COMP-SCI 5542 (SP17) - Big Data Analytics and Applications

Project Report 2 - Due 03/20/17 by 11:59 PM

Yunlong Liu **(22)**
Chen Wang **(44)**
Dayu Wang **(45)**

1. Project Objectives

1.1. Significance

The *image auto-caption* technique has successfully connected image files with text descriptions of the image. Based on this technique, how to connect images with something else became a quite hot topic in data science. In this project, we would like to attempt to link people's moods to representative images about the moods.

Emoji expression is not new things to smartphone era. Is the earliest otago applied expression in language communication scene happened on November 19, 1982, Carnegie Mellon university, Scott fireman (Scott Fahlman) on the university's internal electronic bulletin board (that is the first online chat rooms) issued the ": -)" and ": -)", because this one small action, Scott became the change of the Internet, one of 40 people.

There already exists a very popular visual description of people's moods, which is the **emoji impressions**, normally used in social media sphere and chatting applications. Since sometimes pure text message could be misunderstood, letting the receiver incorrectly speculate the actual meaning of the text, emoji greatly prevents such a misunderstanding in most cases by presenting not only the text but also the mood of the sender when he/she was sending the text. Emoji has been proved to be a splendid invention in modern social media sphere.

After the proliferation of the smartphone, instant messaging tool to become the most widely used applications, let Emoji expressions have a wide range of soil.

In this project, we would like to do **bidirectional linkage between emoji impressions and people's moods**, which means when an emoji is given, a mood is returned to describe the emoji impression; also, when some text is provided, our system can automatically add emoji labels to the text. This technique might save a lot of time for those bloggers and social media fans, for that they do not need to enter both the text and emoji, based on the fact that in most cases, text and emoji consistent in a post in social media.

1.2. Features: Use Case/Scenario

The project has two main different features: *impression-to-mood* and *text-to-impression*, which completes the bidirectional attribute of the project. Table 1 (a) lists the features' description of our system in the viewpoint of the user, and Table 1 (b) lists the features' description of our system in the viewpoint of the developer.

This project contains mainly two parts, the *research* part and the *practical* part. In research part, we focused on the recognition/classification of images, as well as natural language processing. Noticed that most image recognition machine learning/deep learning system has a very low accuracy (all kinds of models behave worse than human beings), our model is not expected to be competitive with human beings, yet reasonably generates trending results. Also, we are looking for a way to use the natural language processing to help image processing, since sometimes people may manual provide additional information, such as image caption, image description, question in natural language about the image, and so on.

Table 1. Description of the features of the *Emoji Interpreter* system from the viewpoint of the user (a) and the developer (b).

(a)

Feature	Description
• Text-To-Impression	• Given an input of text (single word, single sentence, or a paragraph), the system returns the top 3 related moods (emoji impressions) to the user, 1 default emoji and other 2 substitutions for the user to choose. The user will have an option to tweet it directly. The user will have an option to tweet it or justify the emoji with the one they like.
• Impression-To-Text	• Given an input of an emoji image file, the system returns the top 10 popular social media posts related with the input emoji. The user can choose the most appropriate one for himself/herself and tweet it.

(b)

Feature	Description
• Text-To-Impression	• Given an input of text (single word, single sentence, or a paragraph), the system applies the decision tree method, which was used to build the machine learning model to study people's moods, to correctly find out the possibly related moods. Based on the confidence values, the system sets the most possible mood as the default and returns the second and third most related moods to the user as well. We'll reconstruct words contexts with our own models and uses the word K dimensionality dense vector to express words contexts and user's mood.
• Impression-To-Text	• Given an input of an emoji image file, the system applied the Clarifai API to find out the main topic of the image, which is considered the key words of the emoji. Based on the key words, we try to find the most related text description of the mood the emoji tried to show us. Finally, we generate 10 most popular tweets which used the mood we just figured out and show them to the user. We will search 100 most recently tweets which contains the mood that the user may be and show the tweets to the user. Since Twitter is not the main part in this project, we will just apply its REST API for our convenience.

2. Approach

2.1. Data Sources

Table 2 lists the data source for each part of our *Emoji Interpreter* system, which includes the training data, test data, and presentation sample data. Our data was collected manually from the internet.

Table 2. Data source and collection method for each part of the *Emoji Interpreter* system.

Data Description	Size of the Data	Data Source	Collection Method
Image Training Data	800 Emoji Images	Multiple Devices (Apple, Android, Samsung, Twitter, Facebook, etc.)	Manual Collection
Text Training Data	1000 Tweets	Twitter	Twitter API (4j)
Searching Text	Entire Twitter	Twitter	Twitter API (4j)
Representing Data	8 Emoji Images	Website	Manual Setup

2.2. Analytic Tools

Table 3 lists the current analytic tools we would like to use in our project. Also, we attempted to explain about how the analytic approaches may help us solving different kinds of problems. Some of the techniques are mature and some of them are still being developed.

The analytic tools we are using in this project are basically from the COMP-SCI 5542 (SP17) (Big Data Analytics and Applications) lectures and lab tutorials. In this iteration, deep learning tools (e.g., Tensorflow) are also used to provide addition dimensions and accuracy. However, the tools mentioned below are mostly conceptual and theoretical, whereas our target is not only the machine learning task but also the presentation of our project results. Therefore, in addition to the tools below, many presentation tools are also included in our project, such as Tomcat web server, SVG image programming, and other tools necessary to complete the project.

Also, we would like to remark a failure part in our implementation of the machine learning system here. Unlike text processing or natural language processing, image processing and recognition requires more powerful computation ability, since images are big and the will be classified as the container of thousands of dimensions. Therefore, in some of our cases, the Spark machine learning system reported “failure” in some of our tasks, which was logged as “out-of-memory error”. Therefore, though we prepared sufficient amount of data for our project, not all the data contribute to the machine learning part. We have attempted several times to try to fix our training data/testing data, in order for a more reasonable training model for our practical application to use.

Since our project is a web application, we have limited types of interactions between the user and the system. Nevertheless, we may add more kinds of communication to facilitate the user in future.

Table 3. Analytic tools with explanations that will be applied in the development of the *Emoji Interpreter* system.

Analytic Tool	Explanation
• Clarifai API	• We apply the Clarifai API to detect the topics from image files. The detected topics will be used as the text descriptors of emoji impressions, in order to correctly identify the related objects from our machine learning models. Clarify API can help the system to find the main topic of the emoji, then the topic will be used for training and testing the system. Clarify API can help the system to find the main topic of the emoji, then the topic will be used for training and testing the system.
• Twitter API (4j)	• Twitter is the most popular social media platform which includes a "sea" of emoji impression with text. Text from tweets can explicitly give a prevalent interpretation of the emoji impression. Based on the study of the text and emoji within the same tweet, we can well build our machine learning model for our system. It'll extract the most popular twitter has been post with this emotion. When user input an emoji to express their emotion, the most probable saying they want to post will be show then they can use it directly.
• Apache Spark	• We use Spark basically for the text processing part of the system, which includes TF-IDF, LDA, Word2Vec, and etc. If we cannot find the class from our pool of data which perfectly match the user's input, we will use the most similar words as our keys instead. It uses the word K dimensionality dense vector to express; the training set is a corpus, including punctuation, pausing by Spaces.

2.3. Analytical Tasks

Our analytical tasks include mainly two parts: the **machine learning model** and the **relation extraction** for the emoji impressions with people's moods. Figure 4 demonstrates a decision tree example (not final version) of the relationship of those emoji impressions. For our relation extraction part, differ from classic method found in most textbooks (e.g. IBM Watson), we used a "triangular overlapping" method which is novel and completed designed by ourselves. Nevertheless, we never discarded classic approaches. For instance, we used classic "nearest-neighbor" method which includes Spark `Word2Vec` module. The algorithm will be described later in this project iteration report.

