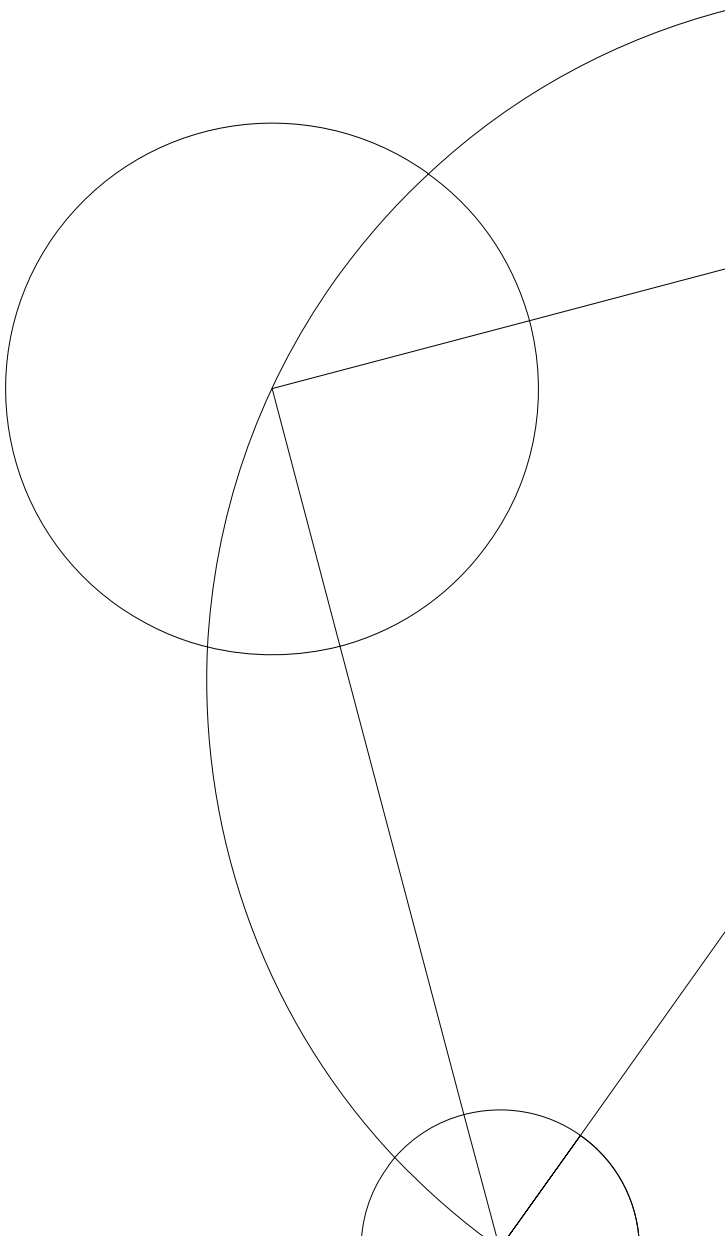
UNIVERSITY OF COPENHAGEN

Language processing 2

Features, Architecture, Combine Them Together



Students: Chenxiao Ma (rct514), Yuqin Zhou (qvk729)

IT and Cognition

Contents

Abstract

This research uses the CLEF 2022 dataset to profile ironic and stereotype spreaders on Twitter, representing their tweets at three levels: naïve features, sparse vectors, and embeddings. We employ three classic machine learning classifiers (logistic regression, support vector machine, and random forest), as well as a new architecture dubbed the voting model, to combine the strength of each feature. The findings highlight the merit of embeddings, which include subword-level information in particular. Furthermore, depending on the classifier's mechanism, the quality of the features will affect the classifiers to varying degrees.

1 Introduction

Kierkegaard used the notion of irony to illustrate the border between aesthetic and ethical stages of existence[18], arguing in his dissertation that people were obliged to think for themselves to defend their values as a result of Socrates' challenging questions, resulting in the birth of subjectivity[17]. Surely, irony, as a rhetorical device, is inextricably linked to subjective experience and is often used in everyday life, with the internet environment being no exception.

However, when people use this narrative to target specific groups, display hostility, and engage in harmful behavior, the norm of a healthy society is irreversibly undermined, despite the fact that the fundamental objective for our society is at least to protect each individual from discrimination, racism, and threats. Given that enumerate texts are produced concurrently all over the world, it is critical to develop effective methods for classifying harmful information in order to limit the number of victims of all kinds. In light of this, one of the tasks at PAN'22, also the theme of this article, is to create adequate profiles for Twitter users, with a special focus on persons who purposefully use irony to spread inappropriate stereotypes to minority groups that are already in an unequal situation, potentially causing harm.

