Gryffindor

Real or Not? NLP with Disaster Tweets

ANIT GUPTA

NIKHIL GUPTA

SAMEERA TURUPU

SAURABH MAYDEO

SONAM DAWANI

SUSANTH DASARI

Motivation

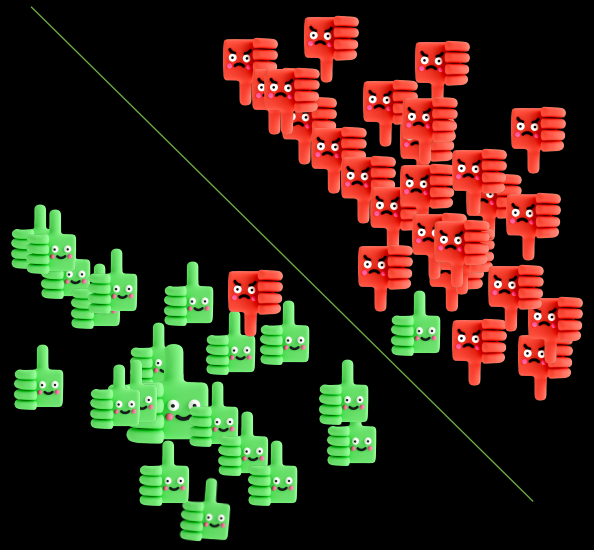
The use and impact of social media has skyrocketed over the past decade

A tweet could even serve as first news regarding an accident or a fire.

But tweets are confusing: “Kobe is on fire today!!!”

It is very hard for a machine to realize that this tweet is not about a fire but, about a person's game.

Our Aim



To build a simple but effective binary classification model to classify **disaster and general** tweets

The model would be an initial attempt to design a pipeline for contextual based **NLP**, which could be later used for topic modelling

Exploratory Data Analysis

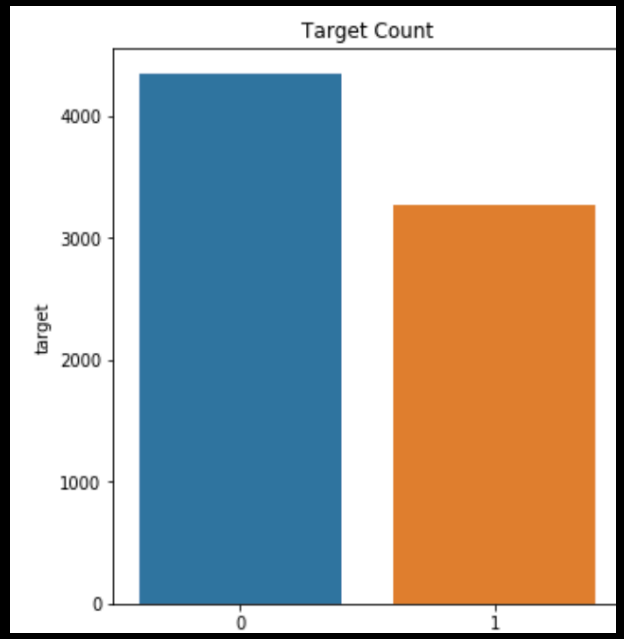
Raw Data

```
Training data shape (rows, cols):  (7613, 5)
****Train Data Info****
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
id          7613 non-null int64
keyword     7552 non-null object
location    5080 non-null object
text        7613 non-null object
target      7613 non-null int64
dtypes: int64(2), object(3)
```

```
Test data shape (rows, cols):  (3263, 4)
****Test Data Info****
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
id          3263 non-null int64
keyword     3237 non-null object
location    2158 non-null object
text        3263 non-null object
dtypes: int64(1), object(3)
```

