CS591 HW5
EMOJI Prediction
Ziran Min
U59274427
minziran@bu.edu

Part 1:
"To predict, given a tweet in English (or Spanish), its most likely associated emoji"

I'm only given the original training and testing data sets of tweets and emoji labels and nothing else, so I have to start from scratch to clean data and build models to make emoji prediction.

1. Data Cleaning
   In order to transform strings of tweets to the input that Machine Learning models can read, I need to implement Word Embeddings to vectorize the data. However, the tweets I have are very dirty, I have to clean the data first.

   The following are my data cleaning steps for each line of tweet:
   (1) Use *tweet-processor* package to remove URLs links, SIMLELYs, and "@user" (in my data, all mentioned individual users' names are changed the to "@user", so I should exclude this noise);
   (2) I write code to went through all the tweets data to collect all unusual characters from tweets (anything that is not English characters, Spanish characters, or number digits). The collection of those characters is summarized in the following figure. Now you can see how dirty, how messy, and how noisy the data are. Thus, I remove all those characters and symbols;

   (3) Change words to lower case;
   (4) Remove stop words (most commonly used words);
   (5) Although some common punctuations are included in the unusual characters I find in (2), to double check that there are no symbols in any tweet, I remove all punctuations from tweets;
   (6) To reduce duplicates (like "eating" and "ate" both means "eat"), I convert every word to their stem words.

2. Word Embeddings

(1) TF-IDF Vectorization

TF-IDF Vectorization is a Vectorization method that is better than Count Vectorization because it considers not only the occurrence of a word in a single document but also in the entire corpus.

It works by penalizing commons words by assigning them lower weights while giving importance to some other words that could only appear in some particular sentences or documents.

The formula of TF-IDF for each word is:

TF =
(Number of times word *w* appears in a tweet)/(Number of words in the tweet)

IDF = log(N/n),
where, N is the number of tweets,
and n is the number of tweets a word *w* has appeared in.

TF-IDF = TF * IDF

Therefore, I use *TfidfVectorizer()* to implement TF-IDF Vectorization

(2) Word2Vec Embedding

In *gensim.models* packages, there is a function called Word2Vec which is an algorithm to learn word embedding from a text corpus.

I used Word2Vec functions to train a vectorization model based on learning from all my given tweets. In the end, for each word in a tweet I could get a 100-dimension embedded vectors. To combine vectors of all words in a tweet as a final feature, I simply take average of vectors.

Another better way is to use TF-IDF scores of each word as weights to multiply with each 100-dimension vectors and then take a weighted average. I will do it for further study later sometime.

3. Modeling and Evaluation

I mainly use Logistic Regression and Random Forest functions from *scikit-learn* because these are the only 2 models that can be trained in limited amount of time on my laptop.

The following tables are the summaries of the best Macro F1-scores for English tweets prediction and Spanish tweets prediction I got:

English:

	TF-IDF	Word2Vec
Logistic Regression	21.559	11.132
Random Forest	16.864	11.090

Spanish:

	TF-IDF	Word2Vec
Logistic Regression	13.597	2.849
Random Forest	11.060	5.366

Here is my findings and reflections:

(1) TF-IDF embedding always gives me higher F1-scores than Word2Vec embedding. To improve the performance of Word2Vec embedding, maybe I should import some huge pre-trained word vectors from Wiki News or Google News;

(2) I can get higher F1-scores from English tweets prediction than Spanish tweets prediction, one possible reason is that I have much larger train sample size of English tweets (90,000) than Spanish tweets (19,000).

(3) However, no matter which embedding method or model I use, I can't get an enough high F1-score. The biggest problem of my methods is that I don't consider words sequence and words relationship between neighbor words. My input are words, not sentences, so models can't actually get the meaning of tweet.

