Final Project Report

— Word-Based Emoji Predictor

Team Member Names.

- Jheng-Hao Huang : jhhuang@umich.edu
- Hong-Yi Yao : hyyao@umich.edu
- Yi-Sheng Hsieh : yshsieh@umich.edu
- Yu-Chih Tung : yctung@umich.edu

Introduction

With the development of the Internet, people began to communicate with each other online rather than in person. However, since it is difficult for users to express their emotions only with words, users created a method to express their emotions via some special characters, called emoji or emoticons, which are a pictorial representation of human facial expressions, e.g. 😊,😞. In general, users add or select the emoji for the emotion they want to convey at the end of a sentence. Some message systems, such as Facebook, provide a way for users to select emoji by recording recently selected emoticons and suggesting them to users. In our project, we want to build a prediction system that automatically suggests appropriate emojis to users by their text inputs. Our method uses supervised learning to train the prediction system. First, we collect large amounts of training data from *Instagram*, a social networking service site, and do pre-processing to determine feature space. After the feature is established, we use several algorithms to build classifiers which will be introduced in following paragraph.

Problem Statement and Goal

Searching for an appropriate emoji is often a tedious work when you need it. This is a problem especially in mobile platforms due to the limited size of screen. In order to shorten the time spent on emoji searching, we want to build an emoji predictor that suggests appropriate emoji according to user's input text.

Related work

There are two existing methods for users to select emojis: 1) Having keyboards where emojis are in pre-classified groups, and 2) Default one-to-one mapping from predefined syntax to emojis.

In the first method, emojis are usually grouped by "themes". For example, all the yellow-faced emojis such as ☺and 😀, are grouped in to a single category 😊 as shown in *Fig-(1)*. In this method, users need to navigate through different groups to find the right one to use. Based on our knowledge, Android 4.4 default keyboards, iOS6+ default keyboards, Facebook emojis keyboard provide this functionality.

The other one-to-one mapping pops out a single suggested emoji based on a direct mapping from the input syntax. The main difference between this method and our proposed system is that this method show the emojis only when the predefined syntax is entered. For example, as shown in Fig. (2), Android 4.4 will pop out 🐶 when user directly input the text "dog" but it will suggest nothing if users input "doggy" or "doooooog". Another example is that an input text ":(" in iMessage will be directly transfered into 😔. The main problems of this method are that users need to remember what syntax will map into each emoji and that emoji suggestion based on a single word usually can't fit the need of users. For example, it is possible that users want to select 💖 rather than 🐶 for the sentence "I really like my dog". Moreover, it is hard for system to define the texts mapping to some emojis such as 🎬 and 🏩.

The closest existing work to our system is a text-based emoji prediction method proposed in [1]. In that work, 50 selected emojis are divided into 10 disjoint emoji groups based on predefined underlying emotions for each emoji. A linguistic analysis is used in this work to classify each sentence into one of those 10 emotion groups. The main difference of this work to our system is that we suggest a finer granularity of each single emoji rather than only classifying sentence to one of 10 emotion groups. For example, 👋 and 🙏 fit no emotion group defined in their systems but those two emojis are widely adopted among users based on our dataset from *Instagram*. Moreover, suggestion of one among 10 emotion groups to users are not helpful because users still need lots of time to find the target emoji in that group.

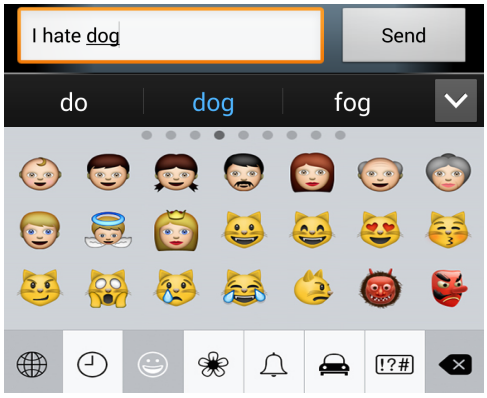


Figure (1): categorized emojis keyboard

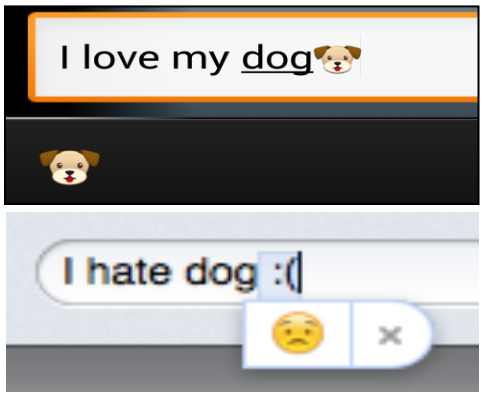


Figure (2): 1-to-1 mapping based on predefined text

Proposed Approach

We want to propose a word-based emoji predictor that suggests the most probable emojis based on the given word set (sentence). For example, given the sentence "I am happy", our system will be predicting a list of smiley emojis, ordered by their likelihood in the context. We will treat it as a multi-classification machine learning problem with a set of words from a sentence as input data, keyword from the dictionary as feature, emoji as target classes.

For the this part of the report, we will structure it in six parts: 1. System Model, 2. Data Source, 3. Pre-processing, 4. Data Observations, 5. SVM classification, k-Nearest Neighbor and Variations, 6. Naïve Bayes

System model

As the system overview shown in Fig. (3), a python script called *EmojiParser.py* is first used to collect Unicode and PNG files of 821 emojis from [2], and those emoji-related data are saved to our local database *emojidb*. After that, another python script called *InstaParser.py* is used to randomly download feeds from [3]. 4 million of feeds with attached emojis are downloaded into another local database called *Instagramdb*. With the collected feeds, *Preprocessor.py* is used to analyze the characteristic of each word shown in feeds and extract proper features from this database. The detailed process at preprocessing will be introduced in the latter section. In total, 5000 words and 1000 bi-words are selected as features and saved at *dictionarydb* in our system.

After those databases are built, different Matlab programs for the corresponding machine learning algorithms, such as Naïve Bayes and SVMs, are processed to train the system models. Those built system models are then fed into evaluator to evaluate performance via numerical simulations. In the mean time, those system models are also incorporated into the Android platform for building the user study application and our final application.

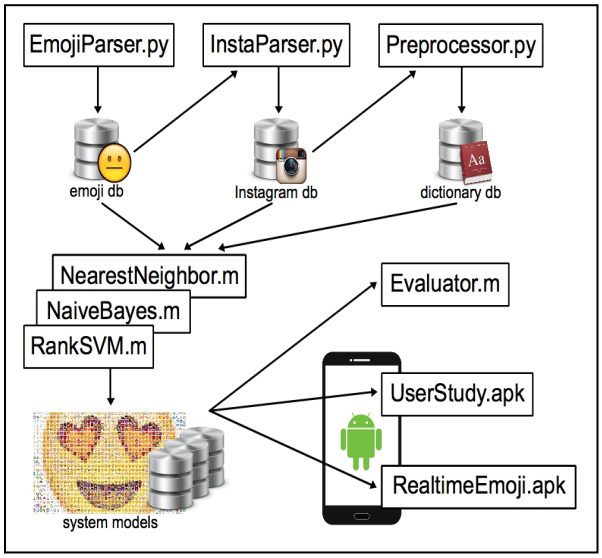


Figure (3): System Model

Data Source

Our data are collected from Instagram with emoji using Unicode-based mapping protocol. The reasons for such choices are described below.