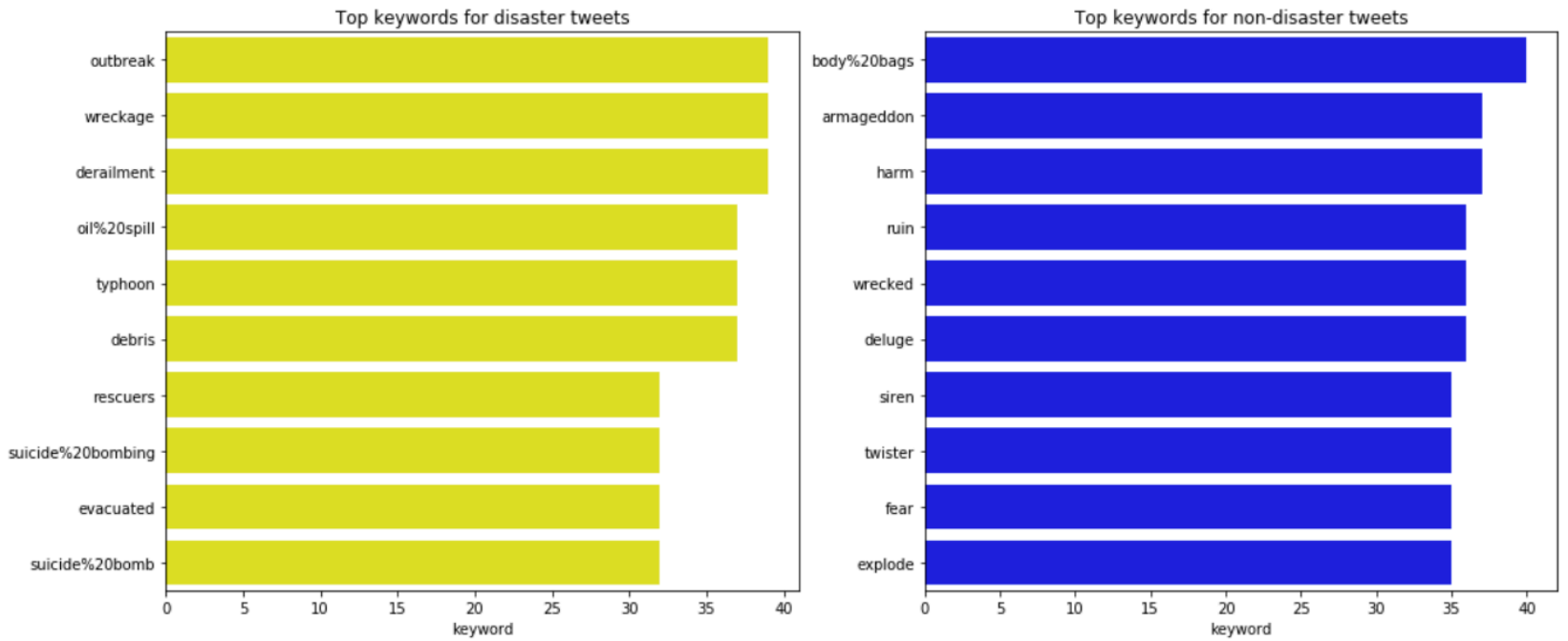
- Hash tags
- user mentions
- urls
- Misspelled words
- Abbreviations
- Average 88 chars per tweet

```
There are 2533 missing values in location.
Percentage of missing values in location 33.27203467752528 %
```