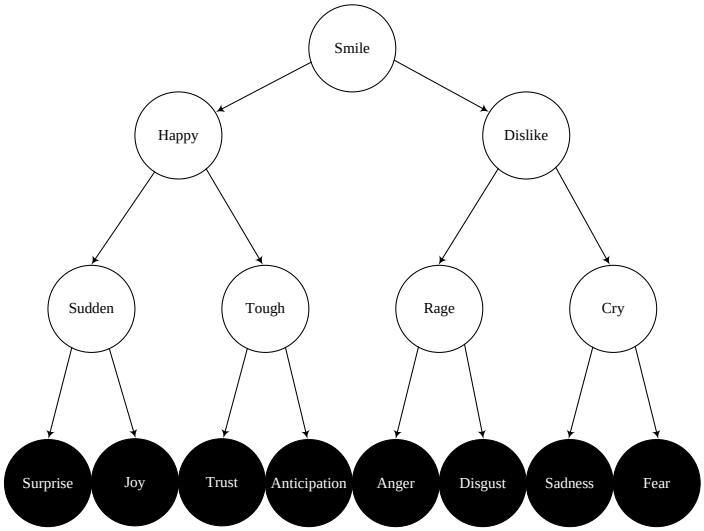


Figure 4. Decision tree model of emoji impressions based on the Plutchik's Wheel of human's moods (left → "Y"; right → "N").

Based on the decision tree model of those impressions, we can further build an ontology model that connects a specific mood with several tweet texts. Based on the time limit and the granularity requirement of the project, this level will be our stop point for the project.

2.4. Expected Inputs/Outputs

Table 5 shows the expected inputs and outputs for the two features of the system.

Table 5. Expected inputs and outputs for the *Emoji Interpreter* system.

Expected Inputs	Expected Outputs
• JPG image	• A list of 10 most popular text posts from Twitter with similar impressions as the input emoji file.
• Text input (Paragraphs, words, sentences, etc.)	• Three emoji impressions, 1 default and 2 substitutes, for the user to select one that best describe his/her current mood.

2.5. Algorithms

In this project, we are using some of developed algorithms in machine learning and image classification/annotation area. Most of the algorithms are built in Apache Spark or Google Conversion APIs. Also, we will have our own algorithm to build our own machine learning model (see Figure 4). The specific algorithm is currently being developed. In the **Implementation** section of this report, specific algorithmic specifications will be elaborated. Similar to our previous projects, this time we attempted to implement a "facet-dependent sentiment analytical system for image recognition".

3. Related Work

3.1. Open Source Projects

3.1.1. OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source project that contains hundreds of computer vision algorithms. It provides a huge amount of image processing operations, ranging from basic operations, like *image filtering* and *geometric transformation*, to smart machine learning models, like *histograms*, *feature detection*, and *object detection*. OpenCV 1.x API is C-based and Open CV 2.x is C++-based. It Java-based API is currently under development.

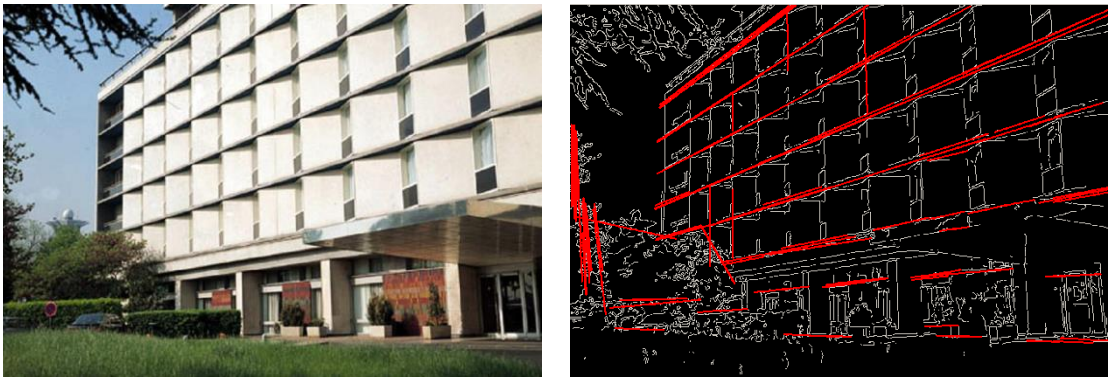
OpenCV is built in a modular structure, which means that the package includes several shared or static libraries. Although it is a C++-based collection of libraries, it handles the memory automatically, which is very similar to those more advanced programming languages and platforms (e.g. Java, Scala, etc.). A rough description of the core modules in OpenCV is listed in Table 6, which can help us understand the basic structure of the entire OpenCV system.

Table 6. Basic Modules in OpenCV System.

Module	Description
• `core`	• A compact module which defines the basic data structures.
• `imgproc`	• An image processing module which includes linear/non-linear image filtering, geometrical image transformations, color space conversion, histograms, and more.
• `video`	• A video analysis module which contains motion estimation, background subtraction, and object tracking algorithms.
• `calib3d`	• Several basic multiple-view geometric algorithms, including single/stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
• `features2d`	• Several salient feature detectors, descriptors, and descriptor matchers.
• `objdetect`	• Detection of objects and instances of the predefined classes.
• `highgui`	• A convenient interface for video capturing, image/video codecs.
• `gpu`	• GPU-accelerated algorithms from different OpenCV modules.

In its feature detection functionality, several algorithms are built in, e.g. Canny algorithm[1], Hough transformation[2], and so on. It detects the features by first looking for the *corners* of the shapes in the image (with background template shapes removed), then followed by probabilistic Hough transformation to find the "frame" of the shapes detected from the image file. Figure 7 demonstrates the result after the feature detection of the input image, which shows a hyperfine granularity and the nice performance of the algorithms built

in the library. The input image and result are taken from the OpenCV API official documentation website.



(a) (b)

Figure 7. Demonstration of OpenCV feature detection algorithms. (a) Input image; (b) Result image.

OpenCV Office Website: http://www.opencv.org
OpenCV API Documentation: http://docs.opencv.org/2.4/index.html

3.1.2. Google Vision

Google Vision API allows users to simply integrate vision detection features, like table detection, explicit content detection, face detection, landmark detection, image analysis, logo detection, and optical character recognition. We have learned some of the functions that Google Vision can provide, which is listed in Table 8.

Google Vision is not completely new to us. Last semester, in COMP-SCI 5551 (FS16) (Advance Software Engineering) class, our project contained a receipt recognition system, which had a very good accuracy. It can scan receipt, outputting the location of purchase, automatically classifying the purchase, and extracting the precise amount in USD.

Admittedly, for text recognition and number recognition, the techniques are quite mature at this time. Rather than just for text and numbers, we really would like to see if Google Vision behaves still very well for general images.

From Table 8, we partitioned Google Vision into several principal modules and give a nice description for each module. Acknowledgeably, those modules are not independent with each other. We separate modules to study based on the "loose coupling and high cohesion" policy. In Table 8, we list the modules mainly based on the dependency graph, which means that the module being studied is dependent on the previously studied module. For example, the "Insight from image" module is further than the "Image analysis" module; the "Sentiment analysis" module is deeper than the "Insight from image module", and so on. In other words, we listed all the modules based on the trend of natural study of ourselves, in order to let it help in our project, or set it as a substitute/replacement of Clarifai API or Tensorflow API in image classification/recognition part.

Table 8. Study of Google Vision API.

Function	Explanation
• Image analysis	• Google Vision API is a REST services API which enables user to have an idea of the image content. It contains machine learning models to classify images into thousands of categories. It includes many different techniques to realize image analysis, such as 2D/3D object recognition, image segmentation, motion detection (e.g. single particle tracking, video tracking, optical flow), medical scan analysis, 3D pose estimation, and automatic number plate recognition. These techniques are being used currently in different fields. Comparing to human abilities of image analysis, these techniques classify images faster and more accurate. Google Vision API classifies the images based on every single detected object in the images, and analyzes the images that uploaded by user or developers in the request.
• Insight from image	• As we know, images analysis technology is a new, developing field in data science and computational area. These techniques are still being improved. Google Vision API, considered as the "pioneer" in this area, can detect sets or categories of the objects in the images broadly. The categories includes (but not limited to) animals, foods, human, tools, cars, machines, natural landscape. Also, it clearly marks the objects in the images with the most possible labels. Situation seems to be the same with every new concept as that Google Vision API improves over time. In my personal point of view, these kinds of techniques are making big changes to the world.
• Inappropriate content detection	• This function is powered by Google Search, which can automatically filter pornography and potentially offensive content. This function is significant, since more than half of the youth are accidentally exposed to pornography nowadays. Dark corners always exist somewhere in the internet world, which is really dangerous to the kids. Those offensive contents must be absolutely stamped out.
• Sentiment analysis on images	• As mentioned above, Google Vision API can analyze images and classify the objects into many categories, this feature also work with people's faces, and it can recognize the emotion on their faces, like joy, anger, sadness, excitement, or trust. Not only it can detect face, but also detect other specific objects, just like product logos, or emoji impressions.
• Text extraction	• This is another very useful function which enables the developers and users to detect the text from the images, and extract them with automatic language identification.

