Note: I was thinking of doing a table referencing the main models and their performance results. It would fil up some space and look professional.

# Which Emoji Talks Best for My Picture?

They used a list of tweets which contained an image, an emoji, and some text. All these three features were important for their dataset.

This paper was more focused on predicting emojis based on the image rather than the text.

# Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm

Pre-processing:

* Texts are tokenised.
* Double letters in words which should not contain double letters are replaced by the normal word e.g. loool, looooool, loooooooooooool all mean lol.
* url, mentions and numbers are all taken to be a <special token>

There model is called DeepMoji and it is a Long Short-Term Memory (LSTM) model. It uses an embedding layer of 256 dimensions to project each word into a vector space. A hyperbolic tangent activation function is used to enforce a constraint of each embedding dimension being within [−1, 1]. Use two bidirectional LSTM layers with 1024 (512 in each direction) hidden units in each to capture the context of each word. Finally, an attention layer that take all of these layers as input using skip-connections is used.

Gejja bicca copy past ghax important:

The attention mechanism lets the model decide the importance of each word for the prediction task by weighing them when constructing the representation of the text. For instance, a word such as ‘amazing’ is likely to be very informative of the emotional meaning of a text and it should thus be treated accordingly. We use a simple approach inspired by (Bahdanau et al., 2014; Yang et al., 2016) with a single parameter pr. input channel:

Here ht is the representation of the word at time step t and wa is the weight matrix for the attention layer. The attention importance scores for each time step, at , are obtained by multiplying the representations with the weight matrix and then normalizing to construct a probability distribution over the words. Lastly, the representation vector for the text, v, is found by a weighted summation over all the time steps using the attention importance scores as weights. This representation vector obtained from the attention layer is a high-level encoding of the entire text, which is used as input to the final Softmax layer for classification. We find that adding the attention mechanism and skipconnections improves the model’s capabilities for transfer learning.

The regularization used for pretraining task is a L2 regularization of 1E−6 on the embedding weights.

Theano was used to develop the model.

They use a transfer of learning approach which they call ‘chain-thaw’. First the new layers are fine-tuned whilst the other layers are frozen (do not change), then starting from the first layer, the remainder of the layers are fine-tuned individually whilst the others are frozen and finally all the layers are fine-tuned together.

Using 1024 dimensions they achieved an accuracy of 43.8%.

Predictions agreed with 82.4% of human annotations.

Github link available

# SVMs perform better than RNNs at Emoji Prediction

Pre-processing:

* case normalisation.
* discard low frequency features.

Best results using multi-class (one-vs-rest) linear support vector machine. Bag of n-grams was used as features, weighted by sub-linear TF-IDF. Logistic regression and random forest were tried but achieved inferior results than the SVM. They used scikit-learn and libliniear for coding this model. The hyperparameters considered during optimization were maximum character/word n-gram size, case normalization, minimum document frequency threshold for excluding low-frequency features, and SVM margin (or regularization) parameter ‘C’ (used maximum character n-grams size of 6, maximum word n-gram size of 4, minimum document frequency threshold of 2, SVM parameter C of 0.10). Results: 36.55 precision, 36.22 recall and 35.99 F1-score.

Also implemented an RNN. It consisted of two bidirectional components, one for words and one for characters. The recurrent components of the network builds two representations for the text (one based on characters, the other based on words), the representations are concatenated and passed to a fully connected softmax layer that assigns an emoji to the document based on the RNN representations. Embedding layers were used before the RNN layers. Used Tensorflow and Keras to implement it. Optimized the hyperparameters of the architecture through a random search for the embedding size of both characters and words, the hidden representation size of the RNN cells, the dropout parameter for each component of the network, frequency threshold for excluding features, RNN architecture, GRU or LSTM, and case normalization. Picked the epoch with the best F1-measure for each hyperparameter setting. For RNN, used a model with embedding layers of size 32 (for characters) and 128 (for words). For bidirectional GRU networks, used hidden units of sizes 32 and 128 for character and word input, respectively, minimum frequency threshold of 4 for characters and 1 for words, dropout parameter of 0.50 at the embedding layers and 0.10 at the RNN layers, and no case normalization.

