Universiteit Leiden

# Master Computer Science

[Using ensemble models with structural information in social media to aid rumour stance classification]


Name: [Chen Wang]
Student ID: [s2230496]


Date: [04/06/2021]

Specialisation: [Advanced Data Analytics]

1st supervisor: [Suzan Verberne]
2nd supervisor: [Stephan Raaijmakers]

Master's Thesis in Computer Science


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

**Abstract**

In previous research, using structural information to address stance classification in conversational social media has been limited to a single sequential model. In this work, we explore the benefit of constructing ensemble models with sequential and non-sequential models to aid rumour stance classification. This is a multi-classification task that aims to predict user's attitudes towards a rumour post. During the research process, we successfully adapt the original BranchLSTM method to make it able to receive various sizes of tweet conversations and make it available to the public. We experiment with combining the sequential model BranchLSTM and the pre-trained model DistilBERT by merging their predicting probabilities to the same tweet and using one-step and two-step voting classifiers to classify the new probability features. Through single-model experiment analysis, we find out that DistilBERT has advantages in addressing a more balanced dataset comparing to BranchLSTM, but BranchLSTM could perform better in predicting the large categories of the imbalanced dataset. Through ensemble experiment analysis, we prove that the ensemble model could learn both features of BranchLSTM and DistilBERT with the help of a voting classifier, but is not able to keep advantages from both deep learning models. Finally, through comparison between sequential voting classifier and non-sequential classifier, we find out that sequential voting classifier could make use of the context but may have less strength in classifying a single simple tweet comparing to the non-sequential classifier.

# 1 Introduction

Social media platforms play an essential role in modern society. People frequently visit social media for news consumption (Hermida et al., 2012), and some media specialists also increasingly collect news from social media (Zubiaga et al., 2013). Twitter, one of the most influential social media platforms, which has hundreds of millions of users, is gradually replacing the status of traditional news outlets (Hamidian and Diab, 2016). As a result, various data mining and text mining methods have been developed to discover news from social media (Dong et al., 2015; Stilo and Velardi, 2016). On the one hand, social media are very convenient for getting information easily and quickly; On the other hand, misinformation has become a serious problem (Ferrara, 2015). The spreading of rumours could confuse users and cause much potential damage to the public. Therefore, identifying misinformation on social media has become a widely discussed topic when it comes to text mining.

It is necessary to define *rumour* correctly before addressing rumours on social media. According to a survey on the detection and resolution of social media rumors (Zubiaga et al., 2018a), some previous studies incorrectly defined rumors as information that is considered false (Cai et al., 2014; Liang et al., 2015). The survey gives the prevailing definition of rumour as "*an item of circulating information whose veracity status is yet to be verified at the time of posting*" by referring to the majority definition in other literature and authoritative dictionaries.

A number of methods have been proposed to identify rumours on social media platforms (Jin et al., 2016; Rubin et al., 2016; Rubin and Lukoianova, 2015; Schifferes et al., 2014; Tacchini et al., 2017; Volkova et al., 2017). A complete rumour resolution process is defined as a complex task (Zubiaga et al., 2018a), but it always receives a piece of information that may constitute a rumour and outputs the actual truth value of the rumour. A rumour resolution system has been defined to consist of four steps (Zubiaga et al., 2018a): (1) rumour detection,

to identify whether a piece of the information constitutes a rumour; (2) rumour tracking, to monitor social media to find posts discussing the rumour and eliminating irrelevant posts; (3) rumour stance classification, to determine how each post is orienting to the rumour's veracity; (4) rumour veracity classification, which attempts to determine the actual truth value of the rumour. By determining the types of attitudes expressed in different tweets discussing the same rumour, the third task could help to the verification process (Qazvinian et al., 2011). As a key component in the rumour resolution system, rumour stance classification has attracted a lot of interest.

Since Zubiaga's original work of utilizing structural information from conversations to address rumour stance classification (Zubiaga et al., 2016a), the four-categories scheme including *supporting*, *denying*, *querying* and *commenting* has been adopted by subsequent research. In this research, we also adopt this scheme to build up a multi-class classification task. Figure 2 is an example of the tree-structure conversation, including each class of the label. The task of rumour stance classification can be defined as follows: Given a set $D$ of Twitter conversations $T_i$, each conversation is a collection of rumourous tweets $t_i$ which are discussing about a rumour $R_i$. Therefore, each Twitter conversation $T_i = \{t_1, ..., t_{|T_i|}\}$; given a new tweet $t_j$ belonging to a new conversation $T_i$ which is discussing about an unseen rumour $R_j$, the trained model $M$ needs to determine the stance of tweet $t_j$ as one of the four categories. In this task, the rumour $R_i$ is the source tweet of the conversation.

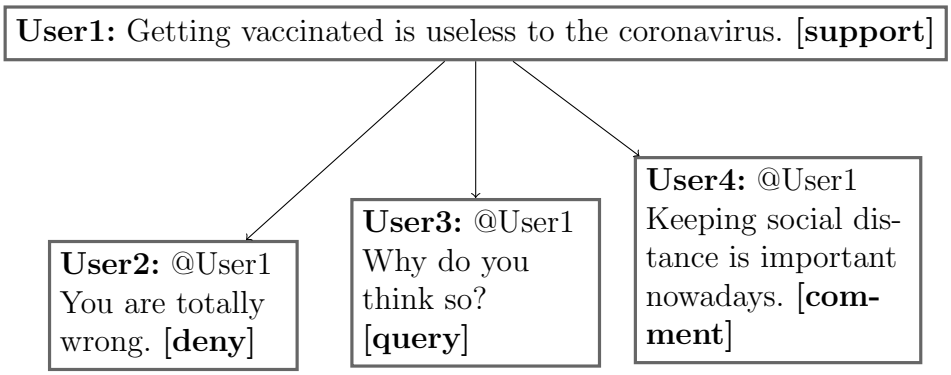



Figure 1: Example of a tree-structure Twitter conversation.

Previous research made considerable contribution to this task, but most of these methods use a single sequential model which co-operates with complex features, and they don't have much difference in their overall strategy. In other words, there are still other types of models that can be explored, and the model architecture can be designed in more ways. Having BERT (Devlin et al., 2018) as one of the representative pre-trained language models that made great success, we give the research questions: (1) how does DistilBERT compare to the single sequential model in rumour stance classification? (2) how does the combination of a sequential model and a pre-trained language model help to rumour stance classification?

In response to these two research questions, we conduct the following work and get corresponding findings:

- We adapt the original BranchLSTM classification method as the single sequential model for this task. The adapted BranchLSTM method can train and predict on the different dataset with various sizes of conversations. We publish the source code of adapted BranchLSTM as well as the random split experiment, which could help later researchers to use BranchLSTM more easily.

- We apply DistilBERT on the rumour stance classification task as another single model. We analyze the difference between two single models on the same task and find out DistilBERT has advantages in addressing a more balanced dataset comparing to BranchLSTM, but BranchLSTM could perform better in predicting the large categories of the imbalanced dataset.

- We explore how the combination of two single models and various voting classifiers work together on rumour stance classification task. We prove that the ensemble model could learn both features of BranchLSTM and DistilBERT with the help of a voting classifier, but is not able to keep advantages from both deep learning models.

- We build four sets of voting classifiers to classify the connected probability vectors from two single models, including two one-step voting classification models and two two-step voting classification models. We find out that sequential voting classifier could make use of the context but may have less strength in classifying a single simple tweet comparing to the non-sequential classifier.

Our work will be detailed in the remaining sections. In section 2, we review some previous research about rumour stance classification. In section 3, we introduce the split of the dataset and the creation of the probability dataset for voting classifiers. In section 4, we explain the structure of our ensemble model with graphical illustration. In section 5, we give the detailed experiment settings of single-model experiment and ensemble experiment, and then offer our experiment results and analyze these results. In section 6, we discuss our findings and give some ideas for the shortcomings and future work. Finally, in section 7, we give our conclusion of this work.

# 2 Related work

In this section, we first review the previous work of rumour stance classification, including original and subsequent work of utilizing structural information in conversations. We also summarize the development of rumour stance-related competitions and introduce the appearance of pre-trained language models.

## 2.1 Early work addressing rumours

Early work of rumour stance classification for tweets only performed 2-way classification of each tweet as *supporting* the rumour or *denying* the rumour, and the classifier is trained and predicted on single tweet units (Qazvinian et al., 2011). After this, some rule-based methods were suggested to improve the performance of this early work. In 2015, Liu et al. (2015) introduced a simple rule-based method to find whether there are positive or negative words in a tweet. However, these methods are not able to predict new and unseen rumours once rumours come out in a new format which is not described in these rules. Similar to this, Hamidian and Diab (2016) have examined how a model trained from seen tweets perform on classifying new tweets discussing the same rumour. These studies provided some ideas in dealing with rumour stance classification in Twitter but still rely on manual participation to some extent which may cost more time and energy. Zhao et al. (2015) shows that tweets that raise questions are likely to report controversial rumors, which motivated the proposal of a new approach considering conversational structure as assistance (Zubiaga et al., 2016a).

## 2.2 Early research utilizing conversational structure

A new way to take advantage of the conversion sequence observed in the conversation thread of the Twitter tree-structure was proposed by Zubiaga et al. (2016a). They separate the Twitter conversation into branches, and use branches as the input of the sequential models. In this work, two sequential classifiers were applied to addressing the rumour stance classification task: One is Linear-Chain CRF, regarding the conversational threads as separate linear branches; another is Tree-CRF, viewing the conversational threads as a tree structure. According to the experiment results, linear CRF and Tree-CRF both have significant advantages in macro-average F1 score comparing to traditional classifiers, including SVM and Random Forest, and the improvement varies from 9 percent to 24 percent. Besides, comparing with the similar sequential model Hawkes Processes which explored the same task before (Lukasik et al., 2016), the Tree CRF has much better performance in macro-average F1 score (Zubiaga et al., 2016a), which has averaged 10 percent improvement in every single event data set. The result shows the ability of Tree CRF to better predict labels in tasks where the label distribution is highly imbalanced, so it is advantageous to use the dialogue structure in the classification process (Zubiaga et al., 2018a). After the proposal of Zubiaga's original research of utilizing conversational structure, some subsequent work has explored other sequential models on the same task. BranchLSTM (Kochkina et al., 2017) was applied on RumourEval-2017 Task 8 Subtask-A and got top performance among all participating teams. These two methods are different in many aspects. The BranchLSTM relies on an LSTM instead of CRF, and it has different inputs with CRF: BranchLSTM uses the representation of a single tweet as input for each time step, while CRF trains and predicts a branch of tweets at the same time. Apart from this, BranchLSTM also adds seven extra features in preprocessing part to help to represent the tweets as vectors. These two methods have very close performance in Macro-average F1 score, with less than 1 percent difference. The authors that utilized conversational structure (Zubiaga et al., 2016a) continued their research in 2018 to explore how other sequential models perform on the same task (Zubiaga et al., 2018b), including Hawkes Processes, linear CRF, Tree CRF, and LSTM. In this research, different types of local and contextual features are compared. It shows that sequential classifiers that use discourse attributes and only use local features in social media conversations are better than non-sequential classifiers (Zubiaga et al., 2018b). More recent research is by Kumar and Carley (2019), it proposes a new method of representing dialogue as a binary constituency tree, which can effectively compare the features in the source post and its replies. By applying this new representation of conversation threads to Tree LSTM, the BCTree LSTM model outperforms the previous best model (Kochkina et al., 2017; Zubiaga et al., 2018b) by 12% and 15% on Macro-average F1 for rumor-veracity classification and stance classification tasks respectively. However, as BranchLSTM is the baseline for most subsequent competitions and researches, we still use BranchLSTM as our single model.