Google Vision Official Website: https://cloud.google.com/vision

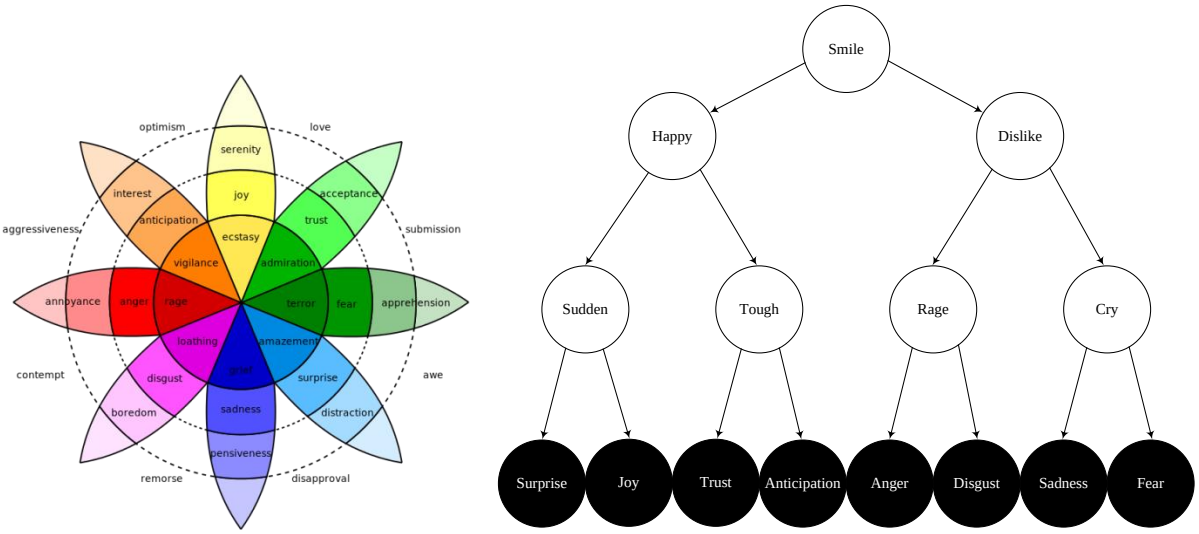
Google Vision API Documentation: https://cloud.google.com/vision/docs

3.2. Literature Reviews

In this section, we are going to present the famous paper of study of emotions of creatures by Plutchik in 1980.

Plutchik, 1980, Emotion: A Psychoevolutionary Synthesis[3].

In 1980, Robert Plutchik constructed a wheel-like diagram of emotions, visualizing 8 basic emotions: *joy*, *trust*, *fear*, *surprise*, *sadness*, *disgust*, *anger* and *anticipation*. In order to complete our project of emoji interpretation, the relations amongst all the emotions are absolutely crucial. Figure 9 (a) shows the exact Plutchik's wheel, which has been delved and studied further and further nowadays, and Figure 9 (b) is our speculation of the decision-tree-structure of the relationships amongst different emotions.



(a) (b)

Figure 9. (a) Plutchik's wheel model of the emotions of creatures; (b) Simplified decision tree model of human's emotions for part of implementation.

In this project, our most difficult job is to find an efficient way to model the Plutchik's wheel of emotions. Our algorithm is currently under development. Nevertheless, we still have some basic idea of the classifying algorithm, which will be mentioned later in this report. In this project, we do want to attempt two different algorithms for building up the machine learning models, the decision tree model (which has been taught during the lecture and the tutorials) and the wheel model, in order to accurately classify the eight basic emotions. This requires us to perform "facet-based" sentiment analysis, whereas currently almost all the sentiment analyzing tools can only complete overall analysis. This is why we do need the decision tree, a level-based graphical structure, to help us divide our work into different levels, the decision at one layer can be made if and only if the decisions at all higher levels had been made. We will first complete the decision tree model, and then using the resulting model to the wheel model, which marks the innovation and distinguished significance of our entire project.

Fortunately, in this iteration, we already found a nice and reasonable way to implement the Plutchick's wheel of emotions of creatures. Definitely, our real model is not a regular octagon, since it is impossible that all the emotions that reside at the corners of the octagon have the same distance of any measurement. The details of the implementation will be articulated in the **Implementation** section.

4. Application Specification

4.1. System Specification

4.1.1. Software Architecture (see Figure 10)

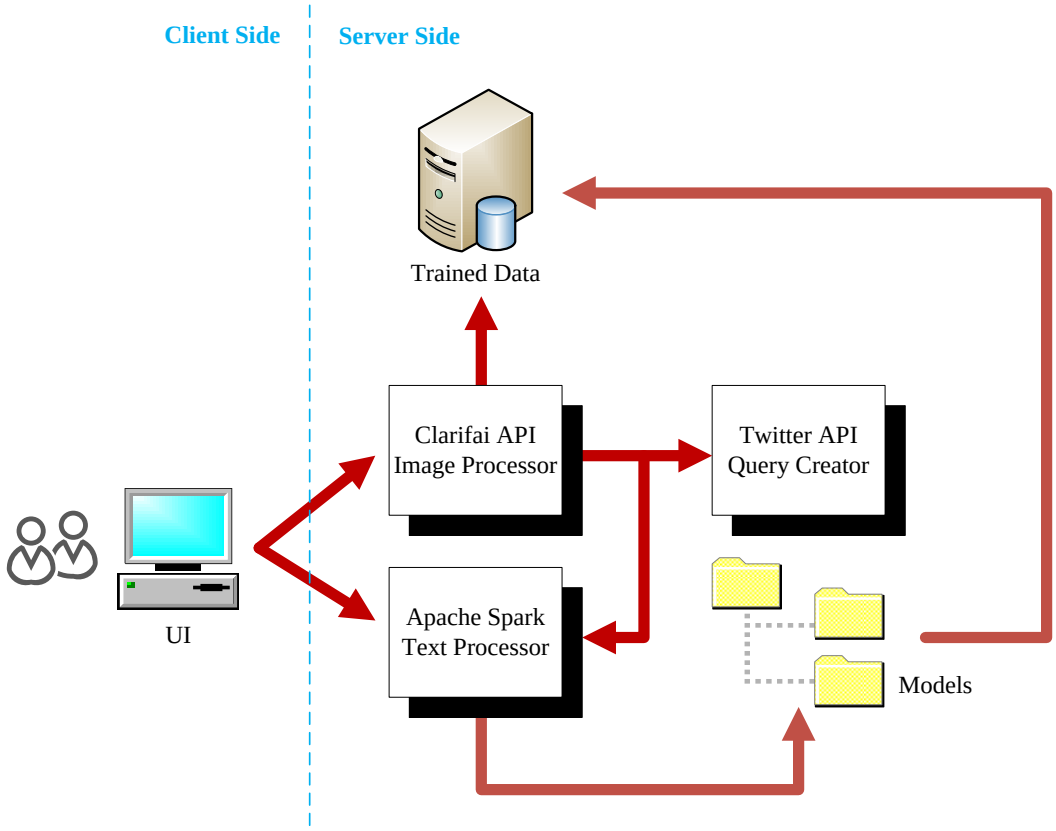


Figure 10. System Architecture of the *Emoji Interpreter* system.