# Semi-Supervised Recognition of Sarcastic Sentences in Twitter and Amazon

**I added this paper because I think that sarcasm should be accounted for in the task.**

Utilize the semi-supervised sarcasm identification algorithm (SASI) of (Tsur et al., 2010). The algorithm employs two modules: semi supervised pattern acquisition for identifying sarcastic patterns that serve as features for a classifier, and a classification stage that classifies each sentence to a sarcastic class.

# A Method for Automatically Generating the Emotional Vectors of Emoticons Using Weblog Articles

Aim of the paper is to evaluate the emotions which are expressed by emojis in a piece of text.

It was assumed that the emoticons in the text expressed the emotions of the text itself.

They use a 14-dimension vector to represent 14 different emotions. joy, trust, fear, surprise, sadness, disgust, anger, anticipation, love, awe, disapproval, remorse, contempt, and optimism.

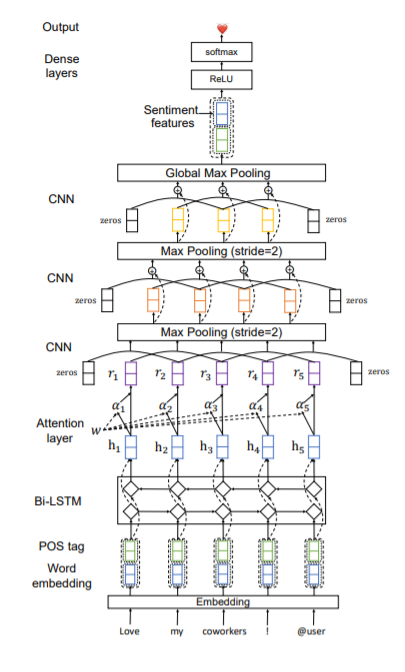
288 emotional words were selected and assigned to one of the 14 emotions. Then, their count in the text documents was found.

Methodology (copy paste):

1. Collected a large volume of weblog articles and extracted only those sentences with emoticons.
2. Determined which sentences with emoticons also contained emotional words, and extracted those sentences.
3. Counted the co-occurrence of emoticons and emotional words in the extracted sentences. For example, in the sentence “I’m mad at myself <anger emoji>,” the emotional word “mad” corresponds to the emotion “anger,” so we increase the frequency of the emotion “anger” for the emoticon <anger emoji>.
4. Using the same process, we tabulated the frequency for all of the extracted sentences.
5. Normalizing the component values of the vectors so that they added up to a value of one, we prepared 14 dimensional emotional vectors.

Some emojis were found to be used in different types of emotions.

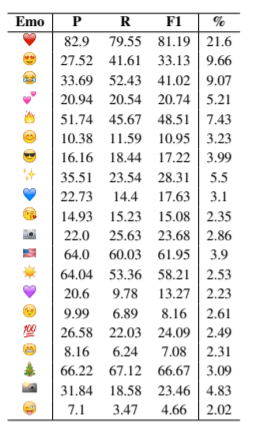
# Residual CNN-LSTM Network with Attention for English Emoji Prediction

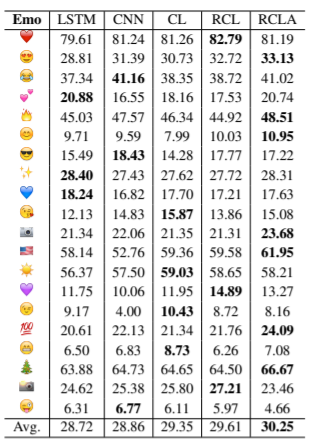


They use a Residual CNN-LSTM with attention (RCLA) model.