## 2.3 Development of rumour stance related competitions

Some semantic understanding competitions have been held to improve the performance of semantic analysis systems, and a series of interesting ideas were proposed in these competitions. SemEval is a series of evaluations of computational semantic analysis systems, and it also paid attention to rumour veracity detection. SemEval-2016 task 6 has addressed stance classification towards a target on Twitter (Mohammad et al., 2016). Task A considered a 3-way stance classification, and the dataset did not provide any relations between tweets, which treated

tweets as individual instances. RumourEval-2017 is a shared SemEval task to identify and deal with rumors and reactions to them (Derczynski et al., 2017), and Task 8 Subtask A is a stance classification task that tries to predict users' attitudes based on their tweets. Inspired by (Zubiaga et al., 2016a), BranchLSTM (Kochkina et al., 2017) was proposed to addressing task A by utilizing the conversational structure. They use the branches of Twitter conversations as the input of the LSTM, and add more structural information to the representation of the branches. It was the only method that considers the whole thread structure as a feature among all participating methods in RumourEval-2017, which got considerable performance. It was then regarded as a baseline system in RumourEval-2019 and was outperformed by 3 submitted systems (Gorrell et al., 2018).

### 2.4 Blossoming of pre-trained language models

In recent years, a series of pre-trained language models were proposed to address natural language processing tasks, some of them well updated the performances on nearly all NLP tasks (Devlin et al., 2018; Howard and Ruder, 2018; Radford et al., 2019). They are first trained on a huge corpus and then be saved as a pre-trained model(weights). BERT (Devlin et al., 2018) is one of the representative pre-trained language models that made great success, got state-of-the-art results in eleven nature language processing tasks. After that, a lighter version of BERT called DistilBERT (Sanh et al., 2019) was proposed, it reduces the size of a BERT model a lot while retaining almost all of its language understanding capabilities and being much faster. Having these state-of-the-art language models, it seems interesting to compare DistillBERT and BranchLSTM on stance classification task to see how they perform on the same task. As they have different model structure and differs in utilizing conversational structure, it is conceivable that they have different advantages and weakness when facing the same task. BranchLSTM is a sequential model which makes use of structural features, which could have a better understanding of relations among different posts; While DistilBERT is pre-trained with a huge amount of corpus, which can have a better understanding of words and sentences. In RumourEval-2019 (Gorrell et al., 2018), an ensemble of BERT models achieved the second best performing system in the rumour stance classification task.

## 3 Data

We use the dataset from RumourEval-2017 (Derczynski et al., 2017) Task 8 Subtask-A, which is expanded from the PHEME dataset by annotating additional conversation threads. The PHEME rumour dataset contains eight breaking news data sets (Zubiaga et al., 2016b), which was used by a series of research about using conversational structure in rumour stance classification.

We first make 5 random splits of the RumourEval-2017 Task 8 Subtask-A dataset as our dataset. Concerning the dataset splits with different sizes of conversations, we evaluate two types of deep learning models as single models and develop a series of ensemble models in addressing the rumour stance classification task. In the single-model experiment, we adapt the original BranchLSTM classification method to make it able to work on a flexible size of conversations. BranchLSTM (Kochkina et al., 2017) and DistilBERT (Sanh et al., 2019) are compared in the same splits of the dataset, and their differences in addressing the same task are explored in detail. As for the ensemble models experiments, we use the same dataset splits as single-model experiment.

## 3.1 Dataset description

Figure 2 is an example of the tree-structure Twitter conversation from the dataset, and it gives the definition of *depth* in a conversation. The source tweet is the root of the tree which locates on depth 0, and a series of replies can be nested on each other. As the discussion of the rumour goes further, the depth of the conversation thread also increases.

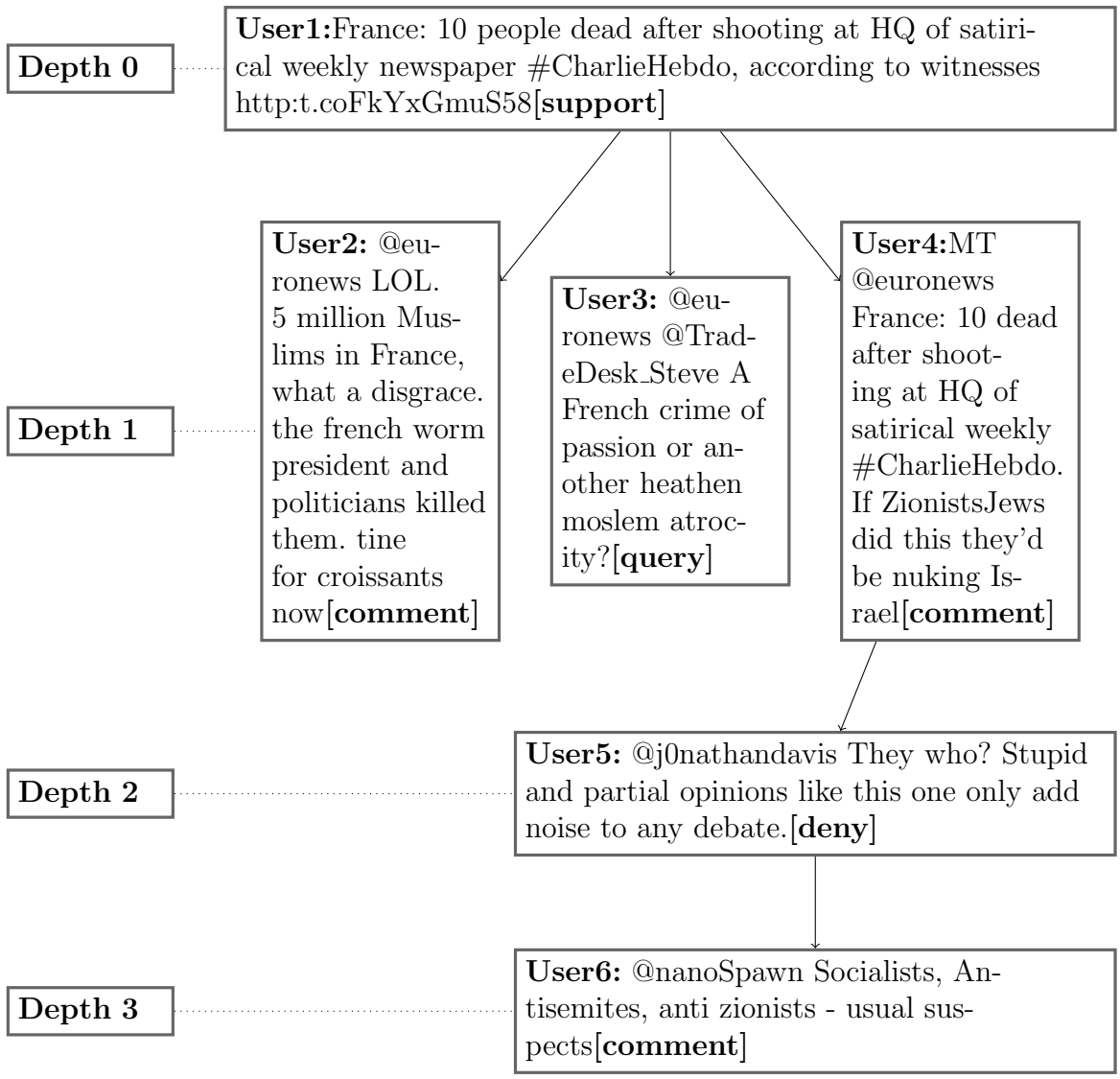



Figure 2: Example of a tree-structure conversation thread.

There are nine events in the PHEME dataset, but only eight events are used because tweets in the ninth event are in German. This dataset was collected, identified, and annotated with the method proposed by Zubiaga et al. (2016b), and the crowdsourcing method was proposed by Zubiaga et al. (2015). In RumourEval-2017 Task 8 Subtask-A, this dataset was expanded through annotating additional conversation threads. The annotated 28 additional threads for the test set, including 20 threads extracted from the same events as the training dataset, and 8 threads from two newly collected events. There are 297 rumourous threads and 4519 tweets in their training set, and 28 threads and 1080 tweets in the test dataset. The distribution of labels in the training and test dataset is summarised in Table1.

| | S | D | Q | C |
|---|---|---|---|---|
| Train | 910 | 344 | 358 | 2,907 |
| Test | 94 | 71 | 106 | 778 |

Table 1: Label distribution of training and test dataset in RumourEval-2017 Task 8 Subtask-A. S, D, Q, C represent "Support", "Deny", "Query" and "Comment" respectively.

## 3.2 Dataset split

The dataset is taken from RumourEval-2017 which is expanded from eight breaking news items in the PHEME dataset (Zubiaga et al., 2016b). The data has been used by a series of sequential models and shared tasks to explore rumour stance classification. To evaluate the models more accurately, we make 5 random splits of the RumourEval-2017 Task 8 Subtask-A dataset. Different from the dataset in RumourEval-2017 which only has one spit, each split of our development dataset and test dataset is extracted from their training dataset. We split up the RumourEval training dataset 5 times stratified, each split creates one test dataset and one develop dataset which has the same number of threads as the RumourEval test dataset and develops dataset. In order to keep the completeness of the whole dataset, and makes the training dataset diverse, the rest of the threads of each split for the training dataset are combined with the original test dataset to be used as a new training dataset.
Figure 3 shows the 5 random splits process to original dataset. The grey, orange and red areas represent training, development, and test data respectively. First, we shuffle the conversations in the original training dataset to reduce the bias of different conversation sizes, because different conversations have a different number of tweets and depth. Then, the training dataset is split 5 times based on the number of development and test conversations. From split 0 to split 4, the proportion of conversations in the training, development, and test dataset is always 272:25:28, which is the same as the proportion in the original dataset. Therefore, the dataset is not split based on the number of tweets or branches but uses conversation as the basic unit.