From Figure 10 we can see that our system architecture holds a client-server architectural style. The trained data and generated models are both on the server side. When an input is given by the UI (user), based on the type of the input (text or image), the system passes the input data into its related processors (image or text processor). The two core processors may call each other since sometimes the result of image processor, which is some text, needs further processing via text processor to finally find out the closest classification of the image (emoji impression). After the processing of the input image, the Twitter API is thus called to collect tweet which contains the interpreted meanings of the input emoji. Then, the top 10 tweets are returned to the user for selection. On the other hand, when the text processing is completed, then the result is passed into the created models of people's emotions, in order to find out the emotion class that best fits the input text. In most cases, the resulting text will be determined between two emotions or amongst several emotions. Therefore, top 3 related emotions with the emoji representations will be returned to the user.

4.1.2. Features, Workflow, and Technologies

4.1.2.1. Activity Diagram (see Figure 11)

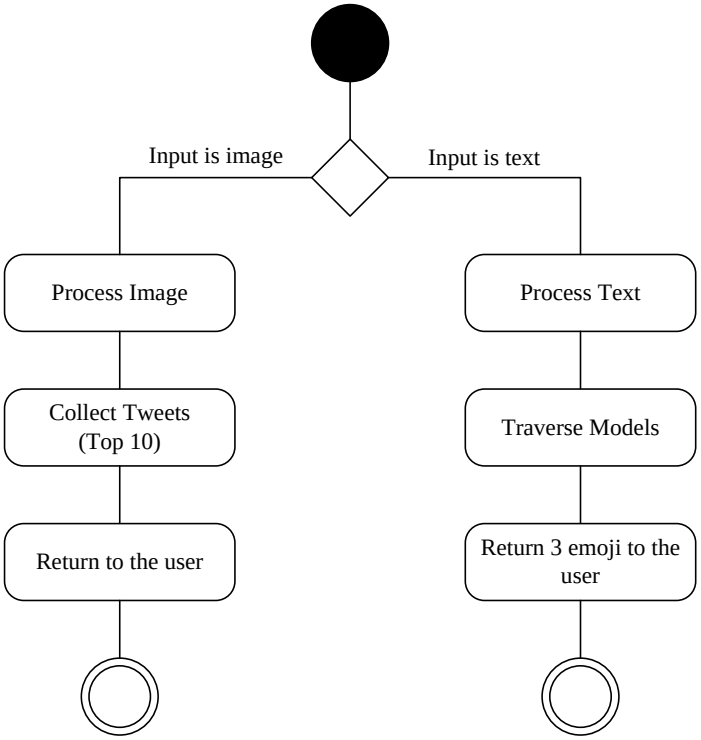


Figure 11. High-level activity diagram of the *Emoji Interpreter* system.

4.1.2.2. Sequence Diagram (see Figure 12)

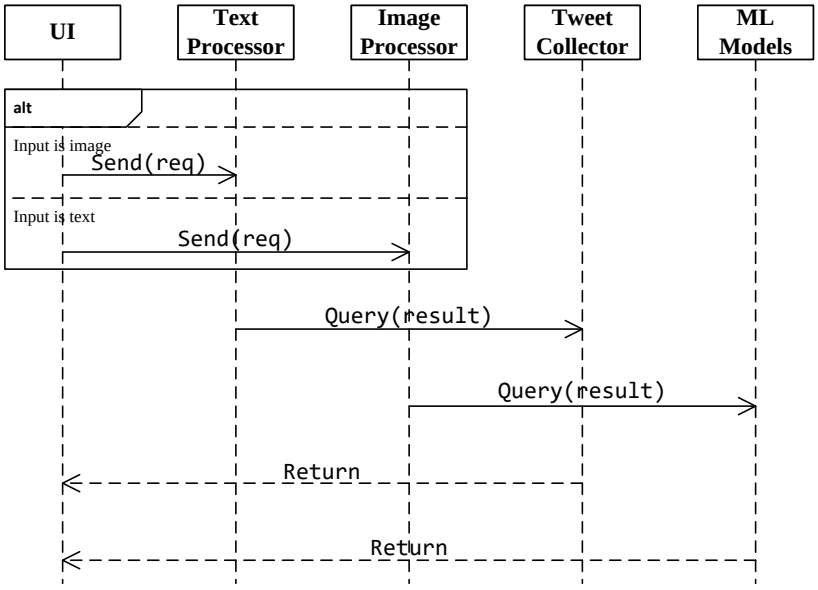


Figure 12. High-level sequence diagram of the *Emoji Interpreter* system.

4.1.2.3. Feature Specification (see Table 13)

Table 13. Feature specifications for the *Emoji Interpreter* system.

Feature	Specification

Text-To-Impression	`input plain_text tx`
	`----------`
	`Spark processor (tx) → sparkResult[]`
	`if sparkResult[] contains class`
	`  return top 3 classes`
	`else`
	`  Word2Vec result (top 3 similar)`
	`----------`
	`output emoji images to the user`

Impression-To-Text	`input jpg image file img`
	`----------`
	`Clarifai processor (img) → textResult[]`
	`if textResult[] contains class`
	`  extract top class cls`
	`else`
	`  Word2Vec result (top similar) cls`
	`Twitter query (cls) → twitterResult[]`
	`twitterResult[].get(10)`
	`----------`
	`output top 10 tweets to the user`

4.1.2.4. Operation Specification (see Table 14)

Table 14. Operation specifications for the *Emoji Interpreter* system.

Input	Output	Exception
• Format: `String`/`File` • File: `jpg` only • File content: emoji • File size: less than 1 M	• Format: `jpg`/`String` • Output: Arrays • Number of results: 10 • UI: Web-based	• Empty input • `null` image file • Size exceeded • Not emoji image

4.1.2.5. Emoji Classification Workflow (see Figure 15)

Emoji in the training part will process through object detection, feature vector and classification algorithm. Then the Classification Model has been built. The testing data has been used for evaluate the model after process of feature vector. In Figure 15, the classification model locates at both the training part and the test part. We set up emoji impressions as the input, as generate key frames using Clarifai API. After that, the feature vectors are extracted from the emoji impressions, which will be passed into the main classification engine. The algorithm of classifying images will generate output models, which serves as the classification model. On the test part, testing data is going to the feature vector extractor. The using the trained models, we can generate our desired results.

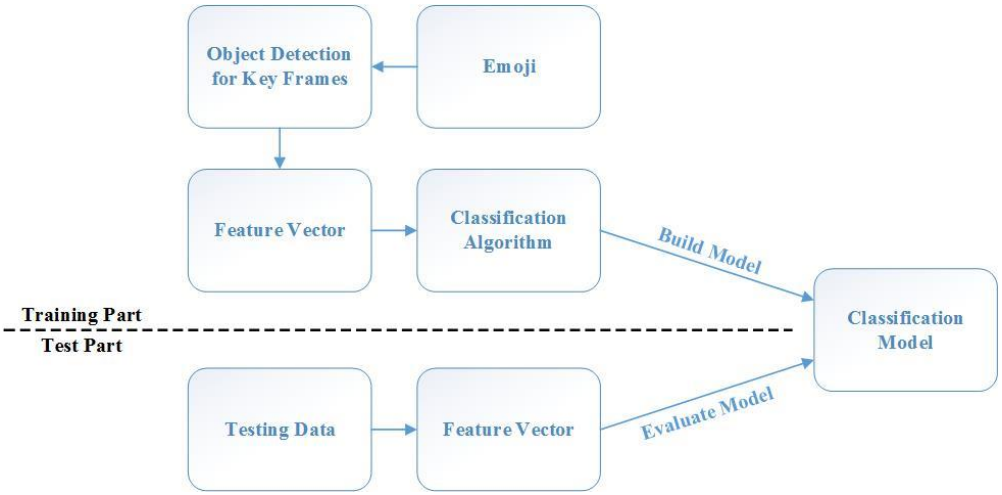


Figure 15. Workflow about the emoji classification ,odel built.