Besides, in the discipline of Natural Language Processing (NLP), irony is considered as figurative language and researchers usually classify it as verbal irony or situational irony[32]. Thus, the task characteristics (i.e., we already have the label) suggested that we could apply similar techniques to other figurative language problems. In PAN'21[30], for example, `LSTM` models[1] and models combining `LSTM` and `BERT`[35] were presented to recognize hate speech. But sadly, these models do not place well in the leaderboard[1]. Moreover, despite the fact that a model based on `AutoML` and `BERT` came in fourth place[2], the overall winners utilized the classical machine learning methods with features engineering. This transformer-based model was defeated by a simple voting model based on the combinations of n-grams[3].

These results suggest that we can first observe and test simple features before developing or combining deep learning approaches. Furthermore, rather than relying just on majority votes, we may be able to improve their voting procedures using machine learning technologies. Besides, Cabrera et al suggest that we can also consider the text's emotional content[28].

Thus, in this manuscript, we first distinguish two candidates, average tweet length and the use of emojis, to aid us in finding distinctions based on their distinct frequency in two categories. Then, to further refine this coarse description, the frequency of specific tokens is employed. Finally, we look at word embeddings as a means to improve our algorithm, which is founded on the assumption that prior success is driven by semantic nonconformity between two groups, which is implicitly represented in token distribution disparities.

We explored two different architectures for merging these features, namely concatenation and voting models. In our voting models, we employed the most accurate learning strategy for each feature, and we experimented two-stage learning, where we also implemented machine

[1] https://pan.webis.de/clef21/pan21-web/author-profiling.html

learning approaches to obtain a more competent method for combining each vote. Finally, We also look at the possibility of using transformer-based models and other advanced neural networks to enhance our model over time, as well as the challenges that any method would face.

In the following section, we will go over our methods in detail.

2 Methodology

2.1 Dataset: IROSTEREO

We use the dataset provided by PAN at CLEF 2022, which was built for profiling **iro**ny and **stereo**type spreaders on Twitter (IROSTEREO)[2]. The class-balanced dataset consists of a `truth.txt` and 420 XML files. The `truth.txt` contains the ground truth labels about whether authors are irony and stereotype spreaders. Each XML file contains 200 tweets pre-processed by the organizers to represent a Twitter user profile. For instance, some strings in tweets have been replaced: `@user` by `#USER#`, hashtags[3] by `#HASHTAG#` and links by `#URL#`. Our goal was to discover a pattern for detecting ironic content using 200 tweets from each Twitter user (420 in total). In this section, we'll show how we chose features and designed classifiers.

2.2 Average tweet length and total emoji counts

Before proposing any model, we firstly extracted some simple but might representative features of the dataset. The first we considered was the average tweet length per user over irony and non-irony classes, as shown in Figure 1. The length of tweets was calculated by character, in order to comply with Twitter's 280 character limit for each tweet. Note that the tweets in IROSTEREO have been pre-processed, so tokens such as `link` and `@user` in Twitter are counted as one character, but here we treat them as multiple characters such as `#USER#` and `#URL#`. The bin width has been set at 5 characters, which is roughly one word in length.[4]. We noticed that the irony class followed a "bell curve," and that irony users tended to write tweets with 150 characters or less when compared to non-irony users. Also, many non-irony users tended to write tweets that were between 155 and 175 characters long, with a few outliers having tweets that were over 250 characters long on average. After looking at the length of tweets, we concentrated on emojis, a well-known but powerful feature.

The use of emojis on social media has a rapid increase over the past years[22]. Many researchers have studied how emojis carry semantic meaning, such as their close relationship to words[5]. Therefore, we believe emojis can partly reflect the emotions and intentions of users. In this project, we calculated the total emoji counts per user over two categories, as shown in Figure 2. In both categories, about 75 users used fewer than 10 emojis in 200 tweets. On a scale of 10 to 50 emojis, irony class had more users than that of non-irony class. Also, both of these categories had outliers who used more than 250 emojis in 200 tweets.

2.3 Sparse vectors

After examining the two features mentioned above, we noticed that irony and non-irony users exhibited different behaviors when writing tweets. The next step was to figure out how to segment these two factors so that words and emojis could offer more valuable information. Utilizing the frequency of each token and emoji in tweets is a reasonable way to attain this goal.

[2]This link contains information regarding PAN at CLEF 2022.
[3]The information about hashtags in Twitter can be found in this link
[4]The actual number is determined by the tokenizer.