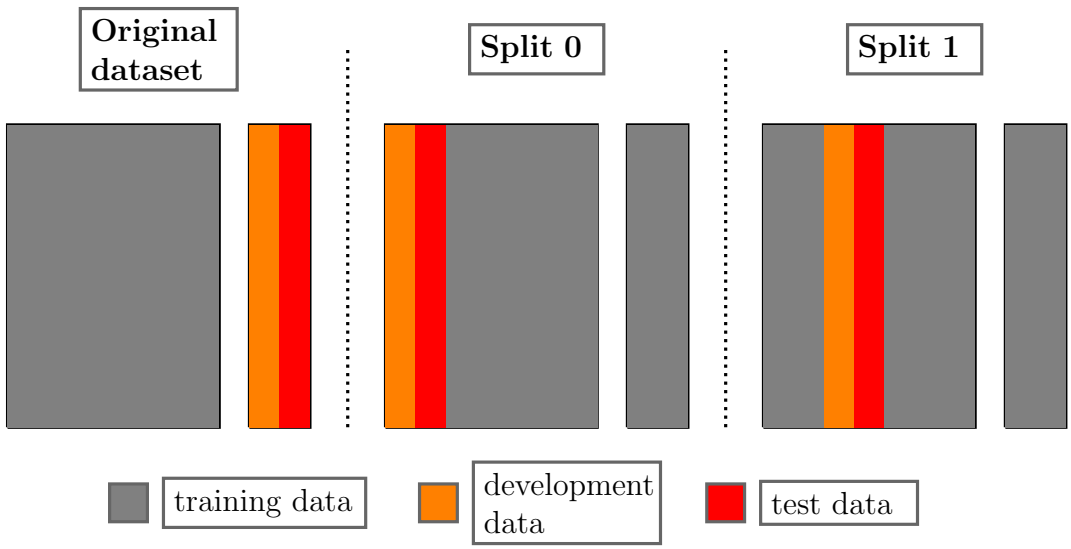



Figure 3: The split of dataset

# 4 Methods

We design a single-model method and an ensemble-model method for our search. In the single-model experiment, we adapt the original BranchLSTM classification method to make it be able to receive a more flexible input format, which successfully works on 5 splits of the dataset. We apply another single model DistilBERT on the same split of the dataset as BranchLSTM by using different preprocessing methods and conduct a series of comparisons and analyses.
By changing the output format of trained deep learning models, they can predict each tweet's expected probability towards each stance. Therefore, in the ensemble method experiment, we design a series of voting classifiers to classify each tweet's feature built with the probabilities predicted by two deep learning models. We build up a pipeline for the ensemble classification system, the complete pipeline consists of a series of steps, including original dataset split, conversation preprocessing, stance probabilities prediction, probabilities preprocessing, feature dataset construction, voting classification, and evaluation.
The ensemble method experiment uses the same dataset split as single-model experiment, and also uses the same data preprocessing methods as single-model experiment because the two deep learning models in the single-model experiment are also used in predicting the stance probabilities. Concerning the dataset of tweets' probability features, we design two one-step and two two-step voting classifiers. We explore linear SVM and linear CRF two kinds of statistical classifiers in one-step voting classification and use CRF-SVM and CRF-CRF as two combinations of classifiers in two-step voting classification. Similar to the single-model experiment, the voting classifiers also differ in whether to use conversation structure. CRF voting classifier is a sequential model which can utilize structural information, but SVM treats each tweet's probability feature as a single unit. The ensemble method architecture and pipeline working process will be introduced in this section, and related experiment details will be introduced in later sections.
In subsection 4.1, we introduce the classification systems of two single models and describe the method of adapting the BranchLSTM script to receive a more flexible input format. In subsection 4.2, we introduce the ensemble model architecture and classification pipeline with an illustration. In the last 4.3 subsection we introduce the strategy of voting classification, including the preprocessing method for probabilities.

## 4.1 Single-model methods

We adapt the BranchLSTM (Kochkina et al., 2017) classification method in RumourEval-2017 task 8 Subtask-A as our first single model and use DistilBERT (Sanh et al., 2019) with different preprocessing steps as our second single model. We will first introduce the original BranchLSTM and the structural features that it uses, and then introduce adapted BranchLSTM as well as DistilBERT.

### 4.1.1 Original BranchLSTM Method

Figure 4 shows an example of the branches in a tree-structure Twitter conversation thread. The source tweet is the root of the tree that receives a series of replies, and these replies can be nested on each other. A branch starts at the source tweet and ends at the last reply on this branch. The sequential models in both single-model experiment and ensemble method experiment need to use the tree structure in conversations.

The model architecture of BranchLSTM is the same as LSTM, which is a sequential model that receives an input at each time step. The name "BranchLSTM" means the neural network uses layers of LSTM units to process the whole branch of tweets, thus incorporating structural information of the conversation. Through preprocessing to each tweet, the input for BranchLSTM at each time step is formatted as a fixed-length vector. At each time step, the output of the LSTM unit is recorded as a label attached to the tweet belonging to the branch. To process the whole branch of tweets with layers of LSTM, each branch is represented by a list of tweet features.
In the original BranchLSTM method, each branch is used as a basic sample during the training process. The whole branches in the training dataset are represented as an array of 2d arrays, where each 2d array represents a branch. Every 2d array representing a branch is initialized with a fixed shape, which the row number is the maximum branch length in this dataset, and the column number is the number of features for each tweet.
In the preprocessing steps of original BranchLSTM[1], non-alphabetic characters are removed, all words are converted to lower case, and texts are tokenized. A tweet is represented with an average of word2vec representations of individual words, and then concatenating with the additional features to a fixed-length vector. All the preprocessing actions are implemented by Python 2.7 with the NLTK package. Once tweet texts are pre-processed, the following features are extracted:

- **Word vectors:** A word2vec (Mikolov et al., 2013) model pre-trained on the Google news dataset(300d) is used, implemented by gensim package
- **Tweet lexicon:** (1) count of negation words and (2) count of swear words.
- **Punctuation:** (1) presence of a period, (2) presence of an exclamation mark, (3) presence of a question mark, (4) ratio of capital letters.
- **Attachments:** (1) presence of a URL and (2) presence of images.
- **Relation to other tweets:** (1) Word2Vec cosine similarity wrt source tweet, (2) Word2Vec cosine similarity wrt preceding tweet, and (3) Word2Vec cosine similarity wrt thread.
- **Content length:** (1) word count and (2) character count.
- **Tweet role:** whether the tweet is a source tweet of a conversation.

### 4.1.2 Adapted BranchLSTM method

In the original BranchLSTM script, the maximum branch length is assigned with a fixed value regarding the original dataset, which could cause errors while training on a new dataset with a different maximum branch length. Because the array of branch arrays is saved as a numpy file, to solve this, we use a regular expression package to extract the shape of the branch array recorded at the beginning of the numpy file and assign the second value of 3d array shape to the maximum branch length before training. After that, the adapted BranchLSTM method could work well on a flexible dataset with different size of conversations.[2]

[1]The original BranchLSTM : `https://github.com/kochkinaelena/branchLSTM`

[2]The adapted BranchLSTM is published in GitHub: `https://github.com/blcuwc/branchLSTM_Cross_Validation`

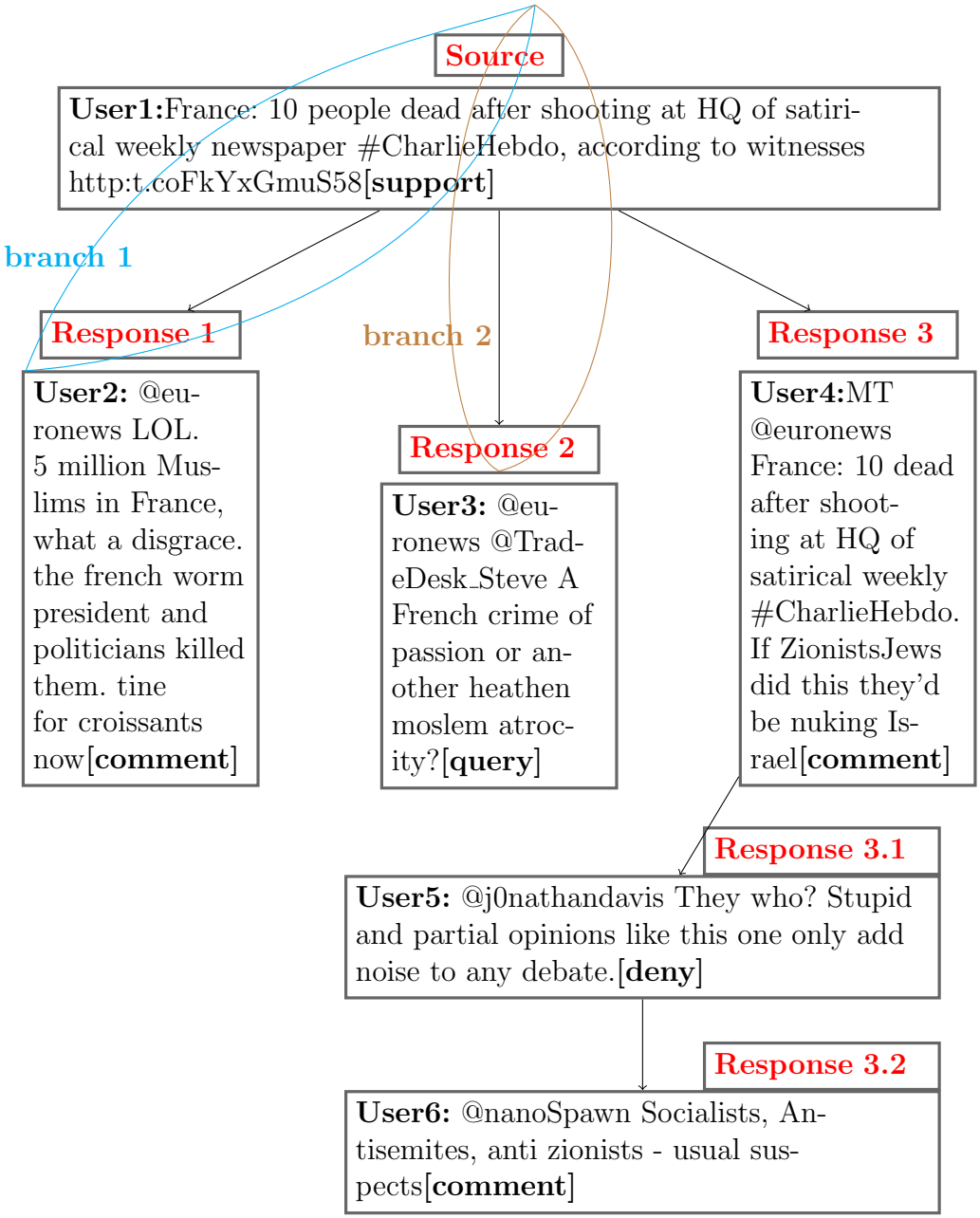



Figure 4: Branches in a tree-structure conversation thread.

The preprocessing step in adapted method is the same as the original method. However, after random splits to the original dataset, the training dataset's maximum branch length will change because of the different sizes of conversations. This leads to the failure of training because the maximum branch length is fixed in the original script. To match the preprocessed data correctly, we adapt the original BranchLSTM scripts for receiving changeable preprocessed data. By recognizing the shape of preprocessed data automatically, the modified script could train and predict various types of conversations. Therefore, the baseline BranchLSTM classification method could work well on a 5 random split dataset successfully.