4.2. Existing Applications/Services Used

4.2.1. Apache Spark

Apache Spark is an open source cluster-computing framework. It uses mathematical operation in the memory units instead of putting the operated datasets into the disk. It can analyze and compute the datasets before they have been written into disk. The speed of Apache Spark finishing the process of computing and analyzing is 10 to 100 times faster than Hadoop MapReduce. It allows users load data to cluster-process and inquiry it more than one time, so Apache Spark are very easy and suitable for machine learning.

Apache Spark has 5 main parts. Spark Core (RDDs) is the foundation of the whole project; it offers distributed assignment dispatch, regular dispatch and basic I/O function. Spark SQL offers structured and semi-structured data. Spark Streaming analyzes data uses streaming by the rapid dispatch ability of Spark Core. MLlib is the framework of distribute machine learning. GraphX is a distributed graph processing framework.

Apache Spark is easy to user because it can be used on Java, Scala, Python and so on. It can run on lots of platforms, such as Hadoop, Mesos, standalone, or in the cloud.

In our application, Apache Spark will be used for analyzing emoji, words and sentences, training and testing the machine learning model, finding the matching emoji and emotion sentences with users' input. All the analyzing and processing of images and words are based on Apache Spark, and it can build a model of analyzing emoji and improve it continuously.

Url: http://spark.apache.org

4.2.2. Clarifai API

Clarify API is a self-service API that user can process and analyze images, audios and videos. It is a REST API that can extract important information in the images, audios and videos and it has the basic operation about REST API: `GET`, `PUT`, `POST` and `DELETE`.

The core parts of Clarify API are those models: `bundle`, `metadata`, `tracks` and `insight`. They work together to process the images, audios and videos. It can filter the secondary information and extract most important information. It can remove the background noise in the images and videos, the noise in the audios, then get the useful things and analyze them and return one or more information about them. It also gives a parameter to show the definition about the information, such that higher value means more accuracy for this estimation. For our project, lots of emoji datasets need train and test for the system, the Clarify API can help the system to find the main topic of the emoji, then the topic will be used for training and testing the system. Absolutely, when the model has been built, it can also use to help to finding related emoji based on the input sentences.

Url: http://docs.clarify.io/overview

4.2.3. Twitter API

Twitter API is an API that can connect users, tweets and entities in objects. It is a bridge for users and twitter database. It can be used for various kinds of languages, e.g. Java, Objective C and other programming languages.

Twitter API can be separated to several different smaller APIs: one is REST API and the other one is Stream API. REST API that can help users to get and write Twitter data. It requires developer using OAuth to do it and it will return a JSON file as the output. Without this API, user can use the Twitter Streaming API to develop a real-time application. For the REST API, users can make requests from website or other applications, and then the server will send those requests to the Twitter REST API and Twitter will response the request and return related output through server. For the Twitter Streaming API, user will see the rendered site directly, and then the data is rendered into view at real-time. For the Twitter Streaming API, a stable web connection is necessary all the time.

For our application, Twitter API will be used when the model has been built, at the function that returns emoji based on user's input. Apache Spark has been used to analyze user's emotion based on their input sentences, the most popular emoji matched this emotion will be returned attached with user's input. Another two reserved choices will also be supported so that user can choose the most accurate emotion they want to express. These emoji are chosen based on the most popular emoji to express related emotion on Twitter. This process will be finished before the system build the model and the chosen emoji will be our standard emoji. On the other hand, when users input emoji, we will give some sentences matched their emoji. Clarify API will be used for analyze the emotion in emoji of user's input, then based on the emotion Clarify API extracted, our system will return some matched sentences with the related emotion. At this step, Twitter API has been used to extract the most popular twitter has been post with this emotion. When user input an emoji to express their emotion, the most probable saying they want to post will be show then they can use it directly. Of course, some reserved saying will be given as the same.

Url: https://dev.twitter.com/overview/api

4.2.4. Spark `Word2Vec`

`Word2Vec` contains a set of models which are related, these models can be used to produce word embeddings by rebuild linguistic words contexts with shallow, two-layer neural network. `Word2Vec` is an important in field of NLP algorithm, for it uses the word K dimensionality dense vector to express, the training set is a corpus, including punctuation, pausing by spaces. So it can be regarded as a kind of characteristic processing method.

`Word2Vec` generally is considered to be a deep learning (depth) model, investigate its original cause, may and the author of `Word2Vec` Tomas Mikolov of deep learning background and related `Word2Vec` is a kind of neural network model, but we think the shallow level of the model carefully, also not be strictly is the model of the deep learning. Again, of course, if the upper `Word2Vec` set a layer of output layer related to specific application, such as Softmax, more like a deep model at this time.

The Skip-Gram model: $p(w_i|w_j) = \frac{\exp(u_{w_i}^{\top} v_{w_j})}{\sum_{l=1}^{V} \exp(u_l^{\top} v_{w_j})}$

The Softmax model: $\frac{1}{T} \sum_{t=1}^{T} \sum_{j=-k}^{j=k} \log p(w_{t+j}|w_t)$

Url: https://spark.apache.org/docs/latest/mllib-feature-extraction.html#word2vec

5. Implementation

The algorithm we designed to implement our system of emotions of human beings is called the *facet-based sentiment analysis algorithm*. In our system, an octagon is placed at the very beginning of time. Then, our model defines each corner of the octagon as an extreme emotion (as shown in Figure 16). Besides, each triangular area between two adjacent emotions is labeled as well. The origin is placed at the center of the octagon.

To convincingly represent our machine learning result, we need a united standard of measurements. Therefore, we chose the Spark `Word2Vec` as the measurements of text. First of all, we created a vocabulary of emotions and pass into the system. Based on our "instructional input", the natural language processor of Spark will return a set of results, which are the vectors of words. Based on those vectors, we can build our own distorted octagonal model. Referring to the original Plutchik's model, we can connect those sparse vectors into a distorted octagonal shape. The shape is what we will use in future. Nevertheless, this did not solve the problem yet, because for a real vector, it may have multiple possibilities. For example, the vector may reside outside the shape; or, the vector may reside inside the "overlapping area" (will be discussed more later). In this case, the model itself cannot provide any additional help to our system, which means we have to design more algorithms on top of the distorted octagonal model. In this iteration, we mainly focused on this algorithm and finally found out a controllable way to express it.

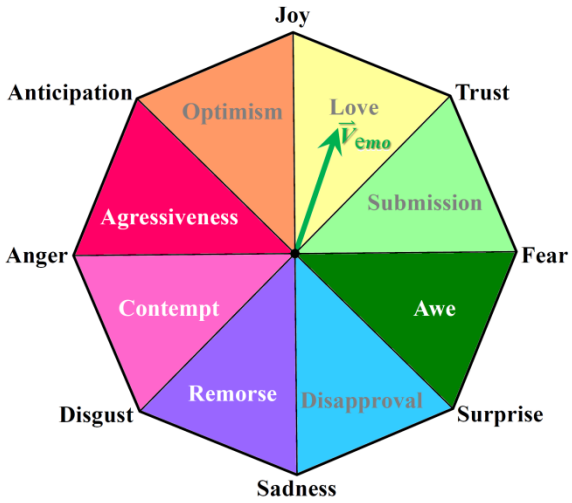


Figure 16. The **octagonal sentiment model** for emoji classification in the *Emoji Interpreter* system. The origin is located at the center of the octagon. Each corner represents an extreme emotion. Each triangular labeled area represents the "combined emotion" which is determined by the two neighboring emotions.

For this model, we should following the steps below to make sure the octagonal model is reasonable (see Table 17). In this iteration, we finished the entire part of this algorithm, which means the text processing part is now complete. Based on this, we can next focus on the image classification/recognition part.

Table 17. General steps of processing the *octagonal machine learning facet-based sentiment model*.