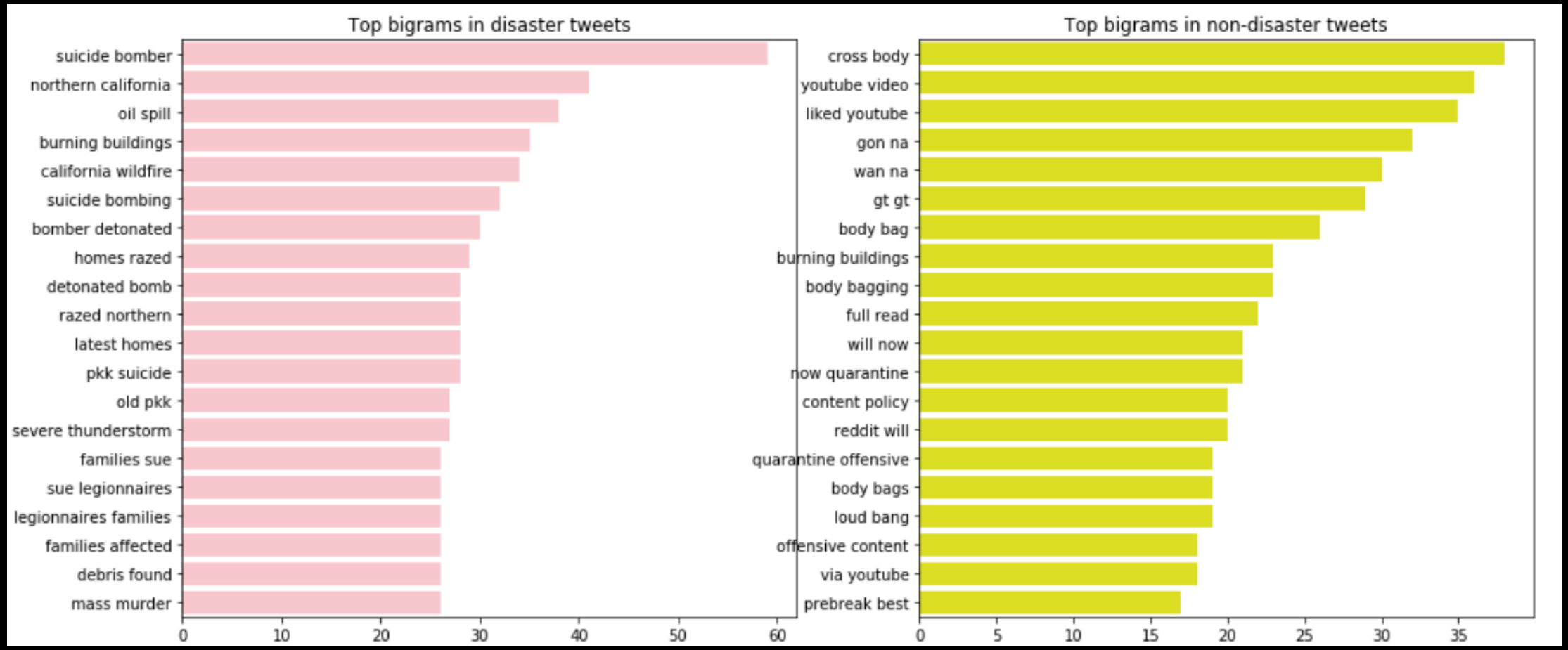


Exploratory Data Analysis

```
Top keywords for disaster tweets out of 218 keywords
Top keywords for non-disaster tweets out of 218 keywords
```



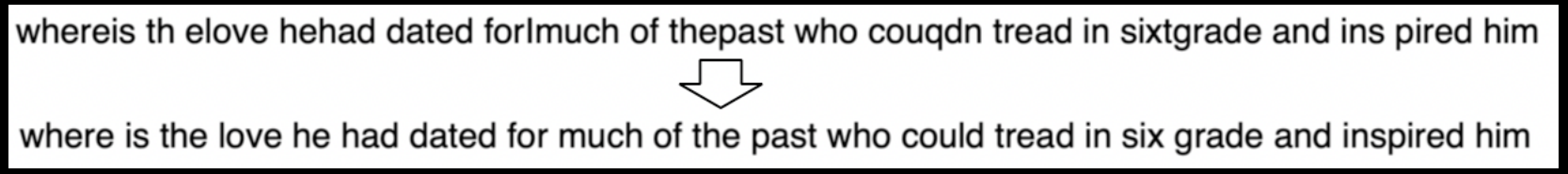
Exploratory Data Analysis



Pre-processing

- Usual Text preprocessing
  - Handle Unicode characters and special characters(such as emojis), Words containing numbers, Lemmatization and Removal of stop words
- Advanced Text preprocessing
  - Contextual spelling correction

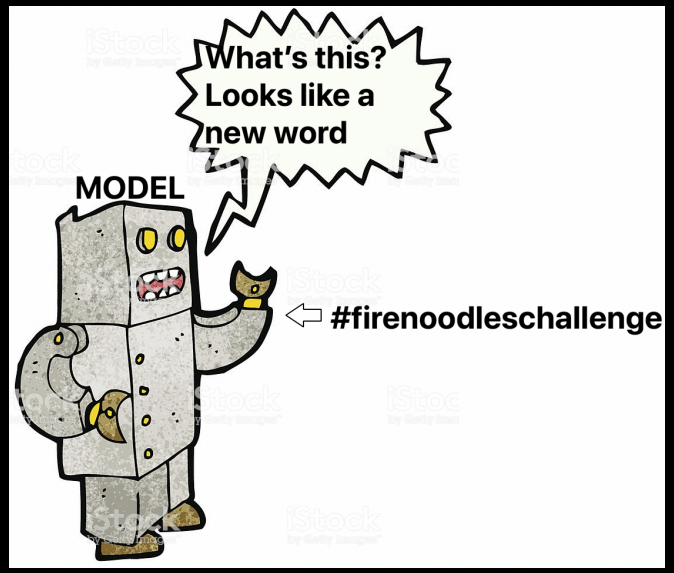
    Time taken for 10876 tweets: 371 seconds