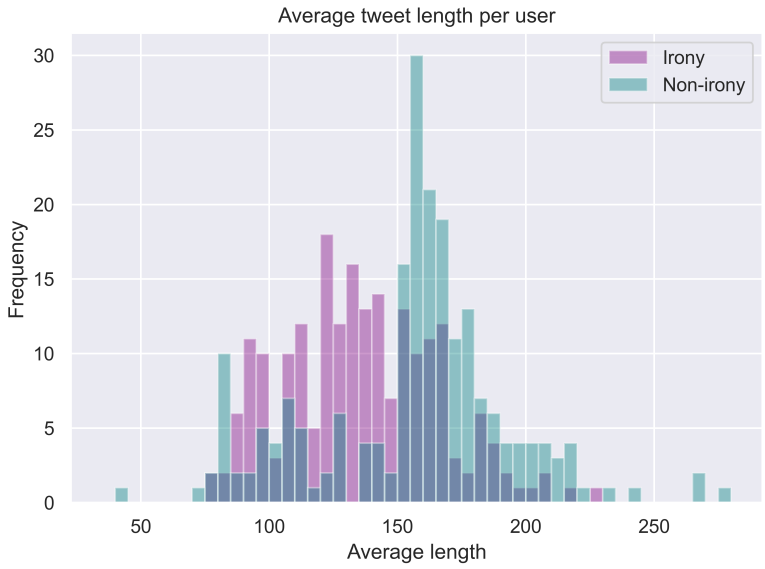


Figure 1: Average tweet length per user over irony and non-irony classes.

First, we extracted emojis in tweets based on the emoji list provided by emoji 1.7.0 and counted the frequency of each emoji over irony and non-irony classes. The top-15 emojis used by irony users were:

😂, 🤣, 🤔, 🙄, 😉, 👍, 👇, ❤️, 🤦‍♂️, 🤷‍♂️, 😒, 😄, 😏, 😁, 😊,

while non-irony speakers preferred to use:

😂, 😭, 🤣, ✅, 🔥, ❤️, 🤔, 🙏, 🙄, 🇺🇸, 💀, 🚀, 👇, 😩, 🇺🇦.[5]

We may deduce from these two lists that the use of emojis in the two categories is slightly different. Therefore, we trained and tested classifiers using the *entire* emoji list with 4702 dimensions.

To gain the frequency of each token, we used three tokenizers: Tweet tokenizer, Multi-Word Expression tokenizer (MWE) and BERT tokenizer. Tweet tokenizer is a rule-based tokenizer designed to tokenize tweets. This tokenizer will not divide hashtags such as `#Truth` to `#␣Truth`, while others tokenizers may make such mistake. Compared with Tweet tokenizer, MWE tokenizer was used to see if different tokenizers would have a significant effect on the classification results. Another consideration for using MWE tokenizer was because that it can re-group some tokens into multi-word expressions, giving us more possibilities to manipulate the tokens. However, unlike subword tokenizers, these two word-based tokenizers are unable to capture meaning-bearing units such as the morphemes $-est$ or $-er$. The tokenizer in the BERT model[12], which is based on the `WordPiece` algorithm first proposed by Schuster et al[34], can assist us in retrieving this information. Additionally, we included `#USER#`, `#HASHTAG#`, `#URL#` as special tokens to avoid tokenization errors.

After selecting the three tokenizers, we tokenized tweets and constructed word vectors for each Twitter user. During the training stage, a token dictionary was built, with tokens as keys and matching values reflecting their frequency. As a result, the classifier was powered up by a 24515-length vector representing each user. At the test stage, we used the frequency of each token in a test set to predict labels. If a word only occurred in the test set, we would simply ignore this token. This choice was made to be consistent with the decision we made in Section 2.4, which will be explained later.

To summarize, we represented each Twitter user by the frequency of words and emojis

[5]Compared to emojis in Twitter, these emojis have the same Unicode characters but are displayed in a different font. The font we used can be found in: this link

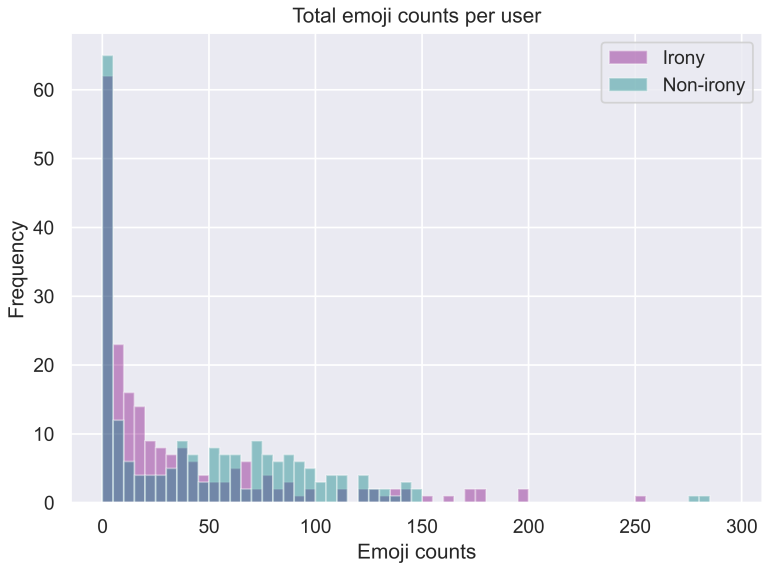


Figure 2: Total emoji counts per user over irony and non-irony classes

they use. This was not, however, our ultimate solution to the challenge. In the following part, we'll go through our last picked key: embeddings.