The BranchLSTM is implemented by Python libraries *Theano* (Bastien et al., 2012) and *Lasagne* (Dieleman et al., 2015). In the original paper, the optimal set of hyperparameters were determined through testing the performance of BranchLSTM on the development set for

different parameter combinations. In our single-model experiment, we adopt the best hyperparameters from the original paper directly. The hyperparameters for the LSTM model and training process are as follows:

- **number of LSTM units:** 100;
- **number of LSTM layers:** 2;
- **number of dense ReLU layers:** 2;
- **number of dense ReLU units:** 500;
- **number of epochs:** 30;
- **learning rate:** 0.001;
- **mini-batch size:** 100;
- **L2 regularization:** 0.0;

#### 4.1.3 DistilBERT method

In contrast to BranchLSTM, DistilBERT is a pre-trained language model which has a good text understanding ability but does not consider conversational structure. The DistilBERT is upgraded from BERT (Devlin et al., 2018), which made much state-of-the-art performance in NLP tasks. The object of pre-training is to learn a powerful language representation capability, and a pre-trained model can be applied to downstream tasks through fine-tuning all pre-trained parameters. DistilBERT is more lightweight and fast than BERT, and needs less resource and time to be pre-trained and can retain most of its language understanding capabilities.
In the DistilBERT single model, we use the same dataset split as in the BranchLSTM experiment. We use Python library *ktrain* to implement DistilBERT, including the preprocessing step, training, and prediction process. The conversation preprocessing step in DistilBERT is different from BranchLSTM. In BranchLSTM, each branch of tweets is considered as a whole part for training and predicting, because it utilizes the information of conversation structure. However, DistilBERT does not consider the relationship between tweets and does not need to keep the complete conversation structure during data preprocessing. Therefore, all the conversations are broken into single tweets and then be fed to preprocessing function in the ktrain package. The preprocessing step is simply finished by *texts_from_array()* function in *ktrain* package, which converts the raw texts to training data array and development data array. The related hyperparameters for the training process are as follows, and they are chosen from the most common hyperparameter settings:

- **batch size:** 6;
- **learning rate:** 0.00003;
- **number of epochs:** 4;

## 4.2 Ensemble methods

The ensemble modeling means combining multiple base models to obtain a better predictive performance than any of the base models. There are usually two families of ensemble models, one is averaging methods while another is boosting methods. In averaging methods, every single base model is built independently and the prediction is averaged to reduce variance; However, in the boosting method, base model are constructed sequentially and attempts to reduce the bias of the combined estimator.

We use the voting classifier which is one of the boosting methods for this task (Bartlett et al., 1998). Here, BranchLSTM and DistilBERT are built as the first-step base models in our ensemble method, and voting classifiers are built as the second-step base models which try to reduce the bias of the combined deep learning models. Figure 5 shows the ensemble method classification pipeline. In the above part of the pipeline, after 5 random splits to the original dataset, 5 splits are preprocessed respectively and fed to BranchLSTM and DistilBERT. By changing the output format of two deep learning models, each deep learning model will predict each tweet's expected probabilities towards each stance. For every tweet, the sum of every stance probability is 1. Having the predicted probabilities from deep learning models, the below part of the pipeline shows the process of connecting probabilities to an eight-length vector. This vector is the feature of the corresponding tweet and will be the input for the subsequent voting classifier.

The idea of using a voting classifier is to take both of their advantages in predicting rumour stance. As BranchLSTM is a sequential model and can make use of conversational structure, and DistilBERT has more advantages in understanding the text itself, we hope to combine their predicted probabilities and let voting classifiers justify.

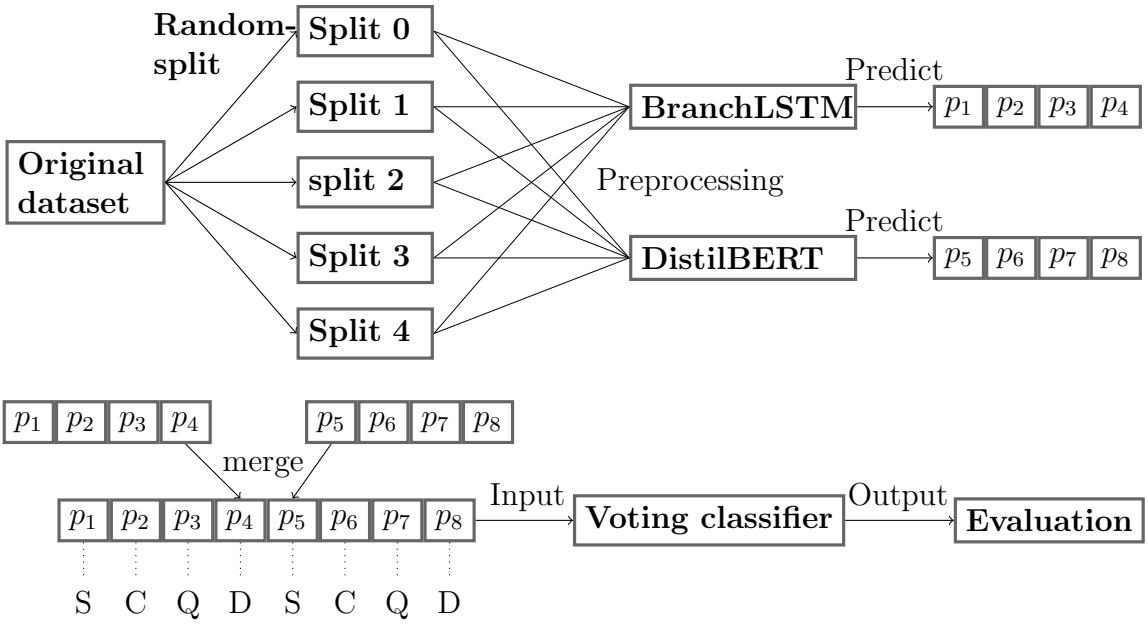



Figure 5: The ensemble method classification pipeline. S, D, Q, C represent "Support", "Deny", "Query" and "Comment" respectively.

## 4.3 Voting strategies

There are two strategies of voting classification: hard voting and soft voting. In hard voting (also called majority voting), the predicted class label for a particular tweet is the class label that represents the majority of the class labels predicted by each classifier. It is rather useless in the case of two classifiers because it can't decide if two classifiers have different predictions. We use the soft voting strategy in our method, and it offers a more reasonable solution. In soft voting, classifiers can provide weights to each label. After the weights are assigned, the predicted class probability of each classifier will be collected, multiplied by the classifier weight, and then averaged. Then the final class label is derived from the class label with the highest average probability.

Through changing deep learning models' output from predicted labels to probabilities, voting classifiers can receive the probabilities as input features, and predict the corresponding stance of rumour. As we have four labels "*Support*", "*Deny*", "*Query*" and "*Comment*", and each label should be equal while being classified, so we assign each label a 25% weight in soft voting.

In this method, we develop four voting classifiers, including two one-step voting classifiers and two two-step voting classifiers. The two one-step voting classifiers are Support Vector Machine and linear CRF [3] (Lafferty et al., 2001), and the two-step classifiers are CRF-SVM and CRF-CRF. The voting classifiers follow the deep learning models and use their output probabilities as input. As a result, voting classifiers also enjoy the same 5 random splits as deep learning models.

The one-step voting classifier is rather simple, which is either an linear SVM classifier or a linear CRF classifier. However, there is a difference between them in classifying these probabilities. SVM treats each probability vector and its corresponding label as a unit, and linear CRF is trained and predict in all probabilities of every branch of the conversation thread. This is the difference in their classifying strategy, which can cause a difference in their classification results. We use a simple grid search algorithm for both voting classifiers, to find better parameters in classification, which can contribute to the performance.

The two-step voting classification process is illustrated in figure 6. In both two-step voting classifiers, the first classifier is linear CRF but not linear SVM. This is because of the difference between linear CRF and linear SVM in the training and predicting process. Linear SVM uses a single tweet as the basic unit for classification, while linear CRF trains and predicts a complete branch of tweets at each time. Therefore, linear CRF needs complete branches for training which always contain four kinds of labels, but linear SVM can be trained and predict on only three kinds of labels. Once we use linear SVM in the first step for two-step voting classification, linear CRF can not be applied in the second step because the complete conversation structure will be broken. On the contrary, after using linear CRF for the first step classification, linear SVM can be trained with only three labels in the second step, which will also only predict three small classes in the second step.

Because of the imbalance of the class labels where many "comment" occurrences, in the first step, the true labels of tweets are all modified to *comment* and *non-comment*. After the first step classification, for CRF-SVM, the predicted *non-comment* tweets will be sent to the second voting classifier. However, in the CRF-CRF voting classifier, the second CRF classifier also predicts four labels on a complete dataset as in the first step. This is because the training of linear CRF needs the complete structure of conversations, thus it can only be trained with

[3] We use sklearn-crfsuite package: `https://github.com/TeamHG-Memex/sklearn-crfsuite/`

complete labels but not part of the classes. The second step could help to re-classify those tweets misclassified from comment to non-comment.

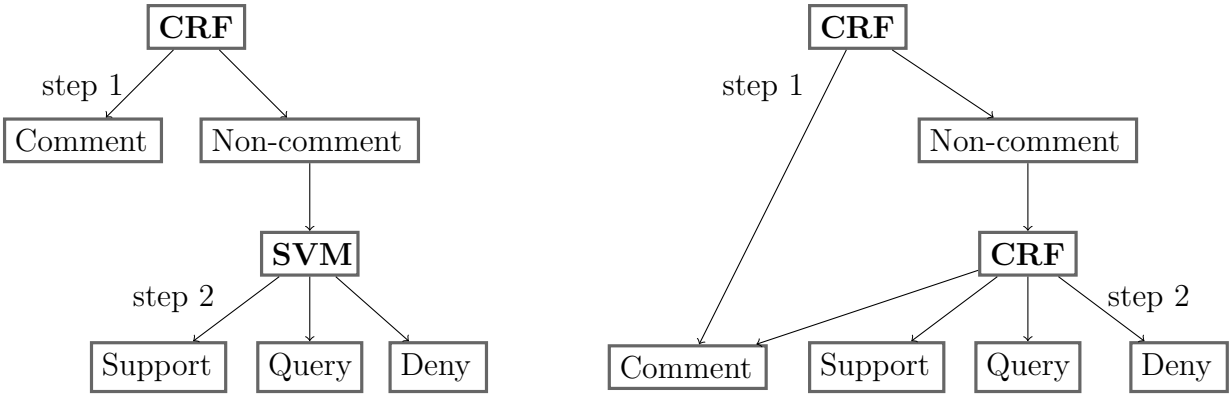



Figure 6: Two-step voting classification structure.

# 5 Experiments

In this section, we introduce the experiment settings and experiment results of single models and ensemble models. In subsection 5.1, we give the detailed experiment settings of both single-model experiment and ensemble experiment. In subsection 5.2 and 5.3, we introduce and analyze the experiment results of the single-model method and ensemble method respectively. In the analysis both single-model and ensemble experiment result, we use the same evaluation metrics, including macro-average F-score, precision, recall, and F-score for per class. The macro-average F-score is the most important evaluation method in this research because our dataset is small and the labels are imbalanced, while the evaluation based on the macro-average F-score takes into account the classifier's ability to produce an output that is more suitable for the class distribution. We also give the micro-average and accuracy of classification results as a reference.
For some specific comparisons, we analyze their error cases to find out their similarities and differences in this task. We also show confusion matrix and depth analysis in some cases, to better understand the distribution of the error cases in a different class and different depth.