There are three existing emoji protocols commonly adopted for most online chatting rooms, mobile apps and operating systems. The first one is an Unicode-based emoji, which is a predefined 1-to-1 mapping implemented by several systems, such as iOS/OSX/Android/Facebook...etc. The second one is a Sybtax-based mapping protocol. For example, a string ":)" is mapped to a smiling emoji,, and ":(" is mapped to a crying emoji, . Most online chat rooms and legacy web sites adopt this Sybtax-based mapping protocol. The last one is an application-defined emoji mapping, such as the one used in Line and WeChat applications. The benefit of this customized mapping is that developers can extend the emoji database without limitation.

In this project, we exclude the third application-defined emoji mapping protocol because this protocol is

not open to public so it is hard to gather the corresponding dataset. The Syntax-based mapping is the easiest to collect dataset because it is most widely adopted. However, this mapping is not coherent between different systems. For example, GitHub use ":)" to map a smiling emoji, but common chat rooms use "(^_^)" to map the same smiling emoji. Thus, to balance between the usability and the cost to gather dataset, the Unicode-based mapping protocol is then selected in our system. There are 821 defined emojis and the list of these emojis can be found at [2].

Our dataset is gathered from *Instagram*, which is a photo-based online social website. We design a python program to crawl the responses of public photos from this website. We choose this website rather than other bigger online social networks such as Facebook because we can access public response in Instagram. Collected data are stored in SQLite for better interface to Matlab and Android phone. In total we collect more than 4,000,000 data, where 3,000,000 of them are used as training data (including validation data set) and 1,000,000 of them are used to evaluate our system.

Preprocessing

In this project, we will use Bag-of-words model to represent feature for each training data. Originally, the dictionary contains all words appearing in our data set, which is large and contains redundancies. The purpose of preprocessing is to build a dictionary that removes those redundancies and ambiguities. The data set are fed into the following steps:

Stop words:

We build our own stop words list that refers to MySQL full-text stop-words [4] and Natural Language Toolkit (NLTK) [5]. Redundant words such as "the", "this" and "those" will be filtered out before further processing.

Stemming/Lemmatization:

After filtering stop words out, we apply stemming/lemmatizing on the words to deal with ambiguity. This process reduces the inflected words and the derived words to their root form. For example, words "learned" and "learning" are both reduced to word "learn". There are several well-define stemmers/lemmatizer with different strengths. Among all existing methods, the WordNet Lemmatizer [6] is the best. It can identify irregular verbs and maps them to their root forms while other methods are only capable of dealing with variation of regular verbs. However, this method requires the knowledge of whether this word is verb or noun before lemmatizing. This problem is hard to solve even if we use the tagging-tokenize function (also provided by NLTK) that labels the lexical category due to occasional wrong tags. We also tried the Lancaster Stemmer, but it over-simplifies the input words, such as "bef" for "before" and it confuses people with "beef". Therefore, we choose the Porter Stemmer as a trade-off originally but find two benefits soon later. First one is that it can be easily modified with customized function we need which identifies some internet slangs, such as "yessss" which repeats the last alphabet for emphasizing. Furthermore, this stemmer is open-source in different programing language so we do not have to program this stemmer ourselves.

Biwords:

After preliminary preprocessing, the dictionary contains only single word. So we create a new set of dictionary words feature called "biwords", which are combinations of two words. The objective is to recognize

negative meaning in a sentence. For example, “don’t like” is merged into “notlike” which is more precise than “not” and “like”. The reason is that the feature “not” represents negative meaning of multiple words but not exactly the opposite of “like”. In other words, words “not” and “like” together cannot exactly convey the original meaning of “don’t like” while “notlike” can. By adding this feature set into the dictionary, we can capture the negative meaning of a sentence. The results of this biwording process is demonstrated on table (1).

TF-IDF:

Finally, we narrow our feature space (dictionary) by Term Frequency–Inverse Document Frequency (TF-IDF), which identifies non-informative words. This method removes words which are almost equally distributed among all emojis and those which rarely appear. For instance, “you” is the former case and “ambrosia” is the other case. The difference between the stop-words process and TF-IDF process is that stop-words removed grammatically meaningless words such as “a” or “the” while TF-IDF removes words that are non-informative for certain applications. For example, the word “thank you” is an important word in our application but not in other applications.

	sentence “I like you”	sentence “I don’t like you”
Without “Biword”		
With “Biword”		

Table (1): Demonstration of “byword”

Data Observations

After pre-processing, a dictionary is built. However, before we start with our method and algorithms, we are interested in the characteristics of our application data. There are several properties that distinct our application from other similar field of study such as topic modeling, document informational retrieval or recommendation systems.

First of all, user usually have input sentence no more than 10 words which contains 3 – 5 informative words that can be served as features. As a result, very few information is available for us to decide which emoji to predict. This makes it difficult for us to treat our problem as Topic modeling where each document has many words. The cumulative number of word in each sentence is shown in Fig. (4).

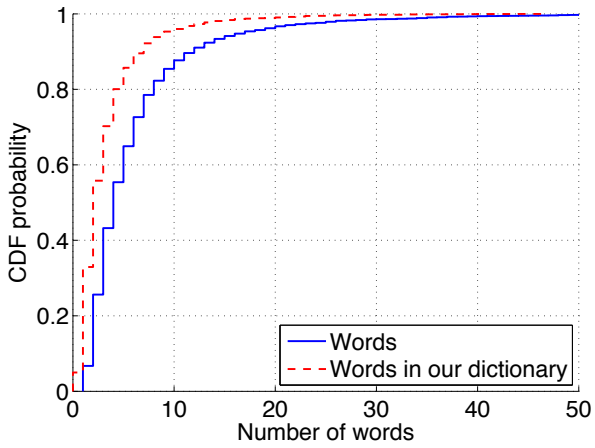


Figure (4): input words length

Second, the distribution of emoji is vey skewed. That is, a relatively small number of emojis are used most frequently. The distribution is drawn in Figure (5). Due to this reason, we will only select the top 100 most used emoji in the rest of our application and report.

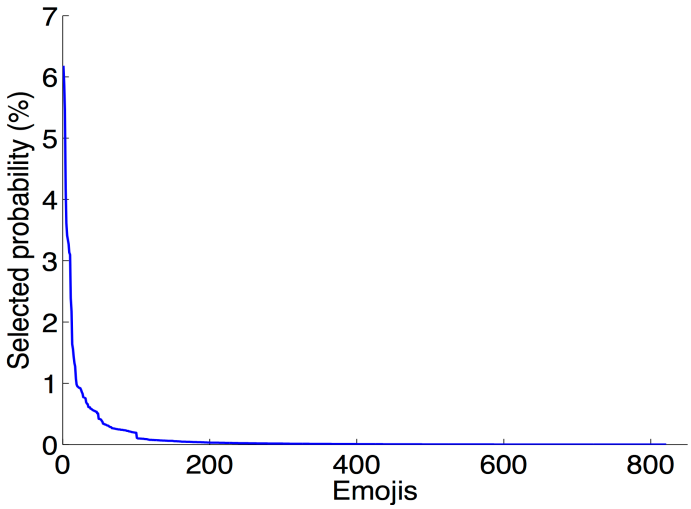
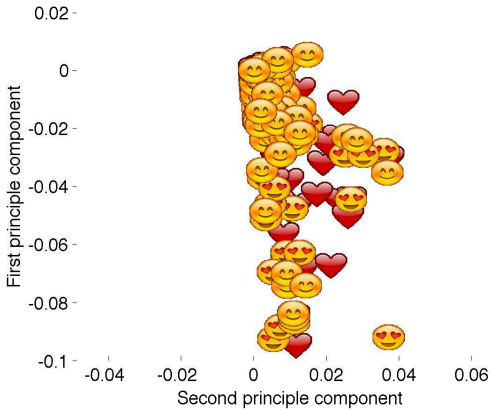


Figure (5): Skewed probability distribution of emojis' prior

Third, many different emojis have similar word feature. This is because users have various preferences in selecting emojis. For example, some people prefer a smiley face emoji after the sentence "thanks" while others prefer a heart-like emoji. To observe this phenomenon, we do Principle Component Analysis and reduce the dimension of each sentence to the two major principle components as in Figure (6). It illustrates the fact that different emojis are intertwined together, resulting in inseparability of data. Table (2) showed how use different emojis when they type similar words/sentences.