2.4 Dense vectors

This section is organized in three aspects: the reasons for choosing embeddings, the two steps for obtaining emoji and word embeddings in tweets, and how we get user embeddings.

Importantly, the purpose of this study is to figure out how to accurately reflect users' writing styles, determining whether or not they are expressing sarcastic content. In the previous section, we assumed that a Twitter user can be represented by the words and emojis they use, leading to the question: how to represent the meaning of a word or emoji. Regarding this question, vectors semantics provides some ideas such as word frequency and embeddings[16]. Embeddings often outdo word frequency for training classifiers. First, embeddings largely reduce vector dimensions. If words are represented as 100-dimensional dense vectors instead of as 29217-dimensional sparse vectors, it saves computational resources and possibly helps with generalization and avoiding overfitting. Second, embeddings capture more semantic and syntactic information than sparse vectors. A specific proof is the parallelogram model proposed by Rumelhart and Abrahamson (1973) solving some analogy problems [33]. Under this framework, we can use word embeddings like GloVe to compute a formula $apple + tree - grape$, obtaining the closest answer $vine$[6]. Therefore, using embeddings that contain semantic information, we can capture extra clues about how users are writing their tweets.

To obtain emoji and word embeddings in tweets, we firstly searched for pre-trained embeddings. We used `emoji2vec` proposed by Francesco et al.(2016)[13], supporting over 1661 Unicode emojis embeddings. These embeddings were trained from their description in the Unicode emoji standard[7] and Google News `word2vec`[23]. Considering `emoji2vec`'s performance, it has been evaluated on a Twitter sentiment analysis task and outperformed than the emoji embeddings trained over 100 million English tweets by Barbieri et al. (2016)[5]. In

[6]Note that this result requires to exclude the three input words and their morphological variants such as *apples*

[7]Description of emojis can be found in this link

the same way, we used GloVe[8] and FastText[9] word embeddings pre-trained on large Twitter datasets. We chose the GloVe because they consistently outperformed `word2vec` for the same corpus, vocabulary, window size, and training time[27]. More specifically, GloVe 200-dimensional word embeddings were used due to its best performance on the IROSTEREO dataset than other dimensions. As our second choice, we used 100-dimension monolingual FastText word embeddings provided by Jose et al.[10], This choice was made mainly because the FastText encoded subword information such as word morphology, as each word was trained as the form of a bag of character n-grams[7]. Note that the FastText contained most of the tokens in tweets, including words, numbers and emojis, which might be a factor influencing our classifier performance.

Next step we did was to align each emoji and word with the pre-trained embeddings. For the emoji alignment, there were 1091 emoji in the IROSTEREO dataset and we matched 728 of them with `emoji2vec` embeddings. Table 1 specifies the alignment between words and pre-trained word embeddings. Consider the two tokenizers we used, Tweet tokenizer produced fewer vocabulary than MWE tokenizer and matched more words with the FastText than the GloVe. On the contrary, tweets tokenized by MWE tokenizer matched less words with the FastText than the GloVe.

Embeddings	**Tokenizers**	**Matched**	**Missed**	**Tokens**	**Matched Pct.**
FastText	Tweet	51248	26027	77275	**66.32**%
	MWE	48973	35991	84964	57.64%
GloVe	Tweet	50336	26939	77275	**65.14**%
	MWE	50288	34676	84964	59.19%

Table 1: Alignment between words in tweets and pre-trained word embeddings.

Here, we did not included BERT tokenizer, since the GloVe and the FastText required that the form of tokens was word-based. It is worth mentioning how we dealt with unknown words. Our decision was to ignore unknown words because we assumed that the `<unk>` embeddings provided by the GloVe and the FastText were incompatible or inappropriate for the IROSTEREO dataset. To comply with this decision, we also ignored unknown words when creating word vectors in the previous section.

Lastly, we need to decide how to use word and emoji embeddings to generate a user embedding. It can be done in a very computationally expensive way. For example, we can concatenate together all the words and emojis used by a Twitter user. However, we chose three computational-friendly and simple pooling functions: sum, average and normalization. Namely, we used:

$$\mathbf{x}_{i,\ \text{sum}} = \sum_{j=1}^{n_i} x_j; \quad \mathbf{x}_{i,\ \text{avg}} = \frac{\mathbf{x}_{i,\ \text{sum}}}{n_i}; \quad \mathbf{x}_{i,\ \text{norm}} = \frac{\mathbf{x}_{i,\ \text{sum}}}{\|\mathbf{x}_{i,\ \text{sum}}\|}.$$

As for the meaning of notations, n_i represents the total number of words or emojis used by a Twitter user i; x_j represents the embedding of the j-th word or emoji and its dimension relies on the pre-trained embedding we adopt; $\mathbf{x}_{i,\ k}$ represents the embedding of user i, where k can be the synonymous of the method of summation, averaging, and normalization, respectively; and $\|\cdot\|$ corresponds to L^2 norm (Euclidean norm). Therefore, if we use FastText 100-dimensional pre-trained word embeddings, for example, we will align the words and use a pooling function like normalization to get user embeddings with a dimension of 100. Then, each user embedding will sever as a training example for classifiers. In the next section, we will describe our classifier selection and how to utilize the features we obtained.