## 5.1 Experiment settings

In subsection 5.1.1, we introduce the single-model experiment code structure and potential problem during the experiment. In subsection 5.1.2, we introduce the voting classifiers' parameters search space and optimization strategy in detail.

### 5.1.1 Single-model experiment

There are three Python scripts in the original BranchLSTM classification method, the first is the preprocessing script, the second is the training and predicting script, and the last one is for results evaluation and analysis. It is rather simple to run the original BranchLSTM classification pipeline. However, after 5 random splits to the original dataset, we need to run the pipeline 5 times and every time the model needs to be initialized.

The code structure in the DistilBERT single-model experiment is similar to BranchLSTM, the difference is that DistilBERT merges preprocessing and training/predicting in one script.

### 5.1.2 Ensemble experiment

There are four sets of voting classifiers in the ensemble experiment, with two one-step and two two-step voting classifiers. Although the number of voting classifiers seems a lot, we only have two kinds of basic classifiers which are linear SVM and linear CRF. All sets of the voting classifiers are one of these two models or combined with two of these two models.
To analyze the contribution of deep learning models, linear kernel and one-versus-one strategy are used in SVM. To achieve the better performance of the voting classifier, the penalty parameter $C$ is optimized by grid search. The search space of C is defined in table 2. The search space capacity of C is 9, thus we have 9 combinations of parameters. As we have one-step voting and two-step voting which both contain SVM classifier, SVM in both ensemble models use the same configuration.

| Parameters | Minimum value | Maximum value | step size |
|---|---|---|---|
| **C** | 0.1 | 1 | 0.1 |

Table 2: Linear SVM parameters' search space

The gradient descent using the L-BFGS method is applied in the training process of CRF, and the coefficients for L1 and L2 regularization are optimized with random search. The search space of coefficients for L1 and L2 regularization is defined with the exponential continuous random variable, with scales 0.5 and 0.05.

## 5.2 Single-model Experiment Results

In following subsections, we first compare two single models' performance from an overall view, and then give an analysis by depth. Finally, we conduct a dependent error analysis for their error cases.

### 5.2.1 Model performance comparison

To compare the performance of two single models, we give their macro and micro average performance in table 3, and accuracy and F-score for individual class in table 4. In table 3, DistilBERT has a 4.8 percent higher macro-average F-score than BranchLSTM. As we use macro-average F-score as the most important metric, DistilBERT is believed to have a better performance in this dataset. However, by looking at micro-average metrics in the table, there is no obvious difference between them. In table 4, the difference between their F-score in "Deny" class is large, but other difference are very limited. To verify whether there is a significant difference between BranchLSTM and DistilBERT is their performance, we use Wilcoxon test (Wilcoxon, 1992) to test both performance in two tables. The $P$ value is much bigger than 0.05, thus we accept the null hypothesis that there is no significant difference between BranchLSTM and DistilBERT's performance. Therefore, the advantage of DistilBERT over BranchLSTM is limited.
Figure 7 shows the BranchLSTM's and DistilBERT's confusion matrix for test dataset predictions. The overall predictions are quite similar, but there are also some differences. The figure

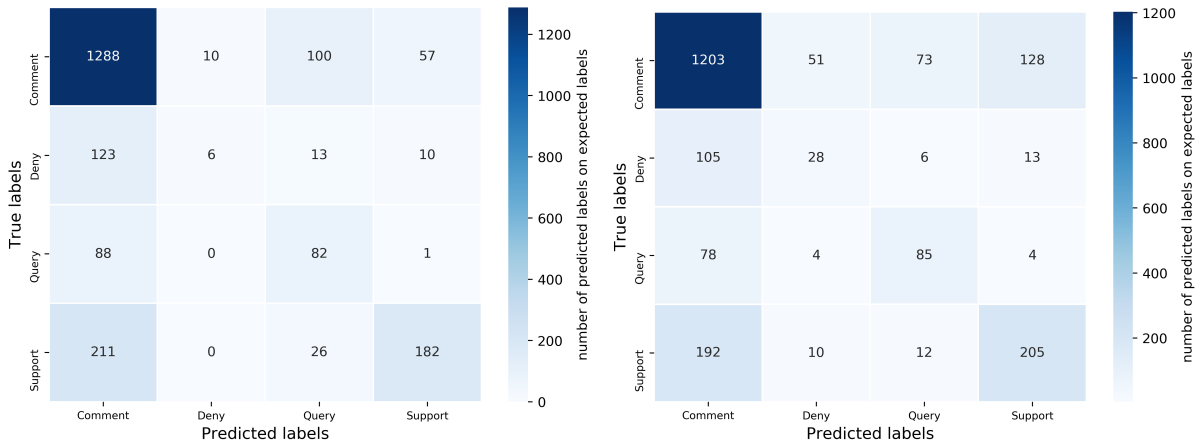
| Model \ Metrics | Macro-average | | | Micro-average | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-score | Precision | Recall | F-score |
| BranchLSTM | 0.525 | 0.459 | 0.457 | 0.704 | 0.704 | 0.704 |
| DistilBERT | 0.530 | 0.497 | **0.505** | 0.688 | 0.688 | 0.688 |

Table 3: BranchLSTM and DistilBERT average performance.

| Model \ Metrics | Accuracy | F-score per class | | | |
|---|---|---|---|---|---|
| | | S | D | Q | C |
| BranchLSTM | 0.704 | 0.809 | 0.072 | 0.405 | 0.541 |
| DistilBERT | 0.688 | 0.788 | **0.229** | 0.469 | 0.533 |

Table 4: BranchLSTM and DistilBERT's accuracy and F-score per class. "S", "D", "Q", "C" represent "Support", "Deny", "Query" and "Comment" respectively.

shows that BranchLSTM has more *comment* tweets correctly classified, while DistilBERT has more other three classes of tweets correctly classified. At the same time, BranchLSTM misclassified more true other three classes of tweets to *comment* than DistilBERT, while DistilBERT misclassified more *comment* tweets to the other three classes. As this dataset is not balanced, we can know that BranchLSTM has better capability in classifying big class *comment*, while DistilBERT is better at classifying other small classes.




(a) BranchLSTM Confusion matrix (b) DistilBERT Confusion matrix

Figure 7: Comparison of BranchLSTM's and DistilBERT's confusion matrix for test dataset predictions

### 5.2.2 Performance at different depths of the threads

Figure 2 illustrates the definition of the depth in a tree structure, and this will be used to help to evaluate the model's performance from another perspective. Table 5 shows the macro-average F-score and accuracy of BranchLSTM and DistilBERT on different depths. The left part of the table is the number of tweets on different depth, and the right part is the performance of the two models. Most of the source tweets are *support*, which is imbalanced in the dataset

and more likely to cause discussions around it. There is the most number of tweets in depth 1, and the number of tweets gradually goes down with the increase of depth until depth 5.
The performance of BranchLSTM and DistilBERT differs a lot on different depths. As for macro-average F-score, BranchLSTM only has a higher macro-average F-score than DistilBERT on depth 0, but DistilBERT performs better on all other depths except depth 5. On the contrary, DistilBERT has higher accuracy than BranchLSTM on depth 1, while BranchLSTM has higher accuracy than DistilBERT on other depths. As we use macro-average F-score as the main metric, this table proves that DistilBERT performs better than BranchLSTM in most situations. It is also believed that DistilBERT performs better than BranchLSTM on the balanced dataset. For example, on depth 1 with more numbers of small class tweets, DistilBERT has both higher macro-average F-score and accuracy than BranchLSTM. Oppositely, on depth 0 and depth 5 with very imbalanced class labels distribution, DistilBERT performs not as good as BranchLSTM on both macro-average F-score and accuracy.
Therefore, we can conclude that DistilBERT has advantages in addressing tweets with more balanced labels comparing to BranchLSTM, and has better performance than BranchLSTM on small classes. However, BranchLSMT performs better in predicting the large class.

| **Depth** | **tweets** | **S** | **D** | **Q** | **C** | **BranchLSTM** | | **DistilBERT** | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Accuracy | MacroF | Accuracy | MacroF |
| 0 | 140 | 132 | 4 | 0 | 4 | 0.9430 | 0.6220 | 0.9142 | 0.4698 |
| 1 | 1271 | 220 | 88 | 110 | 853 | 0.6502 | 0.4136 | 0.6564 | 0.4762 |
| 2 | 276 | 28 | 14 | 26 | 208 | 0.7398 | 0.2534 | 0.6900 | 0.3312 |
| 3 | 157 | 10 | 10 | 5 | 132 | 0.8460 | 0.2918 | 0.7486 | 0.3622 |
| 4 | 96 | 8 | 8 | 11 | 69 | 0.7444 | 0.2594 | 0.6992 | 0.3198 |
| 5 | 52 | 5 | 2 | 3 | 42 | 0.8322 | 0.4660 | 0.7174 | 0.4602 |
| 6+ | 205 | 16 | 26 | 16 | 147 | 0.7618 | 0.3928 | 0.7422 | 0.4832 |

Table 5: Number of tweets per depth and performance at each of the depths of BranchLSTM

### 5.2.3 Error analysis

From the comparison between BranchLSTM and DistilBERT experiment results, we know that they have close performance and respective advantages. To uncover how the models can best be combined and whether they have the potential to compliment each other, we conduct an in-depth error analysis for two models.
As the multi-classification task can be regarded as a set of binary classification problems, evaluation methods in binary classification can be used to analyze error cases here. False-positive and false-negative are two types of errors in binary classification. False-positive is defined as the proportion of actual negatives that are wrongly identified as positives, and false-negative is defined as the proportion of actual positives that are wrongly identified as negatives. In this research, each class's false-positive error cases are those tweets wrongly classified from the other three classes to this class, and each class's false-negative error cases are those tweets wrongly classified from this class to the other three classes. In this and also later error analysis part, we will count and analyze two model's unique and common error cases for each class will be counted. However, we only consider false-negative error cases, because

they show the model's disadvantage in misclassifying tweets belonging to this class. Besides, it is complex to analyze all kinds of error cases.
The goal of this error analysis is to discover the similarities and differences between BranchLSTM's and DistilBERT's error cases and to conclude their features in misclassifying tweets. There is a difference between error analysis and confusion matrices, confusion matrix only shows the distribution of error cases in different classes, but error analysis concern the content of misclassified tweets.
Table 6 shows the common and unique error cases of BranchLSTM and DistilBERT on three classes. From this table, it can be seen that the number of common errors for *support* and *deny* is several times of the sum of unique errors, and BranchLSTM has several times error cases than DistilBERT. That means two models have many similarities in misclassifying *support* and *deny* to the other three classes, but BranchLSTM is more likely to misclassify other classes to comment than DistilBERT to some extent. For *comment* error cases, the sum of unique mistakes is much more than common mistakes, and DistilBERT has nearly twice amount of mistakes than BranchLSTM. This shows that there is not a lot of overlap in the mistakes, which means there exists an obvious difference in misclassifying *comment* class between them. Combining the table with the confusion matrices, they prove that DistilBERT is not as good as BranchLSTM in classifying big class while facing imbalanced dataset, and it is more likely to make unique mistakes in the big class.

| **Class** / **Error type** | **comment** | | **support** | | **deny** | |
|---|---|---|---|---|---|---|
| | LSTM | BERT | LSTM | BERT | LSTM | BERT |
| **Unique** | 74 | 159 | 40 | 17 | 26 | 4 |
| **Common** | 93 | | 202 | | 125 | |

Table 6: Single models' unique and common error cases

It is also necessary to analyze error cases from support to denying and from denying to support which containing bright attitudes. Once a tweet is misclassified like that, the subsequent rumour verification task can be deeply influenced, even cause the opposite result for rumour classification.
Through checking two models' error cases from denying to support, we know that BranchLSTM has 10 denying tweets misclassified to supporting in total, while DistillBERT has 13, and there are 7 in common. We check 9 representative error cases, including 3 BranchLSTM unique mistakes, 3 DistilBERT unique mistakes, and 3 common mistakes. In BranchLSTM unique error cases, these tweets express negative attitudes without using negative words. In DistilBERT unique error cases, these tweets use negative words to deny the original post. This can be understood as DistillBERT has a better understanding of the text meaning than BranchLSTM, but it is not sensitive to negative words which can express the negative attitudes comparing to BranchLSTM.
The confusion matrices in figure 7 show that only DistillBERT has error cases from support to deny but BranchLSTM doesn't. Thus, here we show some representative unique error cases which are misclassified from denying to support by DistilBERT.