Emoji			
Most frequent Word	'thank' 'so' 'love 'beautiful' 'aw'	'so' 'love' 'beautiful' 'cute' 'gorgeous'	'thank' 'so' 'much' 'for' 'love'

Figure (6): Distribution of data in space

Table (2): Most frequent word of emojis

Fourth, we want to see how all emojis lie in the high dimensional space. To do this, we represent each emoji as a high dimensional vector using bag-of-words, and again do PCA by projecting these emojis on their first two principle components. The resulting graph is showed in Fig (7). Interestingly, the first principle component can be interpreted as “holiday” while the second principle component stands for “emotion”. This is because more specific word will be used in special day such as Christmas or Birthday.

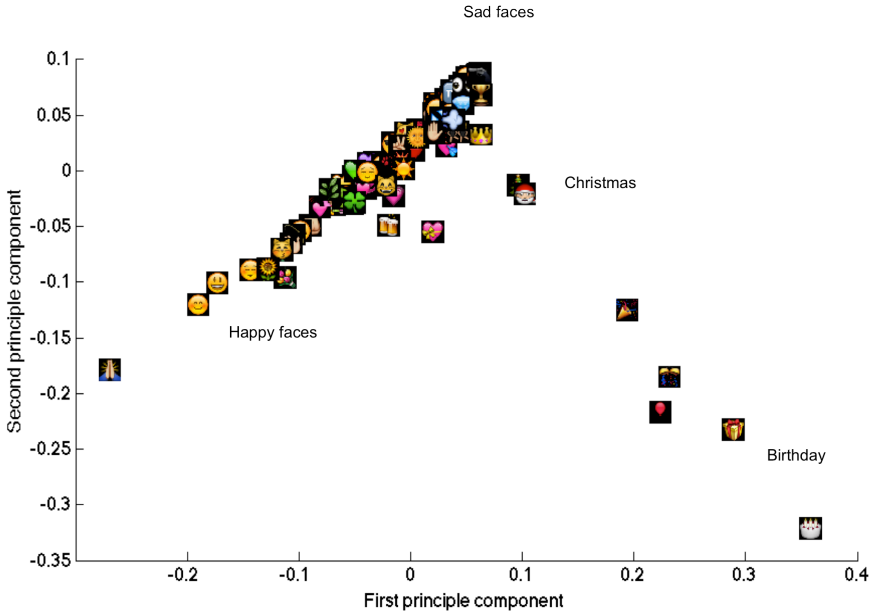


Figure (7): Distribution of emojis in space

Lastly, we want to see how emojis that shared similar meaning can be grouped together. To achieve this, we implement k-means clustering method with K=9 and give each emoji cluster with human interpretation. Again, each emoji is represented as a bag-of-words vector in the space. The result of clustering is shown in Table (3). This result is interesting in the sense that each clustered group has its underlying meaning. It also points out that emoji that associated with “happiness” is hard to dealt with due to its wide range of usage.

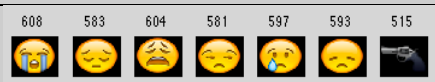
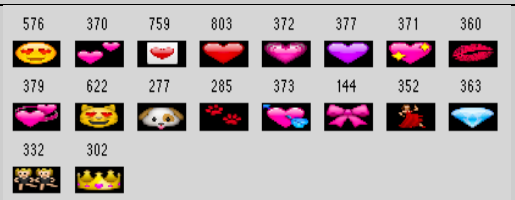
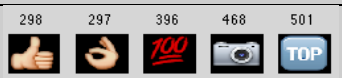
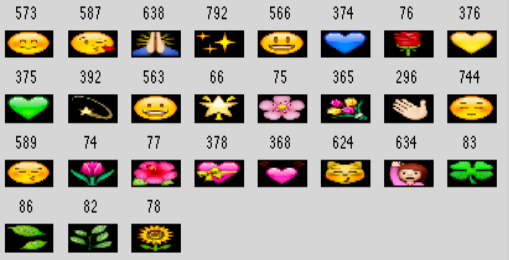
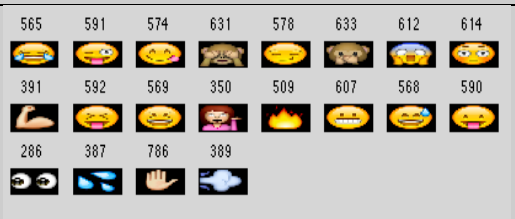
Group	Emojis in group	Descriptions
1		Victory, Applause, Encouragement
2		Christmas
3		Sadness
4		Birthday, Party
5		Love, Heart
6		Approval
7		Happy
8		Happy, Flowers
9		Shock, Embarrassment

Table (3): k-means Clustering of emojis

Support Vector Machine

In the rest of the report, we will mostly use bag-of-words vectors with binary representation as our feature. We will use only the top 100 most frequently used emoji in our data and dictionary described earlier.

We firstly decide to treat this problem as a multi-class classification problem, and we choose to build SVM to classify data. Through data validation, Gaussian kernel fit the best to our data. We adopt the 1-against-all methodology and use the *libsvm* library [7]. That is, we created 100 SVMs for the first 100 popular emojis and feed our training data to these SVMs. One thing worth noting is that we enable "-b" option of *libsvm* which asks SVMs to return probability estimates. With those probability estimates, we can rank the suggested emojis by its corresponding probability estimates.

The main challenge we face in this SVM approach is that there is actually no "clear boundary" among emojis. For example, there are several emojis such as: 💙💕💖💘 representing the same meaning of "heart" and are associated to similar features such as "love", "like", and "valentine". As the result, in the SVM classification, there will be some false negative labels (emojis) being trained as "non-heart" while they should be treated as the other way around. This also leads to very low probability estimate (confidence) for each Support vector machin.

To deal with the problem of low probability estimate of SVM, we reference a method called SVM-HF (SVMs with heterogeneous feature kernel) in [8]. The main idea of this method is to solve the problem of ambiguous region commonly appeared in multi-classification SVM problems as in Figure (8), which is the reason why SVM have low probability estimate in our application. To do SVM-HF, we first train K one-against-all SVMs where K is the number of emojis and expand features in training data depending on its relationship between the K SVMs. The new expanded training data are then put to train a new set of K one-against-all SVMs. Testing data followed the similar procedure where expanded features are being fed into the new SVMs. To see why this will work in solving the addressed problem, we can understand the SVM-HF as putting similar emoji together to reduce the area of that ambiguous region as in Figure (9). Once the ambiguity is reduced, the probability estimate should rise. Unfortunately, the results is not as expected as showed in Fig (10). We found out that SVM-HF has similar performance as the original SVM and this can be accounted for the fact that the inseparable properties of emojis as shown in Figure (6) can harm the performance of both SVM and SVM-HF. As a result, we cannot use SVM-HF to improve our prediction.