  - Abbreviations (NYC -> New York City)
  - Repetitive characters (cooooooool -> cool)
  - Expand contractions (I’ve -> I have)

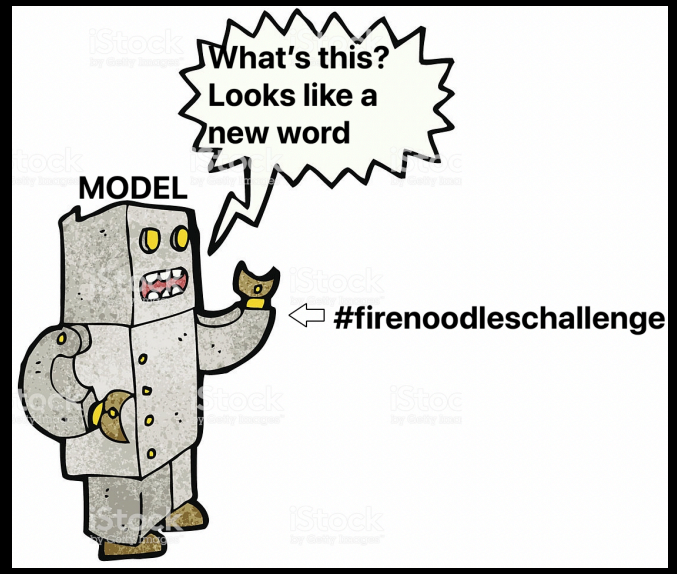
Pre-processing (Cont.)

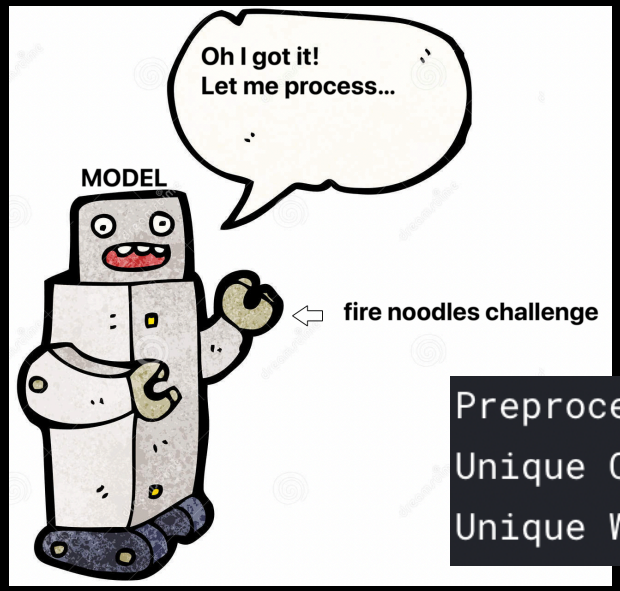
- Word segmentation of hashtags



Pre-processing (Cont.)