- 0. Banksy gets it right. "@juliamacfarlane: #Banksy's take on #CharlieHebdo. #JeSuisCharlie #NousSommesCharlie http://t.co/Y51AAQ6DRm"
- 1. Store owner told @FOX2now Monday that there was a theft, but said it was not MichaelBrown; said it was someone else #Ferguson

- @TroyBramston Source from Ray Hadley shows confirmed same report of gunman claiming there are four packages around Sydney

As for the no.0 error case, only the first sentence is written by the author, and the rest of the tweet is retweet content. The original sentence is much shorter than the retweeted content, and this is probably the main reason for DistilBERT's unique mistake. As for second and third error cases, it's hard to judge their true labels because we need to check the original post to know what they are talking about. This is also confusing to DistilBERT because it lacks context information. This could probably be the reason that only DistilBERT has such errors while BranchLSTM doesn't. The DistilBERT has advantages in understanding the text content while BranchLSTM is more good at utilizing thread structure, this difference makes it harder for DistillBERT to classify this kind of text, but BranchLSTM can perform better.

## 5.3 Ensemble model Experiment Results

Having the error analysis result between two single models, we know that they have their advantages in addressing rumour stance classification task. BranchLSTM could make use of the structural information to understand context, and is better in classifying large categories. DistilBERT is good at understanding text itself and performs better in a more balanced dataset. To explore whether they can cooperate and aggregate their advantages, we design and test a series of ensemble models to combine them with voting classifiers. By applying two one-step and two two-step voting classifiers to the combination of two single models, we get abundant experiment results. In this section, we show and analyze these experiment results in 3 subsections. At the beginning of each subsection, we introduce the motivation for this analysis. Through various analyses of ensemble models' results, we hope to investigate the feasibility and effectiveness of constructing ensemble models with BranchLSTM and DistilBERT to address rumour stance classification task and to find out the difference between sequential and non-sequential, one-step and two-step voting classifiers.

### 5.3.1 Performance best ensemble model

After adding a series of voting classifiers to the combination of BranchLSTM and DistilBERT, the most pressing question is whether the ensemble models help to improve the performance on rumour stance classification. In this section, we show the overall performance of all ensemble models and compare the performance best ensemble model with single models. We also analyze their macro-average F-score by the depth and analyze the similarities and differences of error cases between single models and the best ensemble model.
Table 7 shows all ensemble models' overall performance, it is believed that all models perform roughly equally. As for the other two macro-average metrics, they also perform very similarly, which can also be seen as roughly equally. As a result, choosing the best ensemble is meaningless because no one ensemble model could perform better than others.
Comparing them with single models, we find out that all ensemble models have higher macro-average F-score than BranchLSTM but lower than DistilBERT. We think ensemble models could successfully merge both the features of BranchLSTM and DistilBERT, but can not avoid the shortcomings of single models.
Figure 8 shows the comparison of macro-average F score per depth between single models and one-step SVM ensemble model. In the comparison between BranchLSTM and SVM, SVM has the same trend as BranchLSTM except for source tweets; in the comparison between

| Model \ Metrics | Macro-average | | |
|---|---|---|---|
| | Precision | Recall | F-score |
| SVM | 0.531 | 0.470 | **0.489** |
| CRF | 0.532 | 0.468 | 0.488 |
| CRF+SVM | 0.515 | 0.473 | 0.486 |
| CRF+CRF | 0.531 | 0.467 | 0.486 |

Table 7: All ensemble models' performance on test dataset

DistilBERT and SVM, SVM is almost the same as DistilBERT from depth 0 to depth 4. The macro-average F score of SVM can be seen as the combination of two deep learning models' performance because it has many similarities with both single models in certain depth.
Comparing BranchLSTM and one-step SVM, SVM performs better than BranchLSTM in all depths except for source tweets. The advantages of SVM over BranchLSTM in macro-average F-score could be benefit from DistilBERT, according to the non-source depth performance in the graph. However, the disadvantages to BranchLSTM in source tweets might also be inherited from DistilBERT, because it has the same performance as DistilBERT in source tweets.
For DistilBERT and one-step SVM, SVM almost has the same performance as DistlBERT except depth 5 and depth 6+. The performance of SVM in depth 5 and depth 6+ could be regarded as the inheritance of BranchLSTM, which also has an increase from depth 4 to depth 5 and a decrease from depth 5 to depth 6+.
It appears that the ensemble model takes advantages and disadvantages from both single models, and these features reflected in the comparison of macro-average F score between the best ensemble model and single models. On one hand, the SVM has a better performance than BranchLSTM in deeper depth; on the other hand, it keeps the insufficient performance of DistilBERT for source tweets.

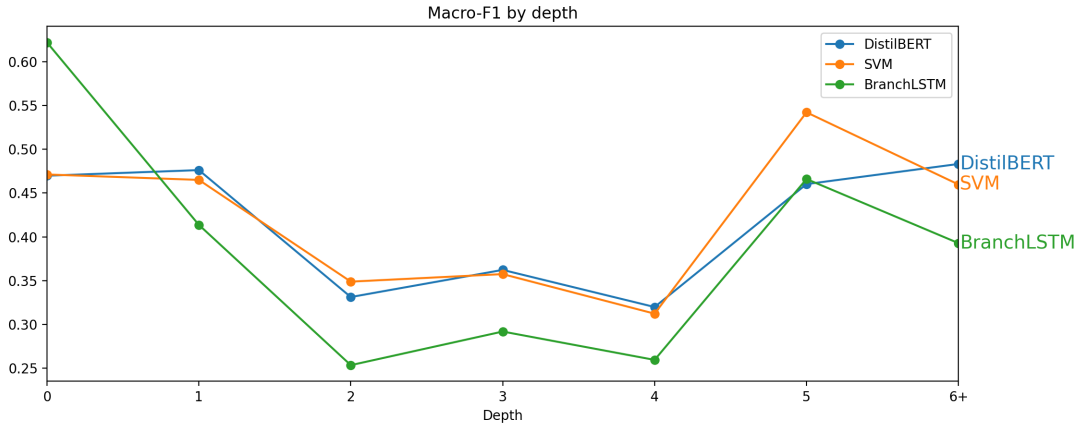



Figure 8: Comparison of single model and one-step SVM ensemble model's Macro-F1 scores by depth of tweet.

Table 8 and table 9 are comparison of unique and common error cases between single models and one-step SVM ensemble model. As for the comparison between BranchLSTM and SVM, there are not many different unique error cases in *support* and *deny* classes, and common error cases are multiple times of their unique error cases. On the contrary, SVM has more unique

*comment* error cases than both BranchLSTM and their common error cases. Therefore, they have similar disadvantages in misclassifying small classes' tweets, but one-step SVM ensemble model is more likely to misclassify *comment* tweets to other classes than BranchLSTM.
As for DistilBERT and SVM in table 9, the situation is similar in small classes but different in big class. For *support* and *deny*, DistilBERT and SVM have many common error cases like BranchLSTM and SVM. Different from table 8, they have more common error cases in *commnet* class. This means DistilBERT and SVM have similar disadvantages in misclassifying small classes' tweets than big class, but SVM has more similarities in misclassifying large categories with DistilBERT than BranchLSTM.

| **Class** / **Error type** | **comment** | | **support** | | **deny** | |
|---|---|---|---|---|---|---|
| | LSTM | SVM | LSTM | SVM | LSTM | SVM |
| **Unique** | 90 | 157 | 42 | 21 | 25 | 65 |
| **Common** | 82 | | 200 | | 126 | |

Table 8: BranchLSTM and one-step SVM ensemble's unique and common error cases

| **Class** / **Error type** | **comment** | | **support** | | **deny** | |
|---|---|---|---|---|---|---|
| | BERT | SVM | BERT | SVM | BERT | SVM |
| **Unique** | 88 | 70 | 21 | 23 | 6 | 8 |
| **Common** | 169 | | 198 | | 123 | |

Table 9: DistilBERT and one-step SVM ensemble's number of error cases

### 5.3.2 Sequential vs non-sequential voting classifier

To investigate the difference between sequential voting classifiers and non-sequential voting classifiers, we compare and analyze CRF's and SVM's classification results in this section. We hope to figure out to what extent can structural information help classifying the predicted probabilities from two single deep learning models. Through error analysis of two voting classifiers, we conclude the advantage and disadvantages of their voting strategy while receiving probabilities from deep learning models.
Table 10 shows the performance of the one-step SVM ensemble model and one-step CRF ensemble model in overall and per class metrics. From this table, we can see that they have very close performance in every macro-average and single class metric. The gap between any two of them is less than 2 percent. Therefore, SVM and CRF have almost the same performance in classifying the concatenated probabilities from deep learning models as input features. While SVM treats every tweet as a single unit, and CRF makes use of structural information of conversation and treats every branch as a unit, it is interesting that they have such a close performance in overall metrics.
Table 11 is the accuracy and macro-average F-score performance of SVM and CRF on each depth. For each depth, they still have a very close macro-average F-score except for depth 0 and depth 5. For source tweets, CRF has a 3 percent higher macro-average F-score than SVM, while in depth 5, SVM has a 7 percent higher macro-average F-score than CRF. For accuracy, the only obvious difference is that CRF has 3 percent higher accuracy than SVM in depth 4.
Since there is no substantial difference between SVM and CRF in overall and per class metrics, error analysis could help to show the difference in voting strategies. Similar to table 6, table

| Metrics / Model | Macro-average | | | S | D | Q | C |
|---|---|---|---|---|---|---|---|
| | Precision | Recall | F-score | | | | |
| SVM | 0.531 | 0.470 | 0.489 | 0.791 | 0.218 | 0.414 | 0.532 |
| CRF | 0.532 | 0.468 | 0.488 | 0.791 | 0.234 | 0.403 | 0.524 |