Step	Explanation
Step 1: `Word2Vec` Data Training	In order to make our machine learning model does make sense, we need to first unite the coordinate system, which means we need to choose the same training algorithm. Therefore, we need to know the coordinates of every corner. Thus, we put all the corner extreme emotions into the `Word2Vec` model, and record their coordination values.
Step 2: Lining the range for each triangular area	After receiving all the coordinate values for every corner in the octagon, we can define each range by lining those coordinating data. This marks the completeness of model generation.
Step 3: User input data coordination	For user's text input, or the image processing result (still text), we find out the related vector in the `Word2Vec` system. Definitely, the `Word2Vec` system requires the input data to be vectors. Therefore, we still need some more modules to transform user's input data into the data that our core system can recognize and process.
Step 4: Generate emotion results	Based on which triangular the vector belongs to, we can simply generate the three closest emotions as the classification. For example, in Figure 15, vector $\vec{V}$ is in the "Love" area, the three closest emotions are: "Joy", "Trust", and "Anticipation", which will be considered as the results used for the downstream components.

5.1. Details about emoji to emotion

The parts for our implementation of the discern the emoji and analyze emotion on it is using Clarify and Spark API. Spark API has been used to detect object on images, and analyze them as feature vector. After classification Algorithm, the classification model has been built. Then testing data will be used to get another feature vector to evaluate the classification model. Through training and testing the model will be built and consummated. For our emoji impression system, we use 8 major emotions about humans and each emotion we chose 100 different kinds of emoji to train the system. Those 800 emoji can ensure the system will build an ideal model to analyze the input emoji. Clarify API has been used to discern and annotation for the emoji. With the Clarify annotation output, and based on its probability for the annotation. We can find the matching class for the emoji.

The core algorithm in this process is random forest. As Figure 18 displayed, when an emoji input the system, it will be separate to 10 binary trees. The emoji will find the final result based on binary search. And with the 10 binary trees, 10 results will be given as the output. Then we compare the frequency for all the results and chose the majority result as the output.

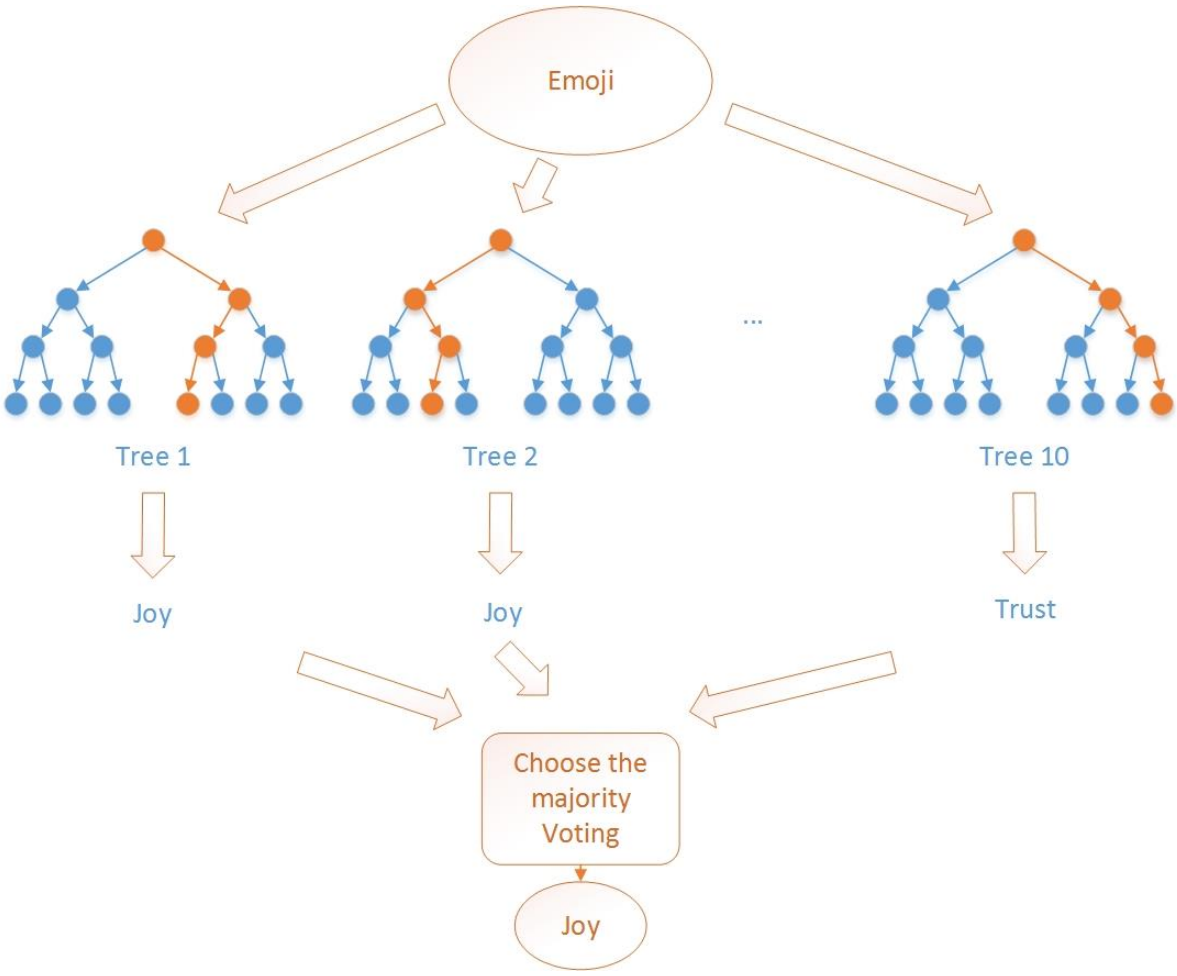


Figure 18. Random forest model process the emoji. Emoji input to the random forest then each tree will do binary search to find the class of the emoji. We have 10 trees in the random forest, then system will return 10 results for the same emoji. Then the system chose the majority results. In this figure, joy has been selected at last.

5.2. Workflow for process of emoji to emotion class

The workflow has been displayed on the Figure 19. At first, we use Spark API to build and training the model, then consummate the model using testing data. The final model will be use to process the input emoji and decide the class of its emotion. Then Clarify API will be used to do the image annotation for the input emoji. Both of the model and annotation will be used in the random forest. And random forest will process these data based on the core algorithm. Final result about the emoji will be give after this process.

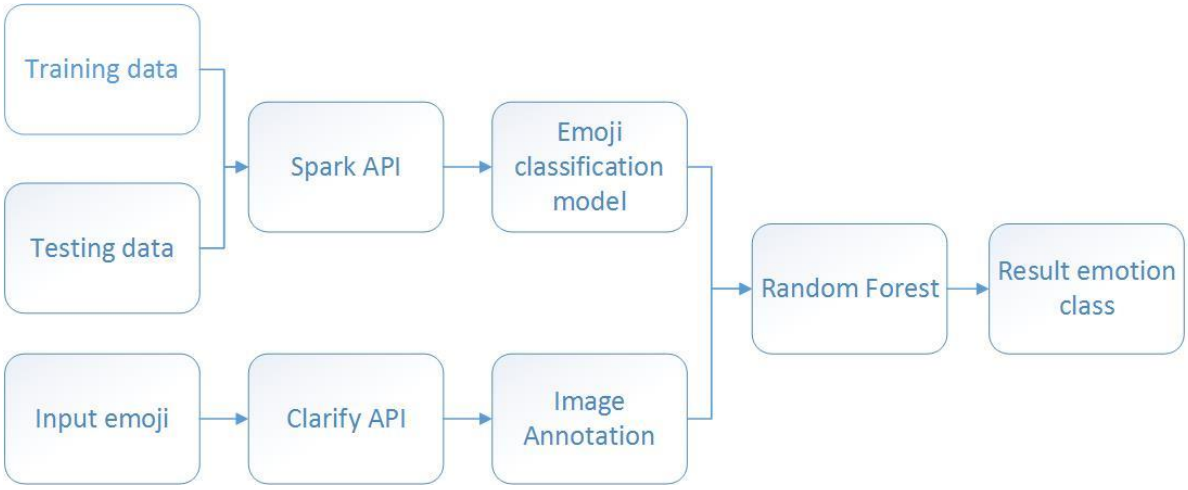


Figure 19. Workflow of process of emoji to emotion class. Training data and testing data will be used for the Spark API then get the emoji classification model. Input emoji will process by clarify API then get image annotation. Both of them will be used for the random forest then get the final result emotion class.