- Word segmentation of hashtags

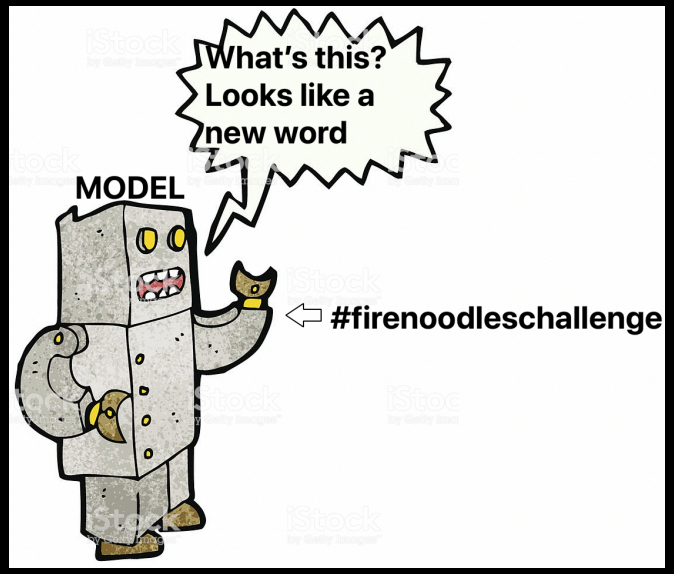


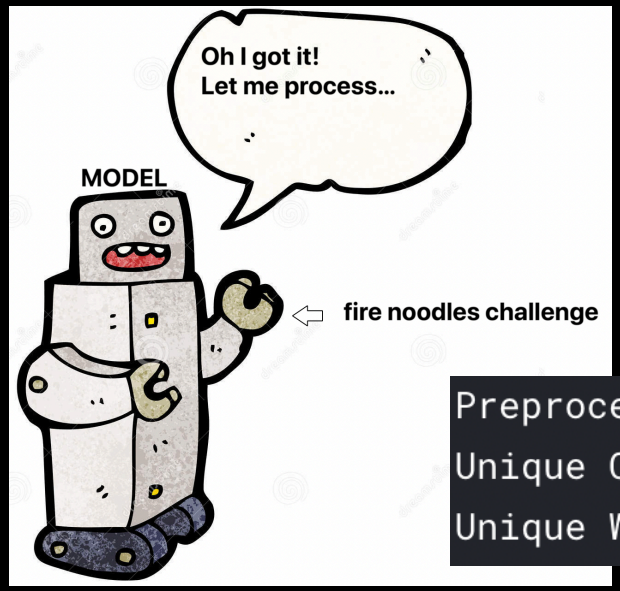



```
Preprocessing step:  hashtag_to_words
Unique Char Count ---> Before: 66 | After: 65
Unique Word Count ---> Before: 23337 | After: 22510
```

Pre-processing (Cont.)

- Word segmentation of hashtags






```
Preprocessing step:  hashtag_to_words
Unique Char Count ---> Before: 66 | After: 65
Unique Word Count ---> Before: 23337 | After: 22510
```

- URLs

```
@nxwestmidlands huge fire at Wholesale markets ablaze http://t.co/rwzbFVNXER
                                                      ⇩
@nxwestmidlands huge fire at Wholesale markets ablaze [URL]
```

Pre-processing (Cont.)

- Analyzed using BERT embedding
  - Before pre-processing

```
Found embeddings for 23.46% of vocab
Found embeddings for  66.31% of all text
```

  - After pre-processing

```
Found embeddings for 74.92% of vocab
Found embeddings for  93.86% of all text
```

Pre-processing (Cont.)

- Analyzed using BERT embedding
  - Before pre-processing

```
Found embeddings for 23.46% of vocab
Found embeddings for  66.31% of all text
```

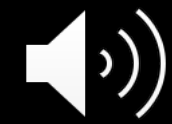
  - After pre-processing

```
Found embeddings for 74.92% of vocab
Found embeddings for  93.86% of all text
```

```
Preprocessing step:  remove_special_ucchar
Unique Char Count ---> Before: 67 | After: 66
Unique Word Count ---> Before: 23374 | After: 23337
Found embeddings for 46.89% of vocab
Found embeddings for  86.35% of all text

Preprocessing step:  hashtag_to_words
Unique Char Count ---> Before: 66 | After: 65
Unique Word Count ---> Before: 23337 | After: 22510
Found embeddings for 48.04% of vocab
Found embeddings for  86.48% of all text

Preprocessing step:  remove_mentions
Unique Char Count ---> Before: 65 | After: 64
Unique Word Count ---> Before: 22510 | After: 20185
Found embeddings for 54.69% of vocab
Found embeddings for  87.91% of all text
```



Pre-processing (Cont.)

- Analyzed using BERT embedding
  - Before pre-processing

```
Found embeddings for 23.46% of vocab
Found embeddings for  66.31% of all text
```