Layers/features:

1. Embedding layer - used to convert a sentence from a sequence of words into a sequence of dense vectors. An embedding lookup table is used.
2. POS tags – combined with the word embeddings to form the final word features. Ark-Tweet-NLP tool is used to extract the POS tags.
3. bidirectional long short-term memory (Bi-LSTM) layer – used to capture long-range contextual information from tweets. At time step i, a hidden state hi is generated which contains both previous and future context information.
4. Attention layer – to help model focus on important words and contexts since different words and phrases have different importance for emoji prediction. The attentional weights are calculated using formulas very similar to those mentioned above. Mhux ha noqghod niktibhom ghax hassle imma jekk nigu bzonnhom qieghdin fil paper.
5. 3-layer CNN – to capture local context information. Each CNN layer has multiple kernels with different window sizes. Residual connections are applied to the CNN layers. Max pooling is applied to the output of the last CNN layer to obtain the hidden representation of tweets.
6. Sentiment features – extracted using AffectiveTweets package in Weka. TweetToLexiconFeatureVector and TweetToSentiStrengthFeatureVector are used as filters.
7. Softmax layer – used to predict the emoji label.

Results: 

F-score of different methods: 

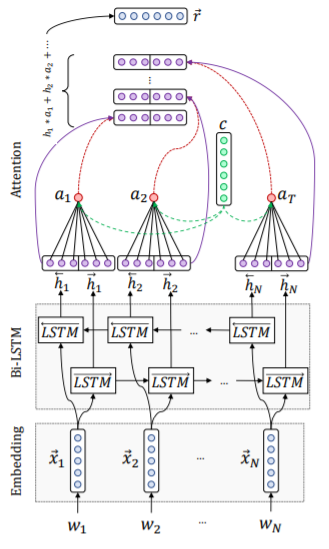
# Predicting Emojis using RNNs with Context-aware Attention

Took an approach which consisted of three main steps, the text processing step, the word embeddings pre-training step, and the model training step.

For word embeddings, they use the word2vec algorithm, with the skip-gram model, negative sampling of 5 and minimum word count of 20, utilizing the Gensim’s implementation. The pre-trained word embeddings are used for initializing the first layer of the neural networks.

Pre-processing:

* Use ekphrasis tool. Consists of tokenization, spelling correction, word normalisation, word segmentation and word annotation.



Model:

* use a word-level BiLSTM architecture with an attention mechanism.
* Embedding layer – project the input text to a low-dimension vector space. Weights are initialised with the pre-training word embeddings.
* BiLSTM layer – bidirectional is used to get word annotations that summarise the information in both directions, one direction is from weight1 … weightN, and the other direction is from weightN … weight1. The final annotation of each word is found y concatenating the annotations found in both directions.
* Attention layer – used to better estimate the importance of each word based on the context. Formulas on paper.

# Predicting Emojis using Hierarchical Attention Neural Networks and Support Vector Machine

Pre-processing:

* Text emojis are changed to unique strings e.g. :) is changed to \_ \_ smile \_ \_.
* Remove mentions, URLs, location mentions, and non-letter characters except for #.
* Tokenisation, including hashtag splitting.
* Converted to lowercase, lemmatisation, and removing of frequent words.

Features (in my opinion some are stupid):

* Text features – extracting information from the text including number of words, hashtags, stop-words, user mentions, mean word length and more. Punctuation such as question marks, exclamation marks or words with all title letters could signify an intensified face emotion.
* Semantic features – capturing location tags.
* Emotion-related features – used the NRC Word-Emotion Association Lexicon to capture emotion.
* Colour-related features – to deal with heart emojis of different colours.
* Sentiment features – SentiWordNet is used to associate each token in the tweet with a positive and negative score.
* Twitter clusters – used Hierarchical Twitter Word Clusters to group words which are mis spelled or are written with a different spelling but mean the same word.