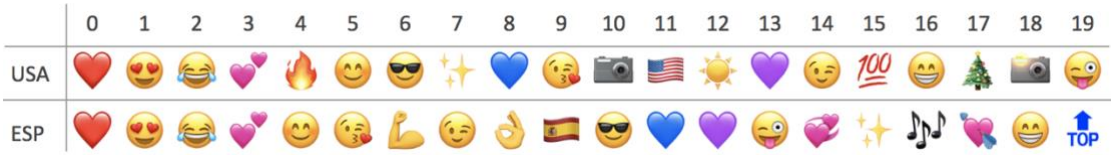
In Data Cleaning, I removed all punctuations, but some punctuations can imply some particular emotions or sentiments

In TF-IDF embedding, I only consider the importance of each word in each tweet, I don't include the features that take word positions and neighbor words features into account.

Logistics Regression is a simple and cheap linear model. Random forest breaks the original order of words. For further study, I should try some models like recurrent neural network that can really learn more from the

sequence and conditional probabilities of words (why this word appears after that words). In that way, I can build model that better understand the context and meaning of a tweet.

Part 2:
To use *any* type of multilingual transfer learning to see if you can use English data to improve Spanish emoji prediction or vice versa.



1. Labels Matching

In order to translate English to Spanish or vice versa and train models, I should unify the labels between English to Spanish first.

There are some labels only in English tweets but not in Spanish, and there are some labels only in Spanish tweets but not in English. Therefore, I wrote code to remove all labels that are not appear in both languages' datasets and corresponding tweets.

As you can see, some labels have different index in two languages. For example, the index of "smiling face with smile eyes" is 5 in English sets but 4 in Spanish sets. Thus, I wrote code to rematch Spanish label indices to the same as those of English.

The labels that need to remove from English: 4, 10, 12, 14, 15, 17, 18
The labels that need to remove from Spanish: 6, 7, 8, 14, 16, 17
The labels that are common are:

English	Spanish
0	0
1	1
2	2
3	3
5	4
6	10
7	15

English	Spanish
8	11
9	5
11	9
13	12
16	18

The indices of Spanish labels that I should rematch to indices of English labels follows the order below to avoid any possible "wrong-rematching". An example of "wrong-rematching" is that: if I change 5 in Spanish to 9 first, and change 9 to 11 later, then when I will change the 9s, that I just transform from 5, again to 11 and those 9s won't be 9 anymore. Therefore, following the order below is very important.

Spanish to	English
11	8
13	19
15	7
18	16
12	13
9	11
5	9

2. Translation and Modeling
   The following is my procedures to test whether Spanish to English translation can improve English tweet emoji prediction.

   (1) I use the new English tweets datasets, that I just manipulate in Labels Matching part, to train a baseline model to prediction emoji in English tweets by TF-IDF embedding and Logistic Regression. I write down base-line F1-score, 22.19.

   (2) I remove URLs links, SIMLELYs, "@user", unusual characters, and punctuations in Spanish tweets (both training and testings sets).

   (3) I use *Translater* function in *translater* package to transform the Spanish tweets that I cleaned in (2) to English tweets.

   (4) In the newly translated English tweets, I take all words to lower case, remove stop words, and transform words to stem words.

   (5) Now I got this newly-translated-clean English training and testing sets. I combine those sets with the original English tweets datasets I mentioned in (1) as new training and testing sets.

   (6) By using TF-IDF embedding and Logistic Regression (same parameters), I fitted the new datasets into the model. This time the F1-score is 15.643

Now we can see after adding translated English tweets into the datasets, the F1 scores decrease. **The Spanish to English translation CAN NOT improve English tweet emoji prediction.**

The reason might be that: The translation function doesn't work well and can't translate every Spanish word to English word, so a lot of long and weird Spanish words well be added into the training set, and this will negatively affect the model performance; Or the Spanish data sample size (19,000) is too small compare to original English dataset (90,000), so maybe more data can improve the performance.

I didn't do the English to Spanish translation part because the translation function I use will take a long time to complete. It is worthwhile for me to find and study a better multilingual transfer learning method later this summer.