  - After pre-processing

```
Found embeddings for 74.92% of vocab
Found embeddings for  93.86% of all text
```

```
Preprocessing step:  remove_special_ucchar
Unique Char Count ---> Before: 67 | After: 66
Unique Word Count ---> Before: 23374 | After: 23337
Found embeddings for 46.89% of vocab
Found embeddings for  86.35% of all text

Preprocessing step:  hashtag_to_words
Unique Char Count ---> Before: 66 | After: 65
Unique Word Count ---> Before: 23337 | After: 22510
Found embeddings for 48.04% of vocab
Found embeddings for  86.48% of all text

Preprocessing step:  remove_mentions
Unique Char Count ---> Before: 65 | After: 64
Unique Word Count ---> Before: 22510 | After: 20185
Found embeddings for 54.69% of vocab
Found embeddings for  87.91% of all text
```

```
Preprocessing step:  Expand_Contractions
Unique Char Count ---> Before: 64 | After: 65
Unique Word Count ---> Before: 20185 | After: 20088
Found embeddings for 54.76% of vocab
Found embeddings for  88.85% of all text

Preprocessing step:  extra_spaces
Unique Char Count ---> Before: 65 | After: 65
Unique Word Count ---> Before: 20088 | After: 20087
Found embeddings for 54.76% of vocab
Found embeddings for  88.85% of all text
```

```
Preprocessing step:  removeRepeated
Unique Char Count ---> Before: 65 | After: 65
Unique Word Count ---> Before: 20087 | After: 20093
Found embeddings for 54.74% of vocab
Found embeddings for  88.83% of all text

Preprocessing step:  correct_misspelled_with_context
Unique Char Count ---> Before: 65 | After: 38
Unique Word Count ---> Before: 20093 | After: 11586
Found embeddings for 74.92% of vocab
Found embeddings for  93.86% of all text
```

SVM

Public Score

0.79959

- maximize the margin around the separating hyperplane.
- RBF kernel
- GridSearchCV for tuning hyper parameters and got best value of c=1.5, gamma=0.6
- Gamma is the kernel coefficient for ‘rbf’.

XGBOOST

- A powerful decision-tree based Ensemble model that employs Gradient Descent.
- feature size depends on the vocabulary of the data.
- hyper-paramaters like regularization, learning rate
- over-fitting -> optimize hyper-parameters like max-depth of tree

Public Score

0.80163

LSTM

Public Score

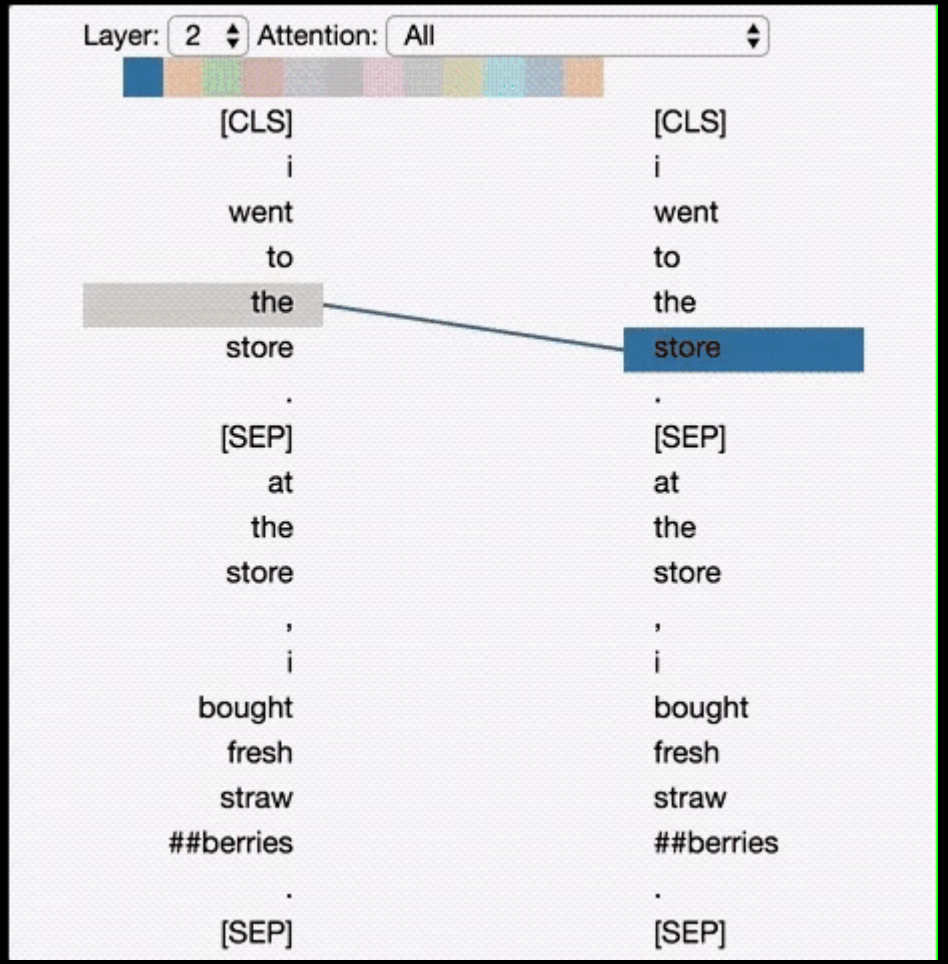
0.80265

- A special type of RNN, capable of learning long-term dependencies
- Bi-LSTMs focus on the past and the future directions of the input
- Embedding layer
- two layers of Bidirectional LSTMs with max pooling and average pooling
- Batch size, epochs, Activation functions are some of the hyperparameter used

BERT For NLP

BERT (Bidirectional Encoder Representations from Transformers) — a major breakthrough which took the Deep Learning community by storm because of its incredible performance.