[8]Pre-trained GloVe word embeddings on Twitter can be found in this link

[9]Pre-trained FastText word embeddings on Twitter can be found in this link

2.5 Classifiers

To evaluate different features, we used three classical machine learning classifiers: Logistic Regression (LR), Support Vector Machine (SVM) and Random Forest (RF).

All models were implemented in Python 3.9 using the Scikit-Learn library[26]. We implemented the regularized `logistic regression` with L2 penalty term and the inverse of regularization strength is set to 1. `Support Vector Machine` was implemented with radial basis function (rbf) kernel and squared L2 penalty. We chose the rbf kernel because it has a strong fitting ability and has proven successful at CLEF 2021[11]. Our last choice was an ensemble tree-based method: `Random Forest`. At the training stage, it will build 2000 sub-samples of the dataset using bootstrap. For each sub-sample, it randomly select a subset of features at each split[10]. Averaging all sub-sample's prediction, the final outcome usually guarantees a good performance.

With the three classifiers, we firstly compared each feature and then combined each feature or changed the classifier to obtain a highly accurate classifier. The following paragraphs show how the experiments were organized:

Single features: We looked at each feature separately to compare the information it contained. Non-embedding features includes average tweet-length, total emoji counts, emoji vector, and word vector obtained from three tokenizers (Tweet, MWE, and BERT tokenizers). For embeddings features, we listed the results of nine combinations between three types of embeddings (`emoji2vec`, GloVe and FastText) and three pooling functions (sum, average and normalization). We also combined `emoji2vec` with the GloVe. This was due to the fact that Jose et al's FastText embedding includes emoji embedding[10]. To lessen the influence of emoji, we also consider to combine the emoji embeddings with the GloVe when comparing it to the FastText.

Concatenation: We concatenated different features together to increase the accuracy. We firstly concatenated non-embedding features: average tweet-length, emoji vector and word vector obtained by the best tokenizer[11]. We also skipped over the total number of emojis, which can be obtained from a linear combination of its basis.

Voting models: The second attempt to merge features was to use voting models. As a first step, we picked the most accurate classifier for each `single feature` and trained it to get votes. Then, to get the final predictions, we combined votes using the LR, SVM, RF, and Majority Vote (MV)[12] classifiers. Our main motivation was that voting models could treat the results of each feature equally important in the initial state. Also, the voting models could only assign weights to the entire feature instead of one dimension in that feature.

We used `5-fold cross-validation` during the training process in the experiment parts. Dealing with `single features` and `concatenation`, we split the training and test set in the ratio of 8:2. For experiments in `Voting models`, we evaluated three different ways of splitting the training and test sets. In the first step, the single features was trained on 60% or 70% of the data, and the voting models was trained on 20% 10% accordingly. Lastly, the results were tested on 20% of the data. The different proportions of data to train the voting models were to see if the size of the data can affect the performance of its architecture.

For the evaluation metrics, we used `accuracy` to select the best performance of the classifiers. This chose was based on two reasons. Firstly, as the PAN'22 requires, the performance of our classifiers will be ranked by accuracy. Therefore, this project focuses on the effect of different features on the accuracy of the classifiers. Secondly. the dataset is balanced where the number of irony users and non-irony users is 210. Thus, if we assume that the two

[10]The size of the feature subset is the square root of the original feature dimension

[11]The best tokenizer is one of the three tokenizers that achieve the highest accuracy when using word vectors as a single feature.

[12]The MV classifier predicts the user's label based on the number of positive or negative labels predicted by the first stage classifier.

categories are of equal importance, we can simply choose the best classifier based on their accuracy. In addition, the accuracy was the average of 5-fold cross-validation results. To be specific, for each training stage, we split the training data into 5 folds to train a model on the 4 folds while validating a model on the one fold, all of this 5 times. The final average accuracy will be more robust than that of hold-out method. After obtaining the accuracy of the classifiers, we selected the highest one and specified its classification details including `precision`, `recall`, `F1 score`, `macro avg` and `weighted avg`.

To conclude, we illustrate here how we selected features and evaluated them for irony or stereotype detection, and we will discuss the results of each experiment in the following section.

3 Results and discussion

3.1 Non-embedding features

To follow our research plan discussed in Section 2.5, we firstly examined `single` non-embedding features: average tweet-length, total emoji counts, emoji vector and word vector as shown in Table 2.

Input features	LR	SVM	RF
Average tweet-length	68.10%	**70.24%**	59.05%
Total emoji counts	53.33%	60.48%	**61.19%**
Emoji vector	**71.67%**	68.57%	64.29%
Word vector (Tweet)	90.71%	88.33%	**90.95%**
Word vector (MWE)	**90.71%**	88.57%	90.48%
Word vector (BERT)	**92.14%**	89.76%	91.43%
Concatenation	91.43%	88.57%	**91.90%**

Table 2: Performance of the three classifiers using non-embedding features separately in `single features` and lastly concatenating these features together.