Table 10: Ensemble models' performance with SVM and CRF voting classifiers on test dataset

| Depth | tweets | S | D | Q | C | SVM | | CRF | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Accuracy | MacroF | Accuracy | MacroF |
| 0 | 140 | 132 | 4 | 0 | 4 | 0.9214 | **0.4712** | 0.9216 | **0.5026** |
| 1 | 1271 | 220 | 88 | 110 | 853 | 0.6588 | 0.4650 | 0.6536 | 0.4586 |
| 2 | 276 | 28 | 14 | 26 | 208 | 0.6918 | 0.3488 | 0.6952 | 0.3536 |
| 3 | 157 | 10 | 10 | 5 | 132 | 0.7828 | 0.3574 | 0.7944 | 0.3808 |
| 4 | 96 | 8 | 8 | 11 | 69 | **0.6798** | 0.3122 | **0.7086** | 0.3292 |
| 5 | 52 | 5 | 2 | 3 | 42 | 0.7856 | **0.5422** | 0.7910 | **0.4796** |
| 6+ | 205 | 16 | 26 | 16 | 147 | 0.7262 | 0.4598 | 0.7278 | 0.4594 |

Table 11: Accuracy and macro-average F-score of one-step SVM ensemble model and one-step CRF ensemble model by depth

12 illustrates the number of unique and common error cases regarding to SVM and CRF. This table has a distinctive distribution in the number of unique and common error cases, on which the numbers of common error cases are much more than the numbers of unique error cases. This distribution shows that SVM and CRF have quite a lot of similarities in misclassifying these probabilities. However, such a few unique error cases make looking for their own misclassifying patterns much easier.

| Class / Error type | comment | | support | | deny | |
|---|---|---|---|---|---|---|
| | SVM | CRF | SVM | CRF | SVM | CRF |
| **Unique** | 12 | 10 | 1 | 3 | 2 | 1 |
| **Common** | 227 | | 220 | | 129 | |

Table 12: SVM and CRF's unique and common error cases

Here are some representative unique error cases which are misclassified from small classes to *comment* class, these error cases intuitively reflect the sequential and non-sequential voting classifiers' features.

- @WSJ had a description. (SVM, deny to comment)
- @CTVNews you guys "confirmed" there were 3 shootings not long ago. How about you wait for official reports before saying things. (SVM, support to comment)
- @CBCNews yessss!! (CRF, support to comment)
- @kristinpuhl If you don't think there is clear motive in fighting or fleeing from police after robbery, you are a fucking moron. #Ferguson (CRF, deny to comment)

The first unique error case shows the disadvantage of SVM without structural information because the context is required to understand the attitude of the user. As CRF makes use of structural information but SVM does not, it is reasonable that SVM misclassifies this *deny* tweet to *comment*. As for the second error case, the author critiques a social media platform for a fake report without confirming the truth and gives trust to the official report. The true label of this tweet is supported, so we can predict that the previous tweet is about the official report. However, SVM misclassified it as a comment tweet, which means it has no idea about what it is talking about.
Another two are CRF's unique error cases, their stance can be predicted intuitively because they express their sentiment directly. Therefore, it can be known that CRF is not as good as SVM in classifying straightforward tweets. In other words, utilizing structural information may have less strength in classifying a single simple tweet comparing to SVM.

### 5.3.3 Comparing one-step and two-step ensembles

To improve the performance of ensemble models on our imbalanced dataset, we design two-step voting classifiers to classify comment tweets in the first step and small classes in the second step. In subsection 5.3.1, we already know that two-step voting classifiers don't have much improvement comparing to one-step voting classifiers in overall metrics. However, it is worthwhile to figure out how one-step and two-step differ in classifying the rumour stance, and how each classifier contributes to the voting process in two steps of the voting process. In this subsection, we compare and analyze the best one-step voting classifier and best two-step voting classifier.
Table 13 shows the performance of best single model, best one-step voting ensemble and best two-step voting ensemble models. Both one-step and two-step voting ensemble models have lower macro-average F-score than DistilBERT, but with a tiny difference. Therefore, there is no much difference among these three models in overall metrics. Besides that, they also have similar macro-average precision, recall, and per class F-score.

| Metrics / Model | Macro-average | | | S | D | Q | C |
|---|---|---|---|---|---|---|---|
| | Precision | Recall | F-score | | | | |
| DistilBERT | 0.530 | 0.497 | **0.505** | 0.788 | 0.229 | 0.469 | 0.533 |
| SVM | 0.531 | 0.470 | 0.489 | 0.791 | 0.218 | 0.414 | 0.532 |
| CRF+SVM | 0.531 | 0.467 | 0.486 | 0.782 | 0.225 | 0.416 | 0.521 |

Table 13: Best single, one-step and two-step models' performance

To better understand the effect of utilizing the sequential structure in ensemble models, and also the difference between one-step and two-step voting classification, we compare the one-step and two-step voting ensemble models' macro-average F1 score by depth. Figure 9 shows ensemble models' macro-average F-score from depth 0 to depth 6+. There is no obvious difference between depth 1 to depth 4, but some gaps exist in source depth and depth 5+. For the root part of the conversations, the two-step voting ensemble model performs better than both DistilBERT and SVM with a nearly 3 percent advantage. The non-sequential DistilBERT and one-step voting ensemble model with a non-sequential voting classifier perform worse in source tweets. Most source tweets are supporting a rumour, this result shows that a two-step voting ensemble model with CRF sequential voting classifier could learn this better. For tweets with a depth deeper than 5, one-step voting ensemble and two-step voting ensemble

also have almost the same macro-average F-score in deeper depth. They both have a better performance than DistilBERT in depth 5, but a worse performance than DistilBERT in depth 6+. Therefore, one-step SVM and two-step CRF+SVM don't have much difference except for CRF+SVM has a better performance in source tweets. This shows that the advantage of a two-step voting ensemble model with a sequential voting classifier is limited than the one-step voting ensemble model in identifying supporting tweets.

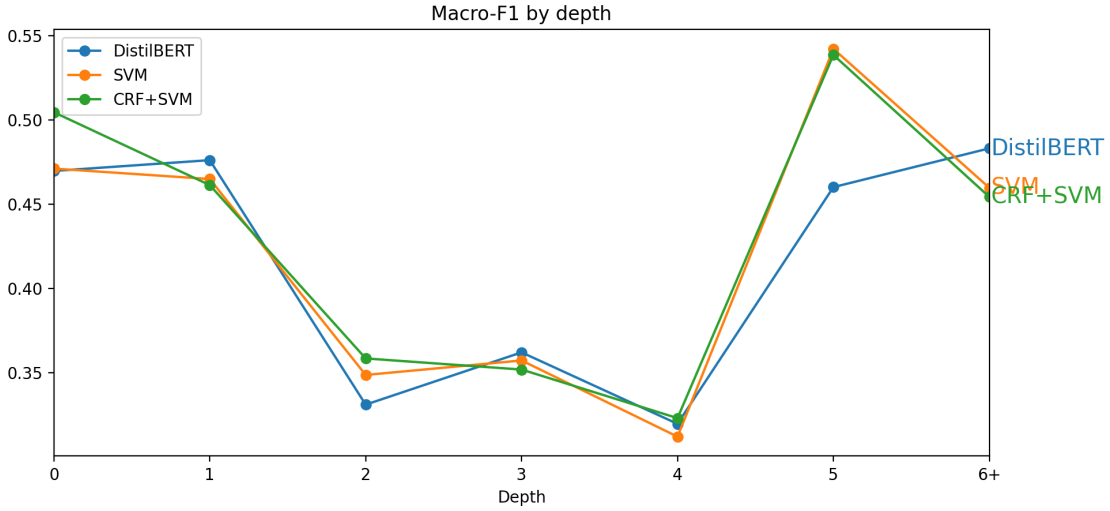



Figure 9: Macro-F1 scores by depth of tweet.

To figure out the same voting classifier's preference to each dimension of the input probability feature in one-step and two-step voting ensemble models, we analyze the SVM's feature weights in both one-step and two-step classification experiments. Interestingly, one-step and two-step voting ensemble models have such close performance. However, because of the different classification strategies, there could be much difference for the SVM's feature preference in two different voting processes. For one-step voting classification, SVM classifies four classes directly, while in two-step voting it first classifies big class and then classifies three small classes. Through this analysis, we hope to explore how BranchLSTM and DistilBERT's predicted probabilities are utilized by voting classifiers.

Table 14 and table 15 show the average feature weights of linear SVM in one-step voting and two-step voting classification respectively. As introduced before, the probability feature predicted by two deep learning models is an eight-length vector. Therefore, to classify tweets' stance based on these feature vectors, the SVM feature weights vector is also an eight-length vector, where each weight offers the importance of a dimension of the probability feature. Using the one-versus-one strategy in linear SVM, we have ($n$ * ($n$ - 1)  2 types of feature weights, where $n$ represents the number of classes. The one-versus-one strategy means every tweet is classified by every pair of two candidate classes for ($n$ * ($n$ - 1)  2 times, and then all the binary classification results are concluded to give the final prediction. As a result, each type of SVM feature weight is an eight-length feature weights vector for dividing two classes. In one-step voting classification, SVM has four classes. In two-step voting classification, SVM works in the second step which only needs to classify three small classes. As a result, linear SVM in one-step voting classification has (4 * 3)  2 = 6 types of feature weights, while in two-step has (3 * 2)  2 = 3 types of feature weights.