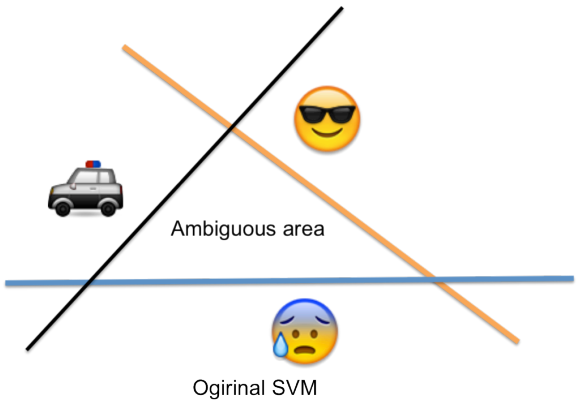


Figure (8): ambiguous area in one-against-all SVM

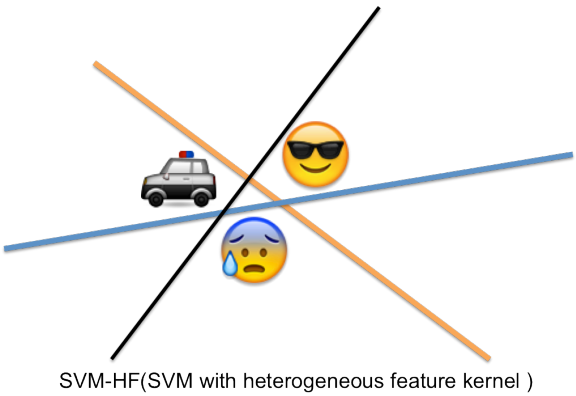


Figure (9): ambiguous area in SVM-H

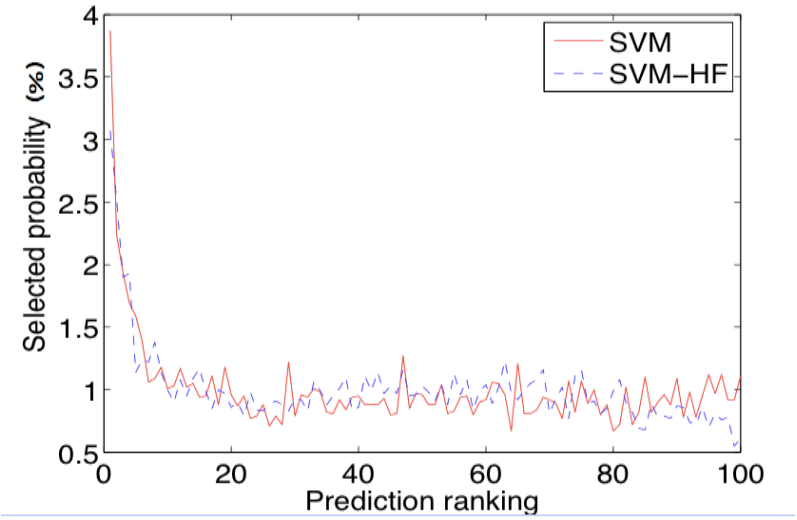


Figure (10): Prediction ranking of SVM v.s. SVM-HF. The X-axis means tha the Xth prediction is correct and Y-axis is the probability(frequency) of such prediction.

To address this issue, we believe that grouping the emoji by clustering them first can reduced the occurrence of false negative emojis when we do SVM or other discriminate modeling. The probability estimate of each individual heart and the probability estimate when grouping them into one heart can be found at figure (10). We plot average probability estimate of all the trace associated with those heart emojis, so the extremely low probability estimate of SVM for individual emoji represents that SVMs are confused among those hearts. On the other hand, grouping all heart emojis together and build a SVM targeting for this group have much more confidence.

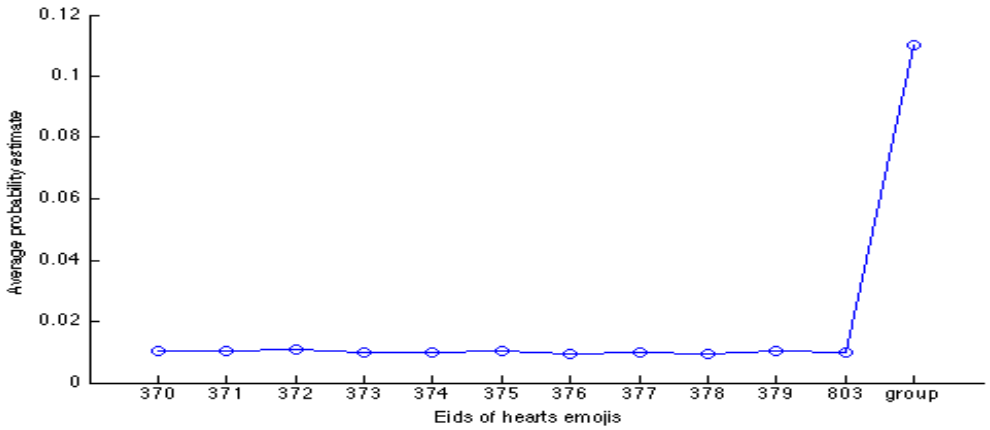


Figure (11): Probability estimate (confidence) of SVM v.s. grouped SVM

As the result, grouping similar emojis seems to be a plausible method to help SVMs to find the boundary. To find groups among emojis, the k-means algorithm is adopted. The results of grouped SVM are interesting in terms of their expression and usage, as shown in the third and forth row of table (4). As shown in that table, the result of SVM w/ group is more coherent compared to SVM w/o group. However, grouping together might impose the problem that we don't know how to rank emojis in the same group. However, how to group emojis

itself is a hard problem because there is no clear boundary between those groups too. For example, the two most prominent groups of emojis are “hearts”, i.e., 💙💖, and “smiles”, i.e., 😀😄, but there also exist emojis that represents both hearts and smiles, i.e., 😍. In addition, we don’t want to constrain our application to only a classification between groups, we decided not to use the grouped SVM algorithms.

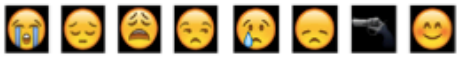
	“I feel happy”	“I feel sad”
Naïve		
Naïve + Normalized		
SVM		
Grouped SVM		

Table (4): Prediction results of normalization and grouping

After understanding the fundamental limitation of our application such as the inseparability of emojis and short input data, we want to treat the problem as Learning to Rank problem. In a traditional ranking problem, one needs to have ranked labels as input training data in order to provide information to build the model. One of such algorithm is rank SVM. In our application, however, we do not have ranked emoji for each input sentence, only one emoji that serves as the “answer” for each sentence. This makes the ranking algorithms hard to be applied to our emoji prediction problem.

However, there are other methods that can capture the properties of our problem. We now choose K Nearest Neighbor (kNN) as our prediction model since it not only can solve the problem of inseparability, but also have to ability to rank the emojis. We put all the training data into a high dimensional space and do kNN. The result is showed in Fig. (15), it has a much better performance than SVM since the problem of inseparability is intrinsically solved. In kNN, the number of neighbor is chosen to be the number of recommendation times the number of training data associated with each emoji. This is to ensure that we have a reasonable range to include all emojis that we are interested in.