Classifiers:

* Different classifiers are used: linear, non-linear, stacking, and deep learning.
* Linear – used Multinomial Naïve Bayes as baseline. Also used Logistic Regression with L-BFGS optimizer and Logistic Regression with L-BFGS optimizer.
* Non-linear – used Random Forest, and AdaBoost with Decision Tree base, both with 300 estimators.
* Stacking – combining count based features and semantic features. One approach was using SVM (tf-idf), AdaBoost (embeddings) and Random Forest (semantic and sentiment extracted features). The second approach was SVM (tf-idf), AdaBoost (embeddings) and Multi-layer Perceptron (tf-idf).
* Deep Learning – used Multi-layer Perceptrons, Recurrent NNs with LSTM, and CNNs.
  + best results using Hierarchical Attention Neural Network – two levels of attention mechanism: for word and for sentence. Structure comprises of word sequence encoder, word-level attention layer, sentence encoder and a sentence-level attention layer.
  + Another approach was the two-layered bidirectional LSTM with a dropout rate of 0.35 and the Adam optimizer.
  + Another approach is the CNN. Allows network to learn and capture patterns for adjacent words in sentences. Can learn and track correlations between close words and inputs.
* The key insight of boosting our Neural Network models was switching from ReLU to ELU as activation function. Proper Dropout Strategy (between 0.35 and 0.4) also improved our validation score.

Multilingual Emoji Prediction

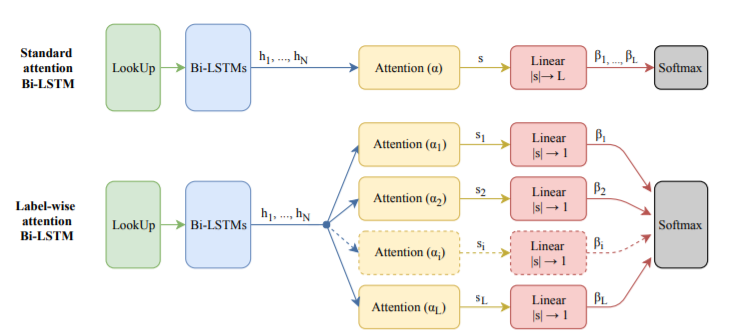
This is the paper of the entire competition. It describes the main participants which were better than the baseline. All of the participant papers are mentioned in the paper and results are shown. Good to compare different models and their results.

# Interpretable Emoji Prediction via Label-Wise Attention LSTMs

The base architecture is the DeepMoji model.

Based on two stacked word-based bi-directional LSTM recurrent neural networks with skip connections between the first and the second LSTM. The model also includes an attention module to increase its sensitivity to individual words during prediction.

The main architectural difference with respect to the typical attention:



Attention is calculated sing formulas very similar to those mentioned previously.

The final prediction is done using the formulas:

# Exploring Emoji Usage and Prediction Through a Temporal Variation Lens

This paper can be referenced to show that some emojis can be better predicted if the time in which their accompanying text was taken into account. Since our dataset only consists of 20 basic emojis, this should not be a requirement. This approach should is more to be considered when a large number of emojis are used. For example, a clover is used more during st. patricks day or a Christmas tree emoji is used more in the Christmas period. This paper can be mentioned in the literature review as a general note on predicting emojis.

# English Emoji Prediction with Gradient Boosting Regression Tree Method and Bidirectional LSTM

The authors implement two models but the primary focus is on GBM.

Pre-processing

* Tokenization using tweetokenize (GitHub link given).
* normalize tweets by replacing all the URLs with “\_url\_” and all the mentions with ”@mention”.