In general, using the average twee-length achieves higher accuracy than total emoji counts, while using word vectors gives better results than emoji vectors. We speculate that emojis carry less information than words. Although emojis are used to communicate simple things or feelings in a fresh way[5], most tweets are still words, which can be verified by Figure 2.

The second finding is that the BERT tokenizer outperforms than the other two tokenizers. This difference can be caused by the `WordPiece` algorithm extracting additional morphological information at the subword level. Lastly, concatenating these features together reduces the accuracy of LR, SVM. We conjecture that this accuracy is supported by word vector (BERT), and what reduces its accuracy is the redundant information provided by the other features. Considering the slight improvement in the accuracy of RF, this result may benefit from the fact that RF use only the square root number of the entire feature.

3.2 Embedding features

This part compares embedding features mentioned in `single features`: `emoji2vec`, GloVe and FastText and finally concatenates all previous features, as shown in Table 3.

First, one previous finding is verified again: emojis are less effective than words in this task. Second, the three pooling functions mentioned in Section 2.4 affect the results to varying degrees. In most cases, normalization slightly improves the performance of RF, while its positive or negative influence on LR or SVM depends on the specific features. For the sum

Input features	LR	SVM	RF
emoji2vec (SUM)	60.24%	67.14%	**72.14%**
emoji2vec (AVG)	60.24%	67.14%	**71.90%**
emoji2vec (NORM)	61.90%	**74.05%**	71.43%
GloVe (SUM + Tweet)	87.14%	75.95%	**90.00%**
GloVe (AVG + Tweet)	86.19%	62.15%	**91.67%**
GloVe (NORM + Tweet)	67.14%	64.05%	**91.19%**
GloVe (SUM + MWE)	88.10%	75.00%	**91.43%**
GloVe (AVG + MWE)	87.62%	63.33%	**91.43%**
GloVe (NORM + MWE)	70.00%	67.12%	**91.90%**
FastText (SUM + Tweet)	87.86%	81.90%	**89.05%**
FastText (AVG + Tweet)	85.71%	85.95%	**91.90%**
FastText (NORM + Tweet)	81.19%	85.71%	**92.14%**
FastText (SUM + MWE)	85.71%	75.95%	**88.33%**
FastText (AVG + MWE)	81.90%	72.14%	**90.95%**
FastText (NORM + MWE)	74.05%	71.19%	**91.43%**
Concatenation	91.67%	88.57%	**92.86%**

Table 3: Performance of the three classifiers using embedding features separately in `single features` and lastly concatenating them with non-embedding features together. SUM, AVG and NORM indicate three pooling function as introduced in Section 2.4

or average pooling functions, their performance on LR and SVM is quite unstable compared to RF. Third, we attribute the slight improvement in the accuracy of the `concatenation` to the subword information provided by Word vector (BERT) and the special way of selecting features in RF. We also present the classification report of this best model, as shown in Table 4.

	Precision	Recall	F1-Score	Support
I	96%	98%	97%	46
NI	97%	95%	96%	38
Accuracy			96%	84
Macro Avg.	97%	96%	96%	84
Weighted Avg.	96%	96%	96%	84

Table 4: Classification report for the best model, concatenating FastText embeddings, subword tokens, emoji vectors, and average tweet length.

Importantly, when it comes to the word embedding methods, the FastText outperforms the GloVe but has less dimensions (100 v.s. 200). This could be due to the fact that FastText carries information on subwords and emojis. To learn further, we combined emoji2vec and the GloVe to train the three classifiers. Various combinations have been attempted, such as using the same pooling algorithm for each or using their best pooling function. The highest result was 90.95%, which is still lower than 92.14% obtained by FastText (NORM + Tweet) using RF. These results suggest that the FastText has higher potential than the GloVe because of its subword information.

3.3 Voting models

As shown in Table 5, The three classifiers outperform the MV classifier by a large margin, implying that each feature should be combined with suitable weights rather than being treated equally.

Furthermore, an interesting result is that with four or five characteristics, RF no longer offers better results. The fundamental benefit of RF is that it can extract important information from hundreds of features, as demonstrated in the previous section. Also, as the size of the training data grows larger, the voting model improves.

Models	MV	LR	SVM	RF	Allocation ratio
NE Voting	79.29%	**91.43%**	90.71%	90.71%	60%, 20%
NE Voting	80.24%	**91.90%**	90.71%	91.43%	70%, 10%
NE Voting	82.38%	-	-	-	80%, 0%
E Voting	83.33%	**91.90%**	90.95%	91.19%	60%, 20%
E Voting	84.24%	92.14%	**92.62%**	92.14%	70%, 10%
E Voting	**86.19%**	-	-	-	80%, 0%

Table 5: Voting model performance utilizing several classifiers. Non-embedding features are referred to as `NE`, and extra embeddings within `NE` features are referred to as `E`. The `allocation ratio` refers to the proportion used to train single feature classifiers and the proportion used to train the voting model. The test set always takes up 20% of the total data set.