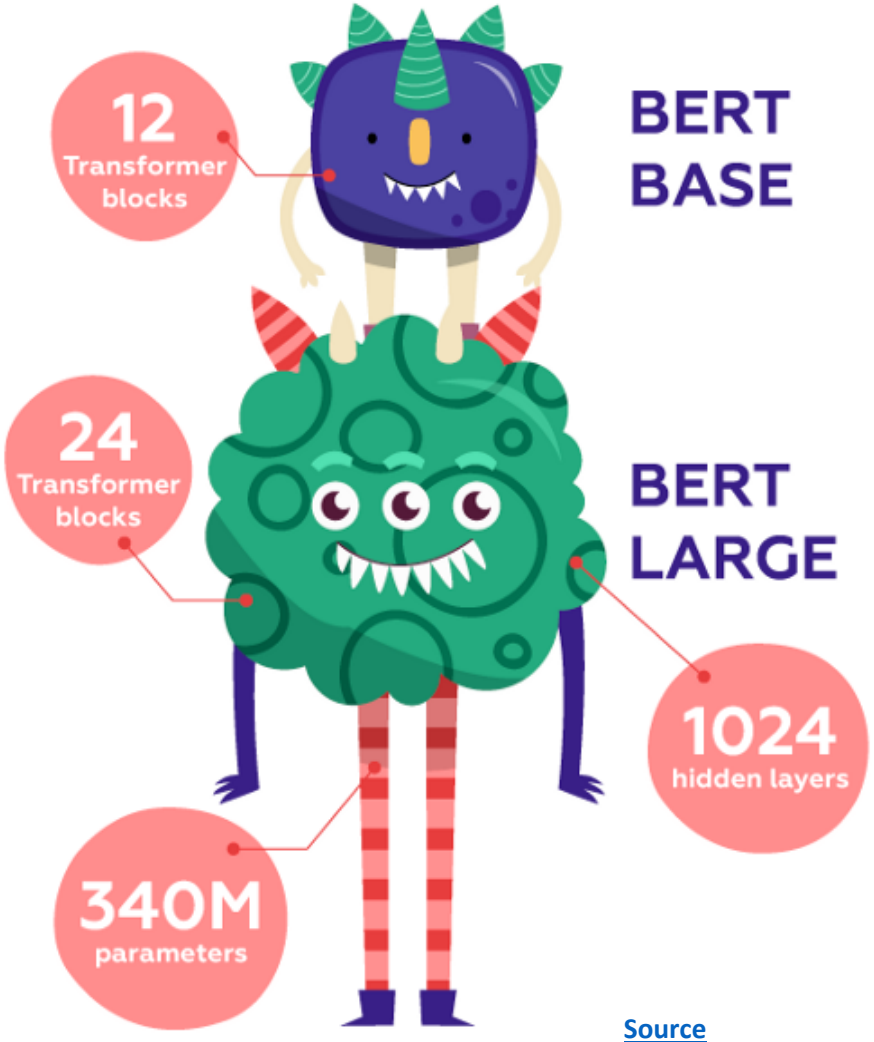
The image on the right visualizes attention as lines connecting the position being updated (left) with the position being attended to (right). Colors identify the corresponding attention head(s), while line thickness reflects the attention score.



Source

BERT Model at Glance

BERT comes in two sizes: BERT BASE, comparable to the OpenAI Transformer and BERT LARGE – the model which is responsible for all the striking results.



Source

BERT is pre-trained on 40 epochs over:

3.3B word corpus

- **800M words** — BooksCorpus
- **2,5B words** — English Wikipedia

16 TPUs for training runs on

INPUT

BERT takes a sequence of words which keep flowing up the stack. Each layer applies self-attention, and passes its results through a feed-forward network, and then hands it off to the next encoder.