The draw back of k Nearest Neighbor is that the time complexity is relatively high in the testing phase compared to other algorithms. It is an undesirable property since in the real application, we want the prediction to be fast to produce responsive system performance. For this reason, we turn ourselves to a generative model, Naïve Bayes classifier.

Naive Bayes Classifier

Due to the difficulty we meet while applying SVM, we turn to build a probabilistic model. We choose Naive Bayes classification and compare it with a baseline that suggests emoji depending purely on emojis' prior probability.

Before implementing the algorithm, we first notice that the prior distribution of emoji is far from uniform. As figure (5) shows, the first 100 most frequently used emojis occupy over 90% of the probability density, which means that we can roughly get 9 successes in 10 prediction by always providing the first 100 frequent emojis. This skewed distribution causes two problems. The first one is that it makes us difficult to evaluate our performance because the successful rate is primarily dominated by those emoji with higher prior. In other word, even we have 100% accuracy in suggesting the later 80 frequent emojis, we might get 50% overall accuracy when we fail to suggest the first 20 frequent emojis. Another problem is that we might get an unreasonable suggestion due to the prior of emoji, which is demonstrated by testing two sentences with absolutely opposite meaning, "I am happy" and "I feel sad". In table (5), the top-left figure is the suggestion for "happy". We can note that the third and fourth suggested emojis also appear at the first two suggestions in top-right figure, which is for "sad". Those emojis are the two most frequent emoji with prior 0.0652 and 0.0641 respectively.

	sentence "I am happy"	sentence "I feel sad"
Original (Un-normalized)		
Normalized		
Weighted Prior		

Table (5): Demonstration of suggestion results (by Naïve Bayes)

To solve this problem, we normalized our dataset by equally assigning fixed number of data to each class (emoji), which removes the influence of prior. On the middle row of table (5), the suggestion makes more sense than that without normalization. The testing result is shown in the right plot of figure (13) compared to the result for raw data set on the left. On the right side of figure (13), we see almost 30% improvement from the baseline. However, when the size of suggestion pool is 24 (Due to the screen size of the mobile phone, we suggest user with 24 emojis on each page), the successful rate drop down 15% compared to original (un-normalized) result, even though the gap between Naive Bayes and baseline is larger. This outcome results from assuming all emojis are equally likely chosen before typing any word, which is not the case in real life. This situation leads to weighted prior solution.

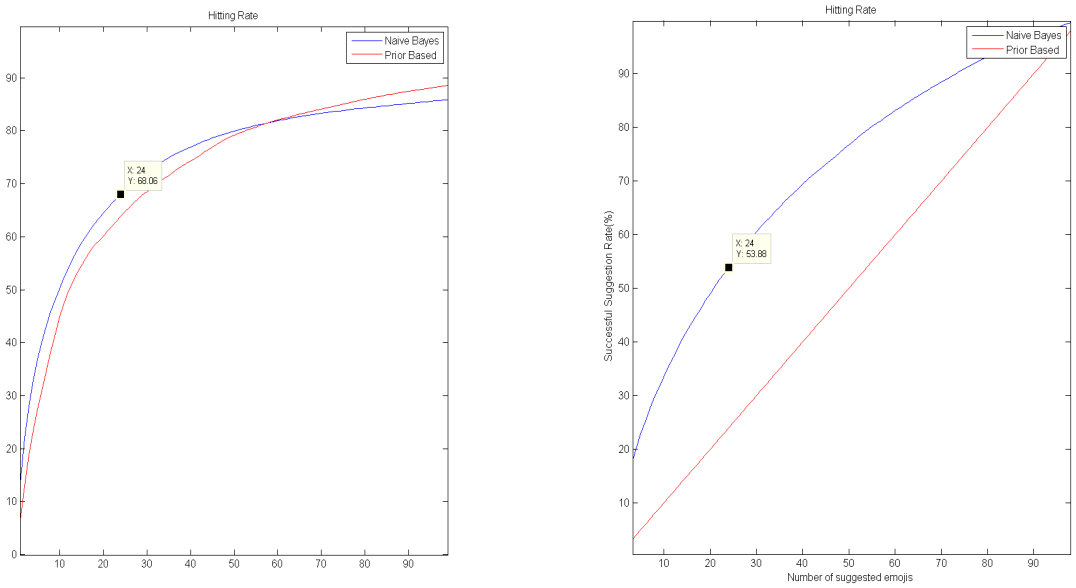


Figure (13): Successful suggestion rate respect to the number of suggestion emojis. The left figure is tested using whole dataset, and the right is for normalized data set. If true emoji is in our suggestion pool, we consider it as a success. Otherwise, it fails.

Instead of normalizing our data (which also discards parts of data), we use the whole data set but modify the prior of each emoji after training. Here is how we modify the prior. We first sort emojis by their prior and then group every 24 emojis together. Each member within a group is assigned their prior by group's average prior. This method balances the difference between emojis because now the most frequent emoji in a group shares the same prior with the least frequent emoji. In this case, we still consider the influence of prior in some sense rather than ignoring them, as in the normalized case. The suggestion for same sentences is shown in the last row of table (5). The testing results of the classifier trained by normalized data and the classifier trained by whole data but with weighted prior is showed in figure (14). On the left side, two classifiers are fed by the whole data set. Weighted prior works much well than another. On the right side, in the case of normalized data, they are compatible since normalized classifier is benefit from the normalized data.

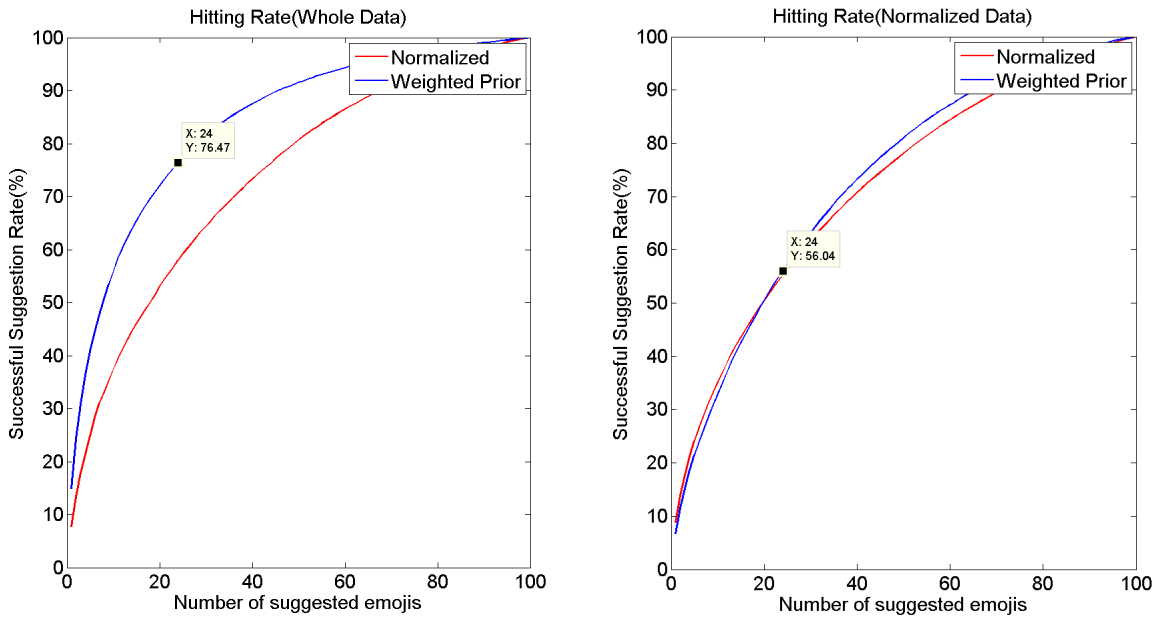


Figure (14): Successful suggestion rate respect to the size of suggestion page. The left figure is using whole dataset, and the right is for normalized data set.