Features

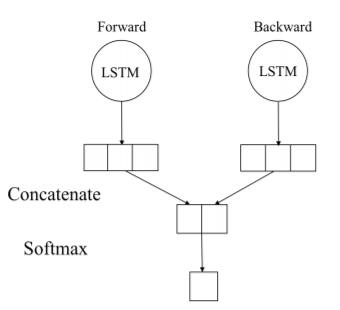
* BLSTM only makes use of pre-trained GloVe as word embedding.
* GBM makes use of the following features:
  + Character ngram – represents the presence or absence of contiguous sequence of 3, 4 and 5 characters to capture the morphological information hidden in the words.
  + POS tags – presents the information about the lexical type of the word. Makes use of Carnegie Mellon University (CMU) tool.
  + Cluster – induced from CMU pos-tagging tool which provides the word cluster using the Brown clustering algorithm.
  + Negation – a negated context is defined from a negation word to one of the punctuation marks:”,”, “.”, “:”, “:”, “!”, “?”. Each word in the negated context is added with the suffix “\_NEG”.
  + Word Ngram – word ngram for n=1, 2, 3, 4.
  + Counting Features – combining all the number of special symbols in each tweet. This includes the number of hashtags, the number of words with all caps, the number of contiguous punctuation, whether the last token contains a question mark or and exclamation mark, the number of mentions, the number of URLs, the number of words with repeated characters, and presence or absence of positive or negative emoticons in the tweet.
  + Sentiment-Specific Word Embedding (SSWE) Feature – can capture the sentiment information of sentences as well as the syntactic context of words.
  + Lexicon Feature – number of sentiment words, the total sentiment score, the score of last sentiment words and the maximal sentiment score for each lexicon. They make use of many different ready-made lexicons.

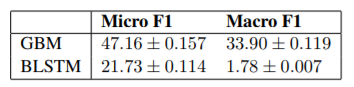
Model

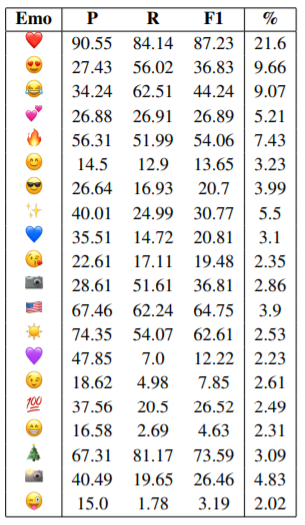
GBM

* Feature vector containing all the above features is used as input.
* method generates base models sequentially and at each step updates the base model by minimizing the loss function value.
* The base model is a single regression tree which fits a set of features by partitioning the feature space into different regions.
* Use lightGBM to build the model.
* Tune the hyperparameters on the training set by grid search.

BLSTM

* Use Stanford’s GloVe embedding as a word embedding.
* Architecture used: 

Comparison between the two methods: 

Results of GBM: 

# Emojis: Too many Choices?

For a Baseline they used a Bag of Words model with a Bernoulli Naive Bayes Classifier. They implemented a Most Frequent Class Model and a Random Model to help draw insights from the baseline. For this model they used NTK for pre-processing and scikit-learn framework for the model training. As a structure, they did Tokenization, Bagination, Tf-idf transform and Training. For tokenization they used NLTK’s TweetTokenizer function, configured to convert all text to lowercase, crop repeating characters to a max of 3 and remove tweeter handles. The BOW was created by making a set of the tokens which appeared per the document. Next they normalized the frequency data into termfrequency times inverse document-frequency(tfidf) representation. Finally, they trained a Bernoulli Naive Bayes classifier on this data.

They also implemented 4 types of neural networks, which did not perform much better than the baseline. (they provided no detail on how they implemented these networks except for the number of layers which they used, they just referenced the paper which they were inspired from). The implementations were LSTM (128 layers), BLSTM (128 layers but bidirectional), CNN-LSTM and CNN-BLSTM.

They also implemented a linear SVM using a one-vs-rest approach on the bag of words. No detail was provided.

# Emoji Recommendation in Private Instant Messages

Textual features:

* bags of words or bags of characters
* total word count
* exclamation and interrogation marks
* n-grams (up to 5-grams).

Sentiment-related features:

* positive, negative, and neutral polarity scores from SentiStrength8 using the available model trained on MySpace comments and tweets, and from Echo9.