5.3. Actual Distorted Octagonal Model

Figure 16 demonstrates a "conceptual" model of emotions of creatures, which is , however, kind of "ideal", because it overlooked the measurements of how different amongst those emotions. Also, for those "triangular" areas define by the center and two adjacent emotion points, none of the shapes overlap, which is absolutely not practical at all.

Therefore, in order to fully implement/simulate the Plutchik's wheel of emotions of creatures, we used Apache Spark `Word2Vec` module as the measurement to weigh each emotion. In addition of the basic eight emotions, we added more emotions and classified each group of emotions as "mild", "regular", and "extreme". Table 20 lists our input file for Spark `Word2Vec`.

In this project, definitely, we cannot let Spark to give us nice results if our input vocabulary size is such small. Therefore, we also input the original paper of Plutchik of emotions of creatures into the text processor as well. Please note that Spark `Word2Vec` takes a "corpus" as the input, not just the words or vocabularies of our intuitive thinking. In future we will pass more data into the Spark `Word2Vec` to make our corpus bigger and bigger. And our results are expected to be more precise, or more fine.

Table 20. Content of the emotion input file.

Mild	Regular	Extreme
serenity	joy	ecstasy
acceptance	trust	admiration
apprehension	fear	terror
distraction	surprise	amazement
pensiveness	sadness	grief
boredom	disgust	loathing
annoyance	anger	rage
interest	anticipation	vigilance

The Spark `Word2Vec` result is listed in below.

```
-------------------------------------------------------------------
pensiveness: (0.084811315, -0.076939434)
serenity: (-0.22185317, 0.1769593)
sadness: (-0.095178716, -0.04110184)
joy: (0.062327653, 0.07952505)
ecstasy: (0.0879568, -0.1424837)
grief: (-0.047081765, 0.16720007)
acceptance: (-0.14575848, 0.1447692)
boredom: (0.24667054, -0.06456961)
trust: (0.1051123, 0.03959592)
disgust: (0.08175722, -0.16723962)
admiration: (0.04882711, -0.15296534)
loathing: (-0.15054098, 0.11843752)
annoyance: (-0.10143862, 0.036245566)
apprehension: (0.057190824, -0.08134955)
fear: (-0.14003055, 0.1244879)
anger: (-0.085408226, 0.0037039549)
rage: (0.12575932, -0.22855853)
terror: (0.03308146, 0.002353456)
distraction: (0.21391006, 0.052723654)
interest: (0.13087626, -0.010330238)
anticipation: (-0.16618755, -0.2013035)
surprise: (0.028713182, -0.15810779)
amazement: (-0.17262912, 0.15352404)
vigilance: (0.12552573, -0.18793426)
-------------------------------------------------------------------
```

Based on the vectorized coordinates, we drew our distorted octagonal model (see Figure 21).

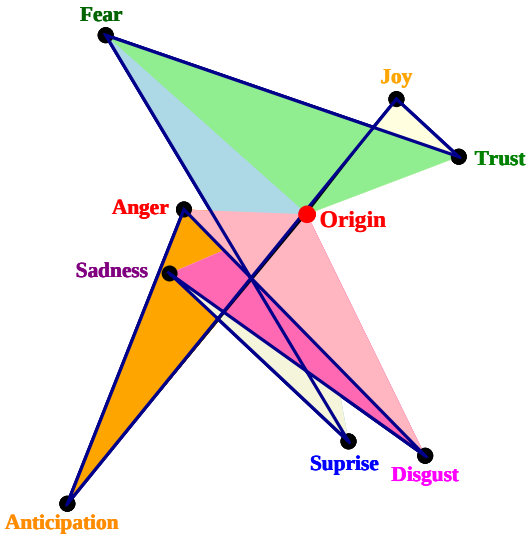


Figure 21. Distorted octagonal model of creature emotions (refer to the Plutchick's wheel of emotions). Black dots represent the vertices in the original regular octagon. Black solid lines represent the edges in the original regular octagon. Colored triangular areas represent the sectors defined in the original regular octagon.

From the distorted octagonal model, we can see that the triangular areas actually overlap, which means for a real vector, it may reside in multiple emotion areas, or does not reside in any triangular area. Therefore, we should design another algorithm to deal with real vectors.

5.4. Two ways for further text training

There are two possible ways of further algorithm to deal with real vectors. The first one is called "further text training", which means for each overlapping area, we need more data that reside in the area for us to do further training. The goal of the training is to determine the probability of the emotion belongs to each triangle in the overlapping area. This learning step can dramatically help improve the accuracy of the entire machine learning.

However, since the Spark `Word2Vec` is already a machine learning step, finding a suitable sizable dataset based the previous machine learning step is very difficult, even impossible. Therefore, we had to abandon this way of training and apply the second way to do it, the *multi-dimensional* learning.

If a real vector resides in an overlapping area, then all the participating emotions are collected into a "multi-dimensional container". Then, results are generated based on each participating emotion independently. We admitted that those emotions are correlated with each other and not kind of independent, but we just assumed that they are independent with each other for simplicity. After that, we can generate multiple answers from the system. What we do next is to select the "consensus" answer from those possible answers. We assumed that the "consensus" answer that appeared the most amount of times amongst all the answers is the reasonable answer to the input, no matter is was text input, or image input. Admittedly, this approach required more time and space complexity at real time. But compared with the first method, we think this method is controllable and convenient to implement and present.

6. Documentation

6.1. User Interface Mock-Up (see Figure 22)

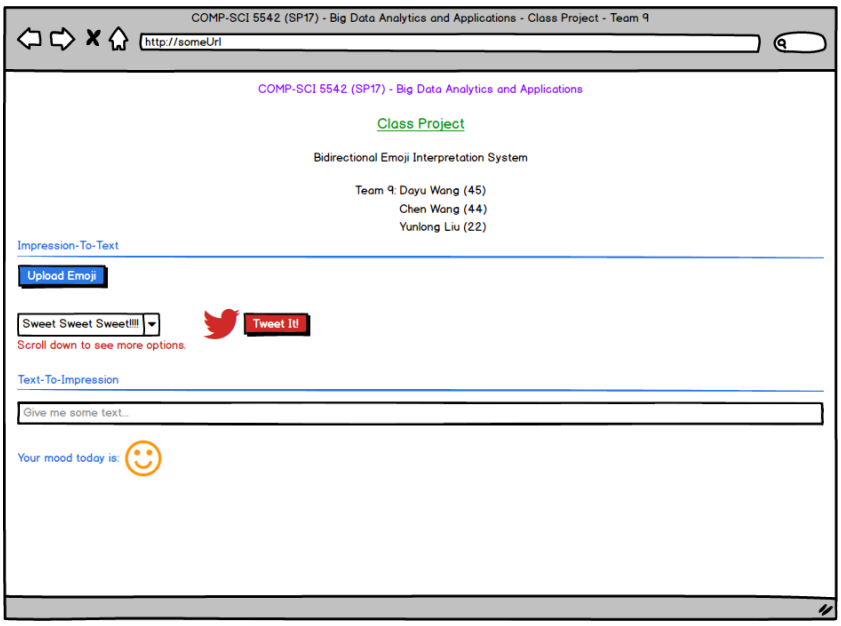


Figure 22. User interface mock-up for the *Emoji Interpreter* system.

6.2. Standard Emoji Impressions for the Eight Corners in Pluchik's Wheel (see Figure 23)

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 23. Standard emoji impressions for the eight corners in Pluchik's wheel in the *Emoji Interpreter* system. (a) Anger; (b) Anticipation; (c) Disgust; (d) Fear; (e) Joy; (f) Sadness; (g) Surprise; (h) Trust.