3.4 Future directions

Although previous section only covers word-level embedding, it can plausibly be extended to the subword level. However, on this dataset, the existing pretrained transformer models (e.g., RoBERTa[20], BERTweet[25]) do not perform well. This could be due to the fact that they were largely trained on the `SemEval 2018` dataset[36], and these trained models may not be transferable. Besides, we have two alternatives for the porting process: use available twitter-based model's architecture but train it based on new data and labels, or use the trained generic model's parameters (and architecture) but solely update parameters. Research conducted by Beltagy et al[4] implies that the latter may outperform the former, and that it is worth putting to the test on this dataset as well. In addition, given our goal is to develop a classifier for users (rather just tweets), collecting all tweets into a single large document and retraining the longformer[6] to get fine-tuned parameters could be a smart move. The most significant disadvantage of all of these approaches is that `K-Fold validation` will require a significant amount of computational resources to be compared to the existing methods in the preceding section.

Apparently, each user's tweet does not always match the classification that has been assigned to them. If a method exists to assess each user's unique tweets and connect the distribution of the degree of irony for each tweet from this person to its label, we can undoubtedly get a better model. However, the existing model, as stated in the preceding paragraph, does not perform well in this regard. We discover that more than half of the tweets from each user in both categories are labeled as negative when we first execute the sentiment analysis using the transformer models. Although there is a distinction between negative and irony, our method's performance is dramatically reduced when we exclusively use tweets with negative sentiment to extract features and train them with our previous model. To make matters worse, we lack a tweet-based label with which to train the algorithm. Fortunately, to train a transformer model that can return irony intensity, we can use the `SemEval 2018` tweet-based data, manually annotate tweets that are plainly sarcastic, and directly collect

data from Twitter, where individuals have annotated their own tweets[19]. We may then utilize this data to create a classifier that correlates the ironic intensity distribution with the label. Further, if the context of the user's conversation is accessible, as we'll discuss later, the model probably can be improved as well. If not, we can utilize `GAN` to generate one while also obtaining an ironic discriminator. But the demand for computing resources has increased further.

3.5 Limitations

The flaws in this study can be roughly divided into two categories: the shortcomings of the dataset we obtained and the defective methodology we used.

For starters, several political science studies demonstrate that people who spread damaging content on Twitter are more likely to interact with people who share their ideologies[9]. Furthermore, harmful tweets are usually triggered by offline events[31]. Users also utilize hashtags to swiftly arrange themselves into groups with similar ideologies[15]. However, the dataset does not give the context: we don't know what kind of information each user interacts with, and we don't have the hashtags' detail (we would not have considered `GAN` otherwise).

None of the strategies we employed in this paper, on the other hand, are good at generalization. If we purely apply `FasText`'s embedding to a simple sentence like "I love you!" the obtained classifier will identify it as irony, whereas "I like him!" will be treated as non-irony[13]. In addition to the previous mentioned paragraph that state-of-the-art transformer models trained for `SemEval 2018`'s task cannot be transferred to this dataset, it can also be shown in the fact that a user classified as irony by the classifier under specific setting of parameters is most likely just tweeting about serious historical facts. We also don't know to what extent the strategy we outlined earlier will be in resolving this issue. In fact, due to the small dataset, the misclassified users were frequently less than four (or even no misclassification). The improved accuracy could simply be due to the algorithm's continual data overfitting; yet, the improved accuracy may not alleviate the problem that our society is experiencing.

4 Conclusion

In this article, inspired by previous year works, we started with two naive features: tweet length and emojis. Further powered up by subword tokens and FastText embeddings, our most accurate approach concatenated all features outperformed voting models where two-stage voting based on machine learning techniques surpassed majority voting but were still constrained by the size of this dataset.

However, given the possibility to enhance our model by incorporating transformers and generative adversarial networks, a cautious design is insufficient; our ultimate goal is to create a functional model for the real world, not just enhance accuracy to identify irony on a toy dataset, which overlooked contexts, structure of the users connections, and key information triggered by offline events.

References

[1] Kwabena Odame Akomeah, Udo Kruschwitz, and Bernd Ludwig. University of regensburg@ pan: Profiling hate speech spreaders on twitter. In *CLEF*, 2021.

[2] Talha Anwar. Identify hate speech spreaders on twitter using transformer embeddings features and automl classifiers. 2021.

[13] Duplicating this content 200 times to mimic 200 tweets yields the same results.

[3] Fazlourrahman Balouchzahi and G Sidorov SHL. Hssd: Hate speech spreader detection using n-grams and voting classifier. In *CLEF*, 2021.

[4] Francesco Barbieri, Jose Camacho-Collados, Leonardo Neves, and Luis Espinosa-Anke. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. *arXiv preprint arXiv:2010.12421*, 2020.