Token representation:

* Tokens can either be count vectors of words or characters gathered, then transformed using TF-IDF weighting scheme. Bags of characters can be really useful to deal with spelling variations and slang words without the need of a knowledge based or external lexicon.

ML-RandomForest algorithm from scikit-learn was used to predict emojis. Used 20 trees with no depth limitation. Each model has been trained with the following methodology:

1. Preprocessing (tf-idf vectorization without stop words, and feature computation)
2. Cross validation (10 folds)
3. Classifier overall and per label evaluation.

# emoji2vec: Learning Emoji Representations from their Description

Used to create word embeddings but considers emojis as well. Created because word2vecembeddings do not cater for emojis in the phrases. Github link is provided for reproducibility. Probably not to be mentioned in literature review… it is more to see if we can use this in order to create word embeddings.

# CoNFET: AN ENGLISH SENTENCE TO EMOJIS TRANSLATION ALGORITHM

Converts a sentence into a series of emojis which best represent the sentence.

3 steps:

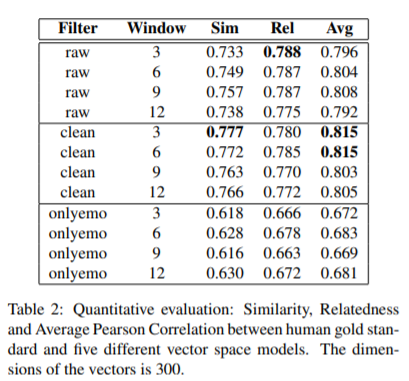
1. The input sentence is split into its constituent n-grams either in an exhaustive manner or using dependency relations.
2. The n-grams of the sentence are translated into emojis using the nearest neighbor in a vectorized linguistic space.
3. The translations are scored using either a simple average or an average weighted by the Term Frequency-Inverse Document Frequency (TF-IDF) score of the n-gram. The sequence with the highest score is chosen.

More detail on each step if needed in paper, but not exactly what we want since the technique does require a substantial amount of emojis, but it can be tried if need be.

# What does this Emoji Mean? A Vector Space Skip-Gram Model for Twitter Emojis

CMU Tweet Twokenizer to pre-process the tweets. Also removed stopwords, punctuation marks, hashtags, and user mentions. Words were lowercased to reduce noise.

Used skip-gram neural embedding model introduced by Mikolov et al. Different skipgram models were trained in order to find the best parameter configuration. The model with 300 dimensions achieved the best results. Used window size of between 3 and 12 tokens. Three different filters were tried in order to find which produced the best results. The first filter (“raw”) removed links and mentions only. The second filter (“clean”) removed links, mentions, punctuation, and stopwords. The third filter (“only emojis”) removes everything but the emoji.



The link to the models is given in the paper.

# Are Emojis Predictable?

Exactly the task which we need to do. The authors are the organizers of the event.

Removed all hyperlinks from each tweet and lowercased all textual content in order to reduce noise and sparsity.

Bi-Directional LSTMs

* s = max {0,W[fw; bw] + d}, W is a learned parameter matrix, fw is the forward LSTM encoding of the message, bw is the backward LSTM encoding of the message, and d is a bias term.
* The vector s is then used to compute the probability distribution of the emojis given the message. Formula in paper.
* s. The loss/objective function the network aims to minimize is Loss = −log(p(em | s)), where m is a tweet of the training set T , s is the encoded vector representation of the tweet and em is the emoji contained in the tweet.
* The inputs of the LSTMs are word embeddings.
* Two representations were used for word embeddings, word representations and character based representations.