In summary, there is no obvious drawback of Naïve Bayes classifier in our application. As a result, we decided to implement it for our Andriod platform.

Evaluation

We evaluate the successful suggestion rate of our system by the following method: in each test data, we locate position of true emoji in the suggestion pool, and compute the percentage of successful suggestion for each position. For example, if we successfully suggest an emojis at the third position (the third most probable) in suggestion pool, we denote the result as 3. On the other hand, if we fail to suggest right emoji within the 100 suggestions, we ignore it. In the end of testing, if we get 76 successful suggestions over 1000 test date at first position and 69 at second position, the successful suggestion rate for 1 and 2 will be 7.6% 6.9% respectively.

Figure (15) shows the numerical results of all algorithms we discussed in the previous sections. We can see that k Nearest Neighbor, due to its intrinsic property (when you think the data in a high dimensional space), perform the best. Naïve Bayes also have good performance. SVM, which suffers from the inseparability of data in our application, has the worst performance. This matches the explanation and reasoning described in the earlier section and the results is as expected.

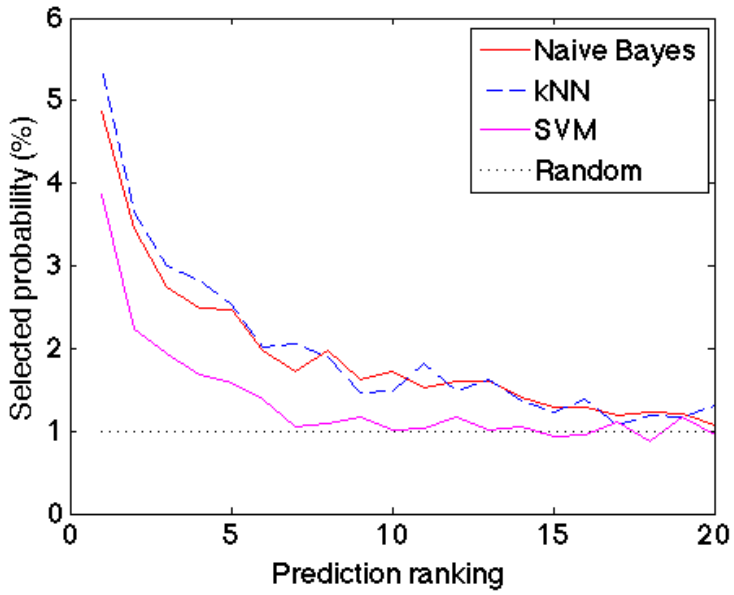


Figure (15): Numerical Result comparison between NB, kNN, SVM and Random prediction

Now that we decided to use naïve Bayes as the algorithm for our application, we want to compare the difference between using whole dataset and normalized dataset. As Figure (16) shows, our system have over 75% successful rate in whole dataset, and suggest correct emoji at first position with approximately 13% accuracy. However, as mentioned previously, this statistic result under-estimates our system since reasonable prediction might be consider as wrong answer in statistic method. In other words, some emojis are similar but considered as different in statistic sense. For example, when system suggests any one of "heart" type emojis to a testing sentence with specific heart, it should be consider as a success, but in case of statistic testing, it is identified as a failure. Because it is hard to find a statistic metric to evaluate the performance of our system, we conduct a user experience survey as a solution for evaluating our system.

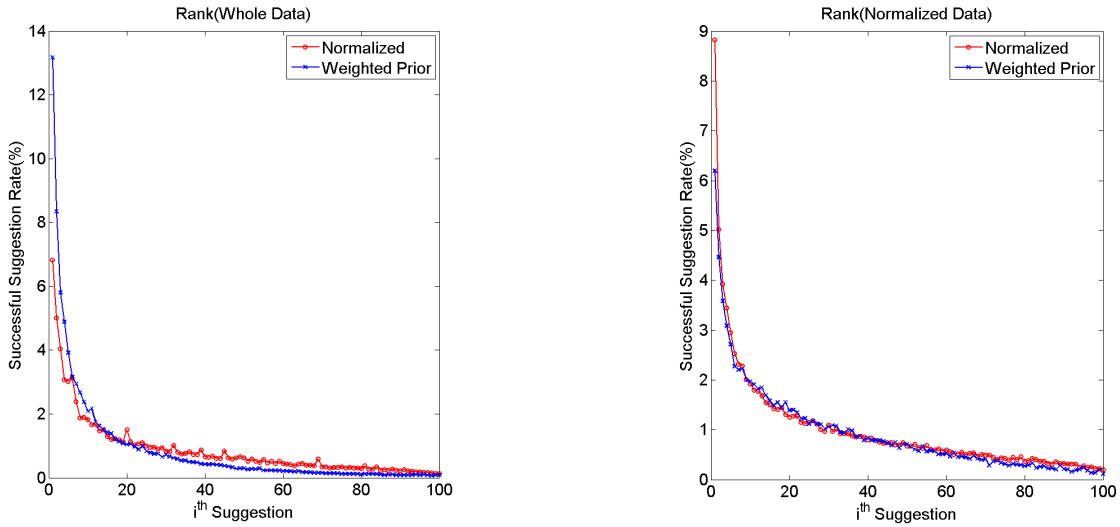


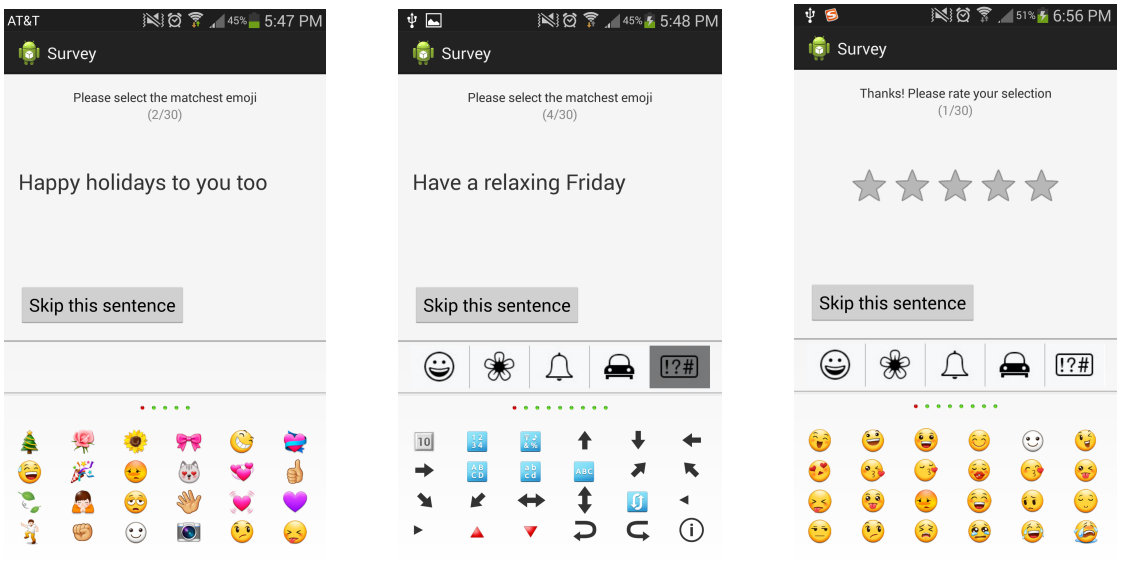
Figure (16): Successful suggestion rate respect to each position of suggestion page. The left figure is using whole dataset, and the right is for normalized data set.

