
# </table>

# Great! Your model has pretty high accuracy on the training set. Lets now see how it does on the test set.

# ### 1.4 - Examining test set performance
# 
# * Note that the `predict` function used here is defined in emo_util.spy.

# In[14]:


print("Training set:")
pred_train = predict(X_train, Y_train, W, b, word_to_vec_map)
print('Test set:')
pred_test = predict(X_test, Y_test, W, b, word_to_vec_map)


# **Expected Output**:
# 
# <table>
#     <tr>
#         <td>
#             **Train set accuracy**
#         </td>
#         <td>
#            97.7
#         </td>
#     </tr>
#     <tr>
#         <td>
#             **Test set accuracy**
#         </td>
#         <td>
#            85.7
#         </td>
#     </tr>
# </table>

# * Random guessing would have had 20% accuracy given that there are 5 classes. (1/5 = 20%).
# * This is pretty good performance after training on only 127 examples.
# 
# 
# #### The model matches emojis to relevant words
# In the training set, the algorithm saw the sentence
# >"*I love you*"
#
# with the label ❤️.
# * You can check that the word "adore" does not appear in the training set.
# * Nonetheless, lets see what happens if you write "*I adore you*."
#
#

# In[15]:


X_my_sentences = np.array(["i adore you", "i love you", "funny lol", "lets play with a
ball", "food is ready", "not feeling happy"])
Y_my_labels = np.array([[0], [0], [2], [1], [4],[3]])

pred = predict(X_my_sentences, Y_my_labels , W, b, word_to_vec_map)
print_predictions(X_my_sentences, pred)


#
# #### Word ordering isn't considered in this model
# * Note that the model doesn't get the following sentence correct:
# >"not feeling happy"
#
# * This algorithm ignores word ordering, so is not good at understanding phrases like
"not happy."
#
# #### Confusion matrix
# * Printing the confusion matrix can also help understand which classes are more diffi
cult for your model.
# * A confusion matrix shows how often an example whose label is one class ("actual" cl
ass) is mislabeled by the algorithm with a different class ("predicted" class).

# In[1]:


print(Y_test.shape)
print('           '+ label_to_emoji(0)+ '    ' + label_to_emoji(1) + '    ' +  label_to
_emoji(2)+ '    ' + label_to_emoji(3)+'   ' + label_to_emoji(4))
print(pd.crosstab(Y_test, pred_test.reshape(56,), rownames=['Actual'], colnames=['Predi
cted'], margins=True))
plot_confusion_matrix(Y_test, pred_test)


#
# ## What you should remember from this section
# - Even with a 127 training examples, you can get a reasonably good model for Emojifyi
ng.
#     - This is due to the generalization power word vectors gives you.
# - Emojify-
V1 will perform poorly on sentences such as *"This movie is not good and not enjoyable"
*
#     - It doesn't understand combinations of words.
#     - It just averages all the words' embedding vectors together, without considering
the ordering of words.
#
# **You will build a better algorithm in the next section!**

# In[ ]:
```