6.3. Output of emoji classification model (confusion matrix)

```
========== Confusion Matrix ==========
0.0  0.0  0.0  0.0  2.0  0.0  0.0  0.0
0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0
0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0
0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0
1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
0.0  0.0  0.0  2.0  0.0  0.0  0.0  0.0
Accuracy: 0.1875
```

6.4. Screenshots of Actual User Interface (see Figure 24)

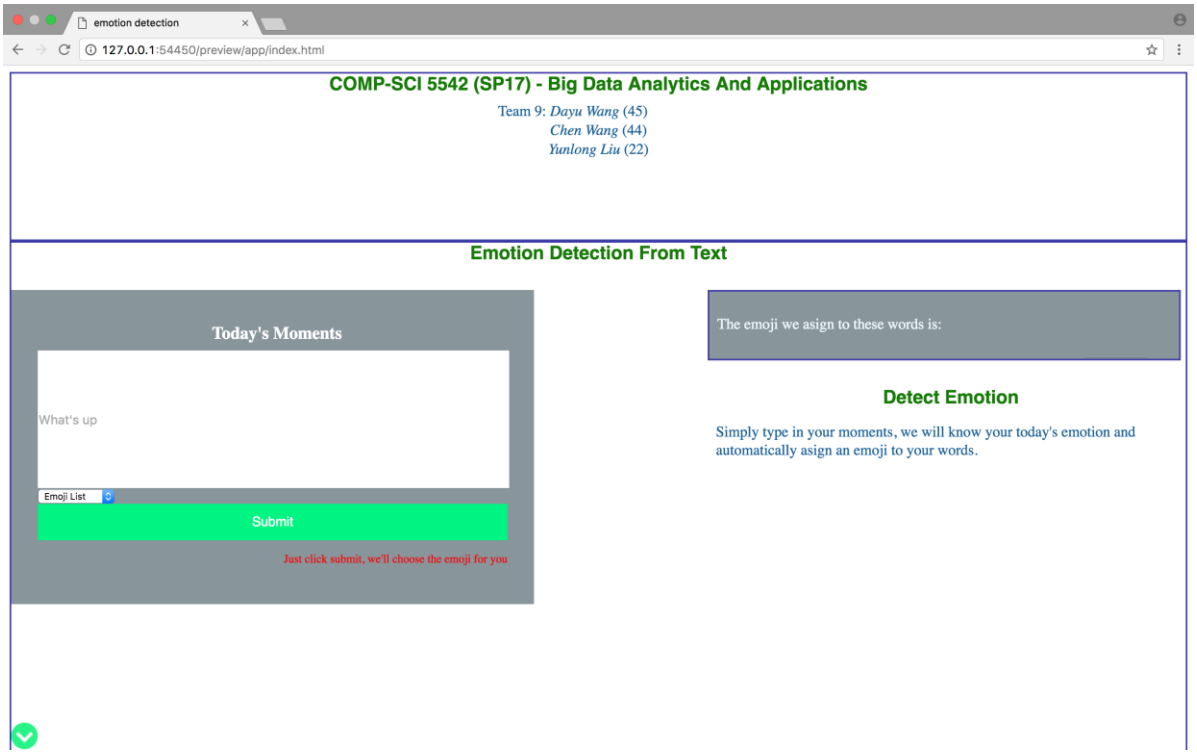


Figure 24. (a) User interface of "text-to-impression".

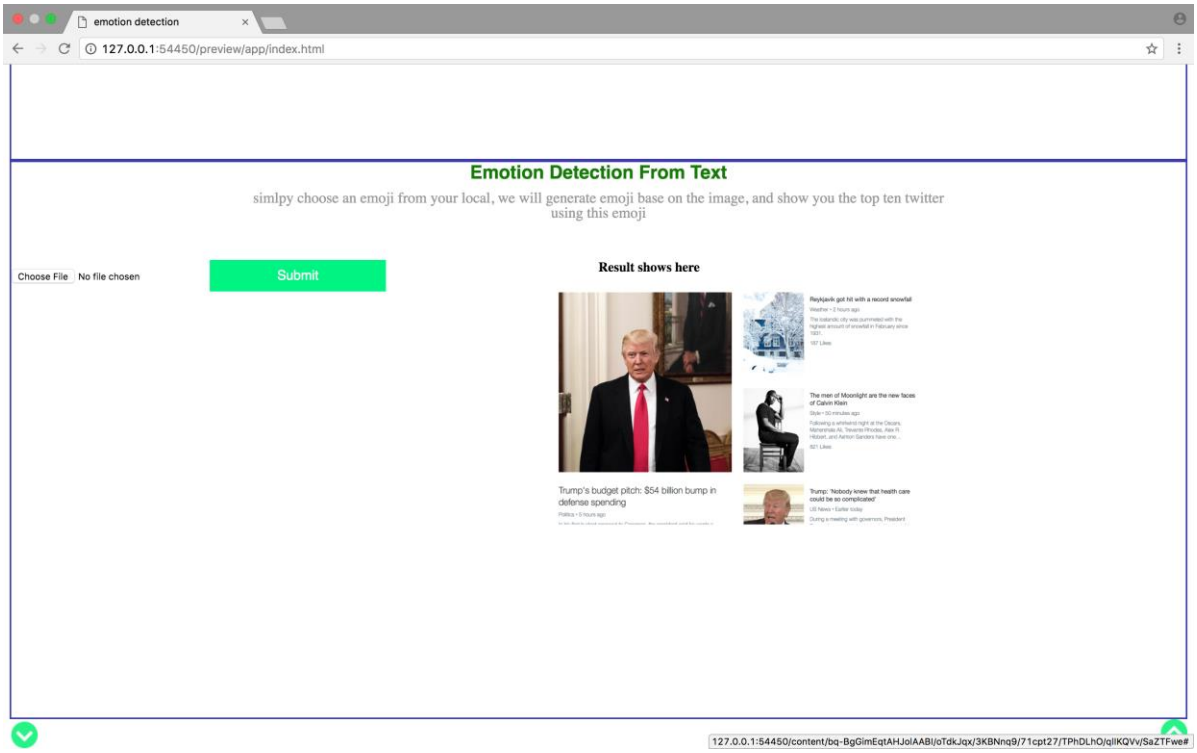


Figure 24. (b) User interface of "impression-to-text".

7. Project Management

7.1. Work Completed (see Table 25)

Table 25. COMP-SCI 5542 (SP17) Project Team 9 - Project Report 2 - Work Completed.

Name (Class ID)	Source Code Completed (Hours)	Report Completed (Hours)	Other Documentation Completed (Hours)
***Dayu Wang* (45)**	• Apache Spark Natural Language Processing Engines (10)	• Project Objectives (1) • Approach (1) • Related Work (1) • App Specification (2) • Implementation (5) • Documentation (1) • Management (1)	• Architecture Diagram (2) • Activity Diagram (1) • Sequence Diagram (1) • **Octagon Design** (5) • **Real Distorted Octagonal Model** (5)
***Chen Wang* (44)**	• Apache Spark and Tensorflow Image Processing Engines (10)	• App Specification (4) • Documentation (2) • Implementation (3)	• Emoji Image Handling (3) • Image Workflows (3) • Image ML Model (3)
***Yunlong Liu* (22)**	• User Interface (5)	• Related Work (3) • Documentation (3)	• User Interface Management (2)

7.2. Work to be Completed (see Table 26)

Table 26. COMP-SCI 5542 (SP17) Project Team 9 - Project Report 2 - Work to be Completed.

Name (Class ID)	Work Description	Estimated Hours	Deadline
***Dayu Wang* (45)**	• Complete the entire algorithm of the application. • Complete the entire algorithm of the training and researching algorithm. • Prepare the whole presentation of the entire project.	• 8 • 8 • 8	• 04/03/17 • 04/10/17 • 04/17/17
***Chen Wang* (44)**	• Finish the pipeline of image processing. • Document the image training model clearly and reasonably. • Collect all training results and research results (e.g. Spark *versus* Tensorflow).	• 8 • 8 • 8	• 04/03/17 • 04/10/17 • 04/17/17
***Yunlong Liu* (22)**	• Implement using Twitter API. • Connect all parts to the final application package. • Design more possible interactions.	• 8 • 8 • 8	• 04/03/17 • 04/10/17 • 04/17/17

7.3. Issues/Concerns: There is **no issue discovered** in this iteration.

8. References

[1] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6), 679-698.

[2] Matas, J., Galambos, C., & Kittler, J. (2000). Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding*, 78(1), 119-137.

[3] Plutchik, R. (1980). Emotion: A Psychoevolutionary Synthesis. Harpercollins College Division.