User study and Android Implementation

User study:

Besides using the emoji in test set as the ground truth to evaluate our prediction, we would like to further understand the quality of our prediction in the view of real user. Therefore we implemented another application for user study.

Here is the overview of our user study program:



For each user, 30 sentences will be sequentially displayed, which are randomly selected from test set but without the emoji. And the user is asked to fill in a proper emoji for each sentence and report the satisfactory degree for this selection. The user can also skip any sentence if they do not understand or like it. There are three methods that will display the emoji list but only one method will show up each time. The first method directly shows the result for our Naive Bayes Classifier prediction algorithm. The second method provides the results in the first method but put emojis in random order in each page. The third method categorizes emojis with similar meaning in the same tab, which is used by the existing emoji input method described in Figure (1). These three methods will be randomly used for each of 30 sentences. For efficiency purpose, we precomputed the prediction for each sentence in the test set and loaded into the app. However, there is no difference between it and the real prediction algorithm implemented in the Emoji Chatroom.

In user study, we capture several metrics for evaluation.

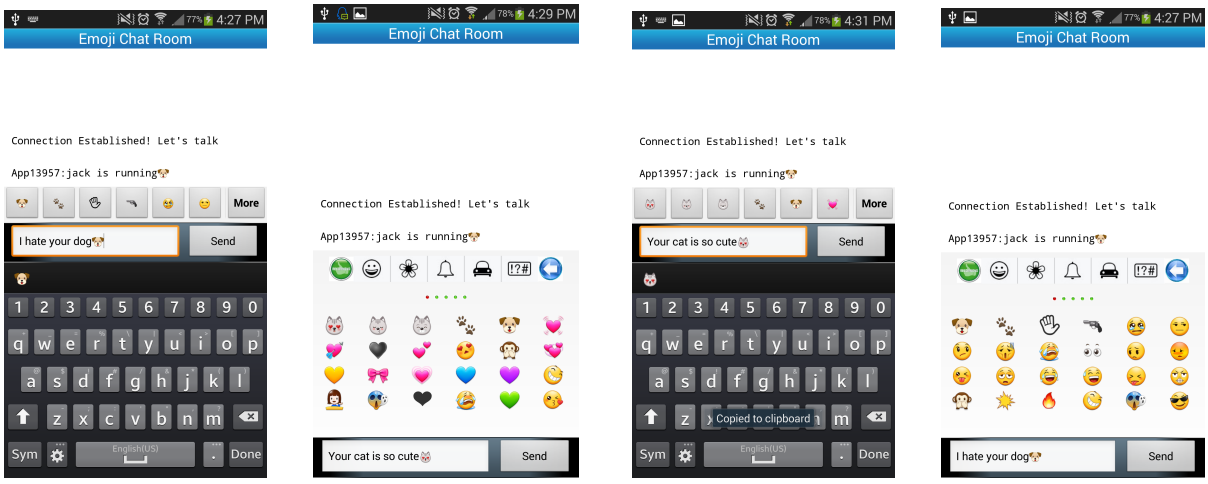
1. Algorithm: the method to display emoji list for this sentence
2. Delay: The delay from the user saw this sentence until he made the choice
3. Emoji id: the emoji they choose
4. Satisfaction degree: satisfactory degree for their selection, in 10 levels

The purpose for this user study is to discover whether our algorithm can provide a good prediction list and save more time to find the user's expected emoji compared with traditional emoji input method. The results are shown in the evaluation part.

Android Implementation : Emoji Chatroom

We build an online chat room app, called Emoji Chatroom, on Android platform as the prototype app for our word-based emoji predictor. Besides basic message sending and receiving function for regular chat room application, Emoji Chatroom will automatically recommend emojis that users are most likely to use in real time based on user's input text. The user can also select emoji in traditional way by pressing "more" button.
We only need to implement the prediction part of algorithms in this app, for the training step is accomplished by matlab process discussed above. According to the evaluation for different algorithm on Matlab, the Naive Bayes algorithm has the fastest prediction speed and nearly the same prediction accuracy compared to KNN which has the highest accuracy. Therefore, we decide to implement Naive Bayes Classifier.

Here is the overview of our system:



And the following two figures shows how emoji recommendation changed with different sentence:

When app starts, it will construct the likelihood matrix which each element is P(word(wid) = 1 | Emoji=eid) based on the database. The prediction procedure works as following

- Load prior and likelihood matrix into this app
- while (the text in the input textfield is changed) do
    - fetch the text, remove all the non-character symbols and repeated characters, and change to lowercase.
    - Split the sentence into tokens(words) and apply Porter Stemming Algorithm[cite: http://tartarus.org/martin/PorterStemmer/] for removing the commoner morphological and inflexional endings from words in English.
    - Calculate the posterior probability for each emoji(class) based on those tokens.
    - Finally, sort all emoji with the probability and select top 18 as the recommendation.

Here is the pseudo-code for emoji prediction in a naive way:

Algorithm 1: plain Naive Bayes Classifier

```
while(user input changed) {
        fetch user input and split into words
        for ( each emoji ) {
                posterior = prior
                for ( each word in dictionary ) {
                        if ( word exist in user input )
                                posterior *= P(word=1 | emoji)
                        else
                                posterior *= (1 - P(word=1 | emoji))
                }
        }
        sort emoji with posterior probability
        return top 20 emoji;
}
```

The main problem for Algorithm 1 is the prediction speed. For each emoji, it need to do production with all words in the dictionary, which is a huge amount of calculation for resource-constrained platform like cellphone. In fact, the size of our emoji set is 800 and the dictionary contains 5000 words, then we need to do 800*5000=4,000,000=4M production operations for one recommendation process. And it will cost above 5 seconds to finish one recommendation loop. That's far from satisfactory for an interactive application. So we must optimize this process.

Luckily, we found in average, there is less than 10 valid words in user's input texts, which is quite small compared to dictionary size(6000 words). So in loading phase, we can calculate a pre-computed posterior distribution with zero input. Then when we got a new set of token, we can update the pre-computed distribution by P(emoji | tokens) = P(emoji | tokens) / P(token=0 | emoji)) * P(token=1 | emoji); It runs much faster than the original one, which needs only less than 100 millisecond for a prediction.

Here are another two optimization to ensure correctness:

1. This algorithm will fail when there exists P(token=1 | emoji)=1 in the likelihood matrix. It means both posterior and P(token=0 | emoji) are all 0 and leads to 0/0 in prediction. To solve this problem, in pre-computing stage, we should keep record of the tokens which P(emoji | token=1) = 1 and skip this calculation (multiply by 0).
2. Finally, in our dataset, some token in the dictionary will not appear together with any emoji. That means for any emoji, P(emoji | token)=0. In this case, if this token appears in the input, the whole prediction is meaningless because the posterior probability is 0 for any emoji. Note that we assume P(emoji | token) is exactly 0 in that case, which is not necessary because it won't affect the ranking if all posteriors are multiplied by the same probability. To solve this problem, we apply a modified Laplace Smooth to the dataset by setting a very small probability to this token(e.g. the smallest probability in the likelihood matrix divided by 10).