Baselines:

* Bag of words
  + represent each message with a vector of the most informative tokens (punctuation marks included) selected using term frequency−inverse document frequency (TFIDF)
  + a L2-regularized logistic regression classifier to make the predictions.
* Skipgram vector average
  + Formula in paper

# An Approach for Text-to-Emoji Translation

They built a dictionary of translations from words to emojis. It included 400+ action-to-emoji translations by Wicke (2017) and all entries from EmojiNet by Wijeratne et al. (2017). From the latter, they perform a term-frequency inverse document frequency (tf-idf) analysis on the labels to weight the most important emoji for each label. Used ConceptNet to extend each label, for example the word “cat” gets extended to “feline” and “kitten”.

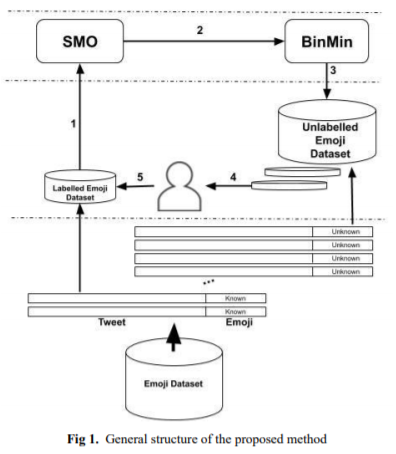
Then, the translation system takes a sentence as an input. The sentence will be filtered for common stopwords. For each word in the sentence, it is checked whether the word is similar (similarity checked here using Python’s difflib SequenceMatcher) to an entry in the dictionary. If there is a match the corresponding emoji will be stored.

To experiment, they fed sentences to four different models: SemEval, InfoRet (theirs), DeepMoji, and DangoApp. They provide an evaluation of each of the model’s results if needed.

# INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

Use JCLAL framework which offers several Active Learning strategies for multi-label learning tasks by using MULAN (java as language).

Proposed method:



1. Train method with a limited number of instances from the labelled emoji dataset. In the Active Learning process, they use the Sequential Minimal Optimization (SMO) [14] method as a base classifier (improved version of Support Vector Machines). They use an adaptation of SMO which deals with the multi-class classifier as pairwise classification tasks of this binary method in Weka.
2. Focuses on the query strategy in the Active Learning process and the query strategy is set as Binary Minimum (BinMin) from the multi-label strategies. The multi-label strategy is transformed into a multi-class classification by assigning only one label for each tweet.
3. BinMin strategy selects the most critical samples.
4. Samples are given to the simulated oracle.
5. The labelled dataset is increased with the selected and labelled sample.

SMO Base Classifier:

* Lagrange multipliers are used for detecting the local maxima and minima of a function subject to equality constraint. These linear equality constraints are (where S is an SVM hyperparameter, k is the negative of the sum for the multiplication of binary labels 𝑦1 and 𝑦2 and multipliers 𝜑1 and 𝜑2 in the equality constraint.):
  + 0 ≤ ϕ1, ϕ2 ≤ S
  + y1ϕ1 + y2ϕ2 = k
* Emojis are assigned as labels to tweets and the sample labelled dataset including tweet and known emoji label is extracted from the large Emoji dataset.
* Train the SMO model to predict labels for the selected tweets supplied by the BinMin query strategy.
* The simulated oracle adds the most convenient samples which contain tweets and their predicted labels to the current labelled dataset.

BinMin Query Strategy:

* selects the optimal unlabelled sample as 𝑎𝑟𝑔𝑚𝑖𝑛𝑥∈𝑈 = (𝑚𝑖𝑛𝑖=1,2,…,𝑑 | (𝑓𝑖(x)|), where argmin is the notation of worst-case for the unlabelled dataset U that includes sample x.
* The minimum absolute distance is then evaluated among the binary classifier 𝑓𝑖(x) on the binary problem associated with class i from a set of d version spaces. Hence, we leverage SMO as the binary classifier and it selects unlabelled samples concerning the most uncertain label.
* The steps are iterated steps until reaching the highest performance scores.

Preprocessing:

* To preprocess of the generated dataset, all hyperlinks, mentions, and hashtags from each tweet are removed and lowercased all textual content to reduce noise.