As the SVM's one-versus-one strategy classifies two classes in every binary classification, only four dimensions of the SVM's feature weights are correlated with input features in every binary classification. Taking a quick glance at both table 14 and table 15, it is easily to find out that the three biggest absolute values of correlated feature weights are all in DistilBERT part. Besides, most of these feature weights are negative, which means DistilBERT has a largely negative influence on corresponding classes in the certain binary classification process. Except that, the variance of SVM's correlated feature weights in the BranchLSTM part is much smaller than DistilBERT's part, which shows that BranchLSMT's feature weights contribute more equally to the classification process than DistilBERT.
In the two-step voting classification, there are only three types of SVM's feature weights because *comment* class is classified in the first step. It is worthwhile to compare the *support* versus *deny* feature weights between one-step SVM and two-step SVM because support and deny are very influential in the spreading of the information. The feature weights corresponding to *support* class don't change a lot in both BranchLSTM and DistilBERT part, but *deny* class's feature weights changed a lot in both BranchLSTM and DistilBERT from one-step to two-step. The BranchLSTM part's *deny* feature weight change from negative to positive, and DistilBERT part's *deny* feature weight decrease from -2 to -11. This means SVM's feature weights for BranchLSTM's *deny* dimension is positively correlated to input feature in one-step voting but negatively correlated in two-step voting, and SVM's feature weight for DistilBERT's *deny* dimension has a more negative influence in two-step voting than one-step voting.

| Stance / Feature type | BranchLSTM | | | | DistilBERT | | | |
|---|---|---|---|---|---|---|---|---|
| | support | comment | deny | query | support | comment | deny | query |
| (support, comment) | **1.857840** | **-0.616074** | -0.802722 | -0.439044 | -7.127957 | **-0.403323** | -3.040130 | 10.571412 |
| (support, deny) | **1.362475** | 1.096037 | **-3.771393** | 1.312881 | **4.551692** | -11.132948 | **-2.573309** | 9.154567 |
| (support, query) | **1.153870** | -0.507858 | 0.221886 | **-0.867897** | **3.040023** | 2.323683 | -10.943504 | **5.579799** |
| (comment, deny) | 0.306139 | **1.375730** | **-3.040178** | 1.358310 | 7.706208 | -11.456274 | **0.360187** | 3.389879 |
| (comment, query) | -0.021349 | **-0.397831** | 0.798213 | **-0.379032** | 4.510649 | **2.292224** | -9.664425 | **2.861553** |
| (deny, query) | 0.602523 | -1.255049 | **2.157468** | **-1.504941** | 1.584350 | 5.780189 | -10.496090 | **3.131550** |

Table 14: Average feature weights of SVM in one-step voting ensemble model

| Stance / Feature type | BranchLSTM | | | | DistilBERT | | | |
|---|---|---|---|---|---|---|---|---|
| | support | comment | deny | query | support | comment | deny | query |
| (support, deny) | **0.412595** | -1.388387 | **2.587989** | -1.612197 | **1.674947** | 6.008834 | -11.168103 | 3.484323 |
| (support, query) | **-1.540901** | -1.318144 | 4.714231 | **-1.855185** | **-4.771620** | 11.233030 | 3.432492 | -9.893904 |
| (deny, query) | -0.910160 | 0.617119 | **-0.351510** | **0.644551** | -3.152053 | -2.469326 | 11.690014 | **-6.068637** |

Table 15: Average feature weights of SVM on two-step voting ensemble model

# 6 Discussion

## 6.1 Results discussion

Firstly and most importantly, we adapt the original BranchLSTM classification method to make it able to train and predict on the different dataset with the various sizes of conversations, which could help later researchers to use BranchLSTM much easier.
In the comparison of sequential and non-sequential single models, we find that DistilBERT has more advantages in classifying the balanced dataset than BranchLSTM. DistilBERT has

better performance than BranchLSTM on small classes, but BranchLSMT performs better in predicting the big class. Through error analysis, it seems valid that DistilBERT has advantages in understanding the text content while BranchLSTM is better at utilizing context information.
In the comparison of all single models and ensemble models, we know that they perform roughly equally. Through depth analysis of single models and the best ensemble model, we find that the ensemble model takes advantages and disadvantages from both the single models. Through error analysis of the single models and the best ensemble model, we find that two single models have similar disadvantages with SVM respectively in misclassifying small classes, but SVM has more similarities in misclassifying big classes with DistilBERT than with BranchLSTM.
In the comparison of the sequential voting ensemble model and non-sequential voting ensemble model, we find that SVM can't make use of structural information to classify tweets that need to be understood based on context, but CRF is not as good as SVM in classifying straightforward tweets.
In the comparison of one-step and two-step voting classifiers in ensemble models, we find that the advantage of the two-step voting ensemble model with a sequential voting classifier is limited than the one-step voting ensemble model in identifying supporting tweets. Through analyzing the SVM's feature weights in both one-step and two-step voting classification, we find that SVM's feature weights for BranchLSTM's *deny* dimension is positive correlated to input feature in one-step voting but negatively correlated in two-step voting, and SVM's feature weight for DistilBERT's *deny* dimension has a more negative influence in two-step voting than one-step voting.

## 6.2 Shortcomings and analysis

There do exist some shortcomings in our experiments and models. First, the labels are imbalanced in the dataset but we didn't take any actions to alleviate it. The imbalance of dataset could limit the performance of the pre-trained model and voting classifiers. To avoid the influence caused by imbalanced dataset, we could try random oversampling to augment minority classes in the pre-processing step. Second, the method of incorporating BranchLSTM and DistilBERT probabilities seems oversimplified to some extent. We know that the ensemble models could learn both advantages and disadvantages from BranchLSTM and DistilBERT through error analysis, and this is the main reason that ensemble models don't improve a lot comparing to single models. We could change the way of combining two models like linear transformation, but not simply connecting two probability vectors. Except that, what we think may help to make the ensemble models more powerful is to attempt other ensemble methods which may combine BranchLSTM and DistilBERT more effectively. The voting classifier is just one of the ensemble methods, other ensemble methods with different ensemble strategies could be applied to single models in the future.

# 7 Conclusion

We expect that the ensemble models combining sequential and non-sequential deep learning models could make use of both their advantages and may improve the performance of single models. We also expect that a two-step voting ensemble model with both sequential and non-sequential voting classifier could utilize the structural information and text information from the probabilities may have better performance than a one-step voting classifier. Although the

improvement of the ensemble model is limited, we reveal many regular patterns through exhaustive analysis. We prove the advantages and disadvantages of sequential and non-sequential deep learning models through error analysis. We also show the different classifying strategies of sequential and non-sequential voting classifiers in ensemble models. The analysis between one-step voting and two-step voting reveals the voting classifier's preference in a different situation.

# References


Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E Schapire. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5): 1651–1686, 1998.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

Guoyong Cai, Hao Wu, and Rui Lv. Rumors detection in chinese via crowd responses. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 912–917. IEEE, 2014.

Leon Derczynski, Kalina Bontcheva, Maria Liakata, Rob Procter, Geraldine Wong Sak Hoi, and Arkaitz Zubiaga. Semeval-2017 task 8: Rumoureval: Determining rumour veracity and support for rumours. *arXiv preprint arXiv:1704.05972*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release., August 2015. URL `https://doi.org/10.5281/zenodo.27878`.

Xiaowen Dong, Dimitrios Mavroeidis, Francesco Calabrese, and Pascal Frossard. Multiscale event detection in social media. *Data Mining and Knowledge Discovery*, 29(5):1374–1405, 2015.

Emilio Ferrara. " manipulation and abuse on social media" by emilio ferrara with ching-man au yeung as coordinator. *ACM SIGWEB Newsletter*, (Spring):1–9, 2015.

Genevieve Gorrell, Kalina Bontcheva, Leon Derczynski, Elena Kochkina, Maria Liakata, and Arkaitz Zubiaga. Rumoureval 2019: Determining rumour veracity and support for rumours. *arXiv preprint arXiv:1809.06683*, 2018.

Sardar Hamidian and Mona Diab. Rumor identification and belief investigation on twitter. In *Proceedings of the 7th Workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 3–8, 2016.

Alfred Hermida, Fred Fletcher, Darryl Korell, and Donna Logan. Share, like, recommend: Decoding the social media news consumer. *Journalism studies*, 13(5-6):815–824, 2012.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

Zhiwei Jin, Juan Cao, Yongdong Zhang, and Jiebo Luo. News verification by exploiting conflicting social viewpoints in microblogs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Elena Kochkina, Maria Liakata, and Isabelle Augenstein. Turing at semeval-2017 task 8: Sequential approach to rumour stance classification with branch-lstm. *arXiv preprint arXiv:1704.07221*, 2017.

Sumeet Kumar and Kathleen M Carley. Tree lstms with convolution units to predict stance and rumor veracity in social media conversations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5047–5058, 2019.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

Gang Liang, Wenbo He, Chun Xu, Liangyin Chen, and Jinquan Zeng. Rumor identification in microblogging systems based on users' behavior. *IEEE Transactions on Computational Social Systems*, 2(3):99–108, 2015.

Xiaomo Liu, Armineh Nourbakhsh, Quanzhi Li, Rui Fang, and Sameena Shah. Real-time rumor debunking on twitter. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1867–1870, 2015.

Michal Lukasik, PK Srijith, Duy Vu, Kalina Bontcheva, Arkaitz Zubiaga, and Trevor Cohn. Hawkes processes for continuous time sequence classification: an application to rumour stance classification in twitter. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 393–398, 2016.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Saif Mohammad, Svetlana Kiritchenko, Parinaz Sobhani, Xiaodan Zhu, and Colin Cherry. Semeval-2016 task 6: Detecting stance in tweets. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 31–41, 2016.

Vahed Qazvinian, Emily Rosengren, Dragomir Radev, and Qiaozhu Mei. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1589–1599, 2011.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Victoria L Rubin and Tatiana Lukoianova. Truth and deception at the rhetorical structure level. *Journal of the Association for Information Science and Technology*, 66(5):905–917, 2015.

Victoria L Rubin, Niall Conroy, Yimin Chen, and Sarah Cornwell. Fake news or truth? using satirical cues to detect potentially misleading news. In *Proceedings of the second workshop on computational approaches to deception detection*, pages 7–17, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Steve Schifferes, Nic Newman, Neil Thurman, David Corney, Ayse Göker, and Carlos Martin. Identifying and verifying news through social media: Developing a user-centred tool for professional journalists. *Digital journalism*, 2(3):406–418, 2014.

Giovanni Stilo and Paola Velardi. Efficient temporal mining of micro-blog texts and its application to event discovery. *Data Mining and Knowledge Discovery*, 30(2):372–402, 2016.

Eugenio Tacchini, Gabriele Ballarin, Marco L Della Vedova, Stefano Moret, and Luca de Alfaro. Some like it hoax: Automated fake news detection in social networks. *arXiv preprint arXiv:1704.07506*, 2017.

Svitlana Volkova, Kyle Shaffer, Jin Yea Jang, and Nathan Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 647–653, 2017.

Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

Zhe Zhao, Paul Resnick, and Qiaozhu Mei. Enquiring minds: Early detection of rumors in social media from enquiry posts. In *Proceedings of the 24th international conference on world wide web*, pages 1395–1405, 2015.

Arkaitz Zubiaga, Heng Ji, and Kevin Knight. Curating and contextualizing twitter stories to assist with social newsgathering. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 213–224, 2013.

Arkaitz Zubiaga, Maria Liakata, Rob Procter, Kalina Bontcheva, and Peter Tolmie. Crowdsourcing the annotation of rumourous conversations in social media. 05 2015. doi: 10.1145/2740908.2743052.

Arkaitz Zubiaga, Elena Kochkina, Maria Liakata, Rob Procter, and Michal Lukasik. Stance classification in rumours as a sequential task exploiting the tree structure of social media conversations. *arXiv preprint arXiv:1609.09028*, 2016a.

Arkaitz Zubiaga, Maria Liakata, Rob Procter, Geraldine Wong Sak Hoi, and Peter Tolmie. Analysing how people orient to and spread rumours in social media by looking at conversational threads. *PloS one*, 11(3):e0150989, 2016b.

Arkaitz Zubiaga, Ahmet Aker, Kalina Bontcheva, Maria Liakata, and Rob Procter. Detection and resolution of rumours in social media: A survey. *ACM Computing Surveys (CSUR)*, 51 (2):1–36, 2018a.

Arkaitz Zubiaga, Elena Kochkina, Maria Liakata, Rob Procter, Michal Lukasik, Kalina Bontcheva, Trevor Cohn, and Isabelle Augenstein. Discourse-aware rumour stance classification in social media using sequential classifiers. *Information Processing & Management*, 54(2):273–290, 2018b.