Therefore, we got the final algorithm:

Algorithm 2: optimized Naive Bayes prediction

```
for each emoji, set P(emoji | tokens) = precompute(emoji)
for each emoji {
 tokenList = getTokenRecord(emoji)
 if ( tokenList is not empty ) {
  for each token in tokens {
   if ( token is in tokenList ) skip calculation (multiply by 1), remove it from tokenList, continue
   else if ( token is include in dictionary )
    P(emoji | tokens) = precompute(emoji) / P(token=0 | emoji)) * P(token=1 | emoji);
   else
    P(emoji | tokens) *= (minimum_probability / 10);
  }
  if ( tokenList is not empty ) P(emoji | tokens) = 0
 }
 else {
  for each token in tokens {
   if ( token is include in dictionary )
    P(emoji | tokens) = P(emoji | tokens) / P(token=0 | emoji)) * P(token=1 | emoji);
   else
    P(emoji | tokens) *= (minimum_probability / 10);
  }
 }
}
```

By using this algorithm we can both achieve good results for accuracy while keeping good efficiency. The average delay for predicting is around 100 millisecond, which allows us to do the prediction once the user have a key stroke. In our test, you can hardly feel the delay for prediction process.

Other Challenges: Initial loading time

In order to construct the word dictionary, prior matrix and likelihood matrix, we need to load data from sqlite database. It will cost long time to load the entire likelihood matrix since this table has 60K records and the available memory for Android app is small. However, we realized that it is still possible to make prediction before finishing loading the likelihood matrix, for the unloaded cell can be viewed as P(emoji | token)=0 and will be updated later. And this method works well in practice.

Survey result

In the survey of 41 participants, as shown in Table (6), the average time for participants to find target emojis to select is improved from 10.9 seconds (with existing grouped emoji keyboard) to 8.2 seconds (with our real-time emoji prediction). Moreover, the average rank (defined as the position of the emojis being selected) in our system is 14.6, which implies that most people select emojis predicted by our system shown in the first page while the average rank of existing systems is 33.6 (which used categorized emoji). As shown in Fig. (17), the first 24 emojis (shown in the first page) will be selected with 81.32% probability. One thing worth noting is that this probability is dramatically higher than the results evaluated by numerical simulation. This is because in the real world, users are satisfied with multiple emojis rather than one. For example, while the "right answer" of input sentence "What a nice picture" in our dataset is 🙌, and this emoji is not shown up at the first page based our prediction. Participants answering this question in our survey actually select the alternative two emjis, 💝 and 😘 suggested by our system because they think both of them fit the sentence well. However, this property doesn't imply that users of our system compromise themselves to use only emojis suggested by us. As shown at Table (6), our system also provides the highest satisfaction rating among participants for the emojis selected by them.

	Categorized emojis	Random order	Emoji prediction
Time (s)	10.9	9.9	8.2
Rating (1-5)	3.9	3.8	4.1
Index	33.6	25.4	14.6

Table 6: User Survey statistic

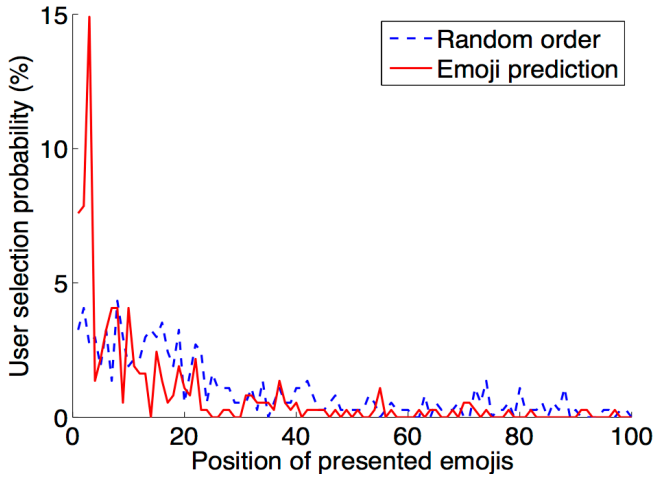


Figure (17): Rank of survey

Conclusions and Future Work

In this project, we provide a real-time emoji suggestion system based on input sentence. Input sentence is filtered, stemmed and merge into biwords (if needed) before parsing into our system. There are three approaches in training classifier. The first approach is SVM and its variations. Because of the inseparable property of our dataset, SVM cannot classify well. The second is kNN, which obtains the best performance while it might take a long time to predict one sentence. The last one, Naïve Bayes and its variations are used. Those methods are intuitive and easily implemented on mobile device by a probability look-up table. As a result, we choose Naïve Bayes when we demo our system.

Future works will be linguistic analysis and on-line training. We would like to include linguistic analysis into our system is due to an observation. Intuitively, classification will be more accurate when we observe more words in a sentence. However, the performance becomes worse when the number of features (words) goes up as illustrated in Figure (18). As a result, this system can be benefitted from further analysis on understanding the meaning of input sentence. Our current system understands language on an extremely basic level by simply mapping words to their root words and highlighting the negative mean of “not” via biwords. While gaining more knowledge of language, the system can predict emojis not only by supervised learning way but also by semantic analysis. It is expected to be more accuracy. In terms of on-line training, our ultimate goal is building a customized emoji predictor which adapt its prediction criteria by collecting user’s typing pattern. The challenge is trading off between the model learning by off-line training and stochastic update model that fed by user input.

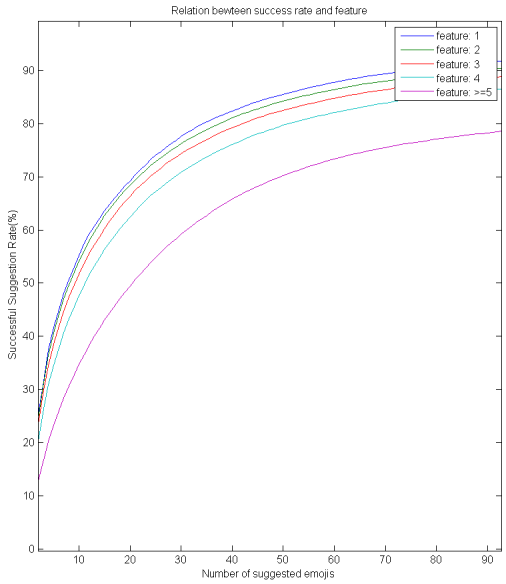


Figure (18): Data with more features confuses classifier and leads to worse performance

Reference

[1] Urabe, Yuki; Rzepka, Rafal; Araki, Kenji, "Emoticon recommendation system for effective communication," Advances in Social Networks Analysis and Mining (ASONAM), 2013

[2] http://apps.timwhitlock.info/emoji/tables/unicode#note5

[3] http://instagram.com/

[4] http://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html

[5] link: http://www.nltk.org/index.html

[6] http://tartarus.org/martin/PorterStemmer/

[7] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[8] *http://www.godbole.net/shantanu/pubs/multilabelsvm-pakdd04.pdf*