[5] Francesco Barbieri, Francesco Ronzano, and Horacio Saggion. What does this emoji mean? a vector space skip-gram model for Twitter emojis. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3967–3972, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[6] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[8] Joan C. Borod and K. R. Scherer. *Psychological Models of Emotion*, pages 137–162. Oxford University Press, 2000.

[9] William J Brady, Julian A Wills, John T Jost, Joshua A Tucker, and Jay J Van Bavel. Emotion shapes the diffusion of moralized content in social networks. *Proceedings of the National Academy of Sciences*, 114(28):7313–7318, 2017.

[10] Jose Camacho-Collados, Yerai Doval, Eugenio Martínez-Cámara, Luis Espinosa-Anke, Francesco Barbieri, and Steven Schockaert. Learning cross-lingual embeddings from twitter via distant supervision. In *Proceedings of ICWSM*, 2020.

[11] Tanise Ceron and Camilla Casula. Exploiting contextualized word representations to profile haters on twitter. In Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi, editors, *Proceedings of the Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21st - to - 24th, 2021*, volume 2936 of *CEUR Workshop Proceedings*, pages 1871–1882. CEUR-WS.org, 2021.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[13] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel. emoji2vec: Learning emoji representations from their description. *CoRR*, abs/1609.08359, 2016.

[14] Anjalie Field and Yulia Tsvetkov. Entity-centric contextual affective analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2550–2560, Florence, Italy, July 2019. Association for Computational Linguistics.

[15] Sarah J Jackson, Moya Bailey, and Brooke Foucault Welles. *# HashtagActivism: Networks of race and gender justice*. Mit Press, 2020.

[16] Dan Jurafsky and James H. Martin. *Speech and language processing*. Prentice Hall, Pearson Education International, 2014.

[17] Søren Kierkegaard. *Kierkegaard's Writings, II, Volume 2: The Concept of Irony, with Continual Reference to Socrates/Notes of Schelling's Berlin Lectures*, volume 28. Princeton University Press, 2013.

[18] Søren Kierkegaard. Concluding unscientific postscript. In *Concluding Unscientific Postscript*. Princeton University Press, 2019.

[19] CC Liebrecht, FA Kunneman, and APJ van Den Bosch. The perfect solution for detecting sarcasm in tweets# not. 2013.

[20] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[21] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028, 2002.

[22] Gary Lupyan and Rick Dale. Why are there different languages? the role of adaptation in linguistic diversity. *Trends in Cognitive Sciences*, 20(9):649–660, 2016.

[23] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[24] Saif M. Mohammad and Svetlana Kiritchenko. Using hashtags to capture fine emotion categories from tweets. *Computational Intelligence*, 31(2):301–326, 2015.

[25] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. Bertweet: A pre-trained language model for english tweets. *arXiv preprint arXiv:2005.10200*, 2020.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[28] JESUS HIRAM CABRERA PINEDA and SABINO MIRANDA JIMENEZ. Profiling hate speech spreaders on twitter through emotion-based representations notebook for pan at clef 2021. *Cabrera, Hiram; Miranda Jiménez, Sabino; S. Téllez, Erick.(2021). Profiling Hate Speech Spreaders on Twitter through Emotion-based Representations Notebook for PAN at CLEF 2021. INFOTEC. http://ceur-ws. org/*, 2936, 2021.

[29] Lin Qiu, Han Lin, Jonathan Ramsay, and Fang Yang. You are what you tweet: Personality expression and perception on twitter. *Journal of Research in Personality*, 46(6):710–718, 2012.

[30] Francisco Rangel, Gretel Liz De la Peña Sarracén, BERTa Chulvi, Elisabetta Fersini, and Paolo Rosso. Profiling hate speech spreaders on twitter task at pan 2021. In *CLEF (Working Notes)*, pages 1772–1789, 2021.

[31] Stig Hebbelstrup Rye Rasmussen and Michael Bang Petersen. From echo chambers to resonance chambers: How offline political events enter and are amplified in online networks. 2022.

[32] Antonio Reyes, Paolo Rosso, and Davide Buscaldi. From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, 74:1–12, 2012.

[33] David E Rumelhart and Adele A Abrahamson. A model for analogical reasoning. *Cognitive Psychology*, 5(1):1–28, 1973.

[34] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

[35] Moshe Uzan and Yaakov HaCohen-Kerner. Detecting hate speech spreaders on twitter using lstm and bert in english and spanish. 2021.

[36] Cynthia Van Hee, Els Lefever, and Véronique Hoste. Semeval-2018 task 3: Irony detection in english tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 39–50, 2018.

A Division of labour

The project was equally shared among the team members (C.M., and Y.Z.). We worked together on all procedures such as discovering features, analyzing the results, and writing this report. Therefore, we firmly believe that this report should be viewed as a critical collaborative output. The related code of this project is attached to the GitHub.