OUTPUT

The output of each position is a vector of size called hidden_size (768 in BERT Base). This vector can be used as the input for a classifier you choose.

7,7% absolute improvement

80,5% The fine-tuned model for different datasets improves the GLUE benchmark to 80.5%

4,6% absolute improvement

86,7% MultiNLI accuracy to 86.7%

1,5% absolute improvement

93,2% the SQuAD v1.1 question answering Test F1 to 93.2

So on over a total of 11 language tasks

Advance Techniques used to train BERT

1. Masked LM (MLM)

   Randomly mask out 15% of the words in the input — replacing them with a [MASK] token

2. Next Sentence Prediction (NSP)

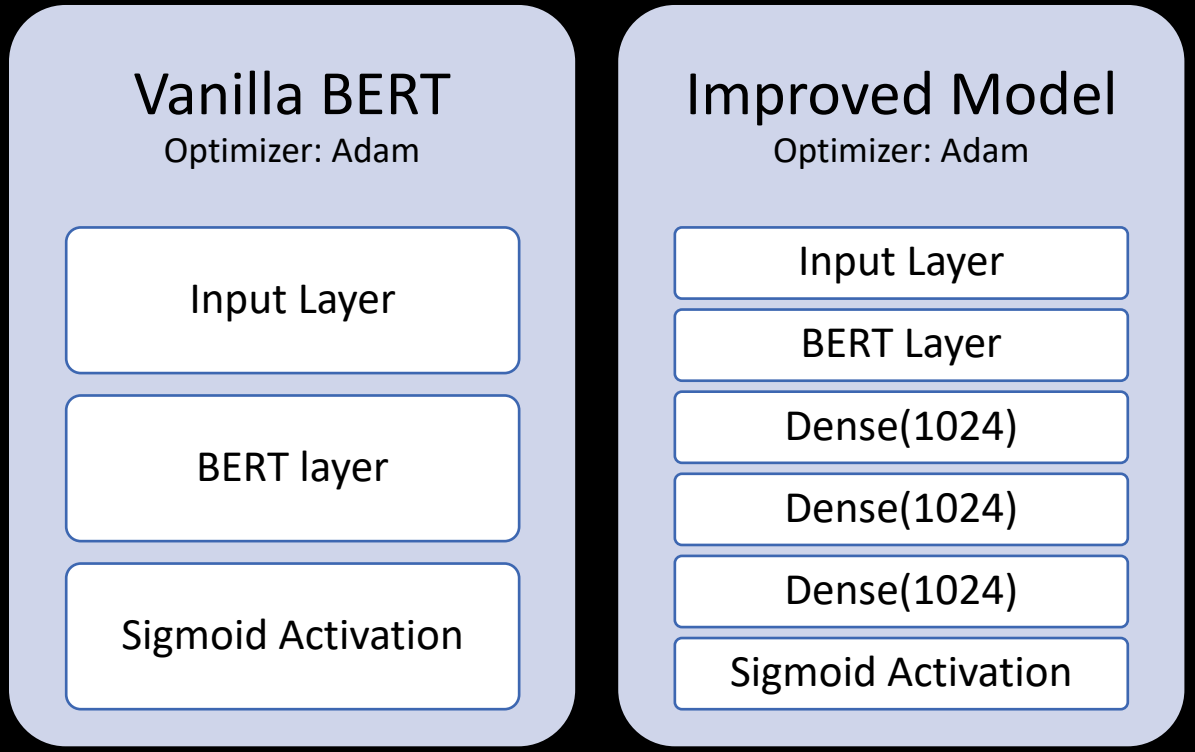
   In order to understand *relationship* between two sentences, BERT training process also uses next sentence prediction.

3. BERT Tokenization Technique

   BERT utilizes WordPiece for tokenization which in effect, splits token like “playing” to “play” and “##ing”. This is mainly to cover a wider spectrum of Out-Of-Vocabulary (OOV) words.

Model Training and Fine Tuning

- Pre-trained BERT:
  - BERT Large
  - 24 Layer, 1024 Hidden, 16 Attention heads
  - English only
  - 30,000 vocabulary
  - 340 Million parameters
- Challenges:
  - Complex model architecture
  - Limited to small batch size



Training strategy - 1:

Train model for 1-2 epochs with a LR decay of factor 10 → Train half epoch & decrease LR → Shuffle the data and repeat the process for 2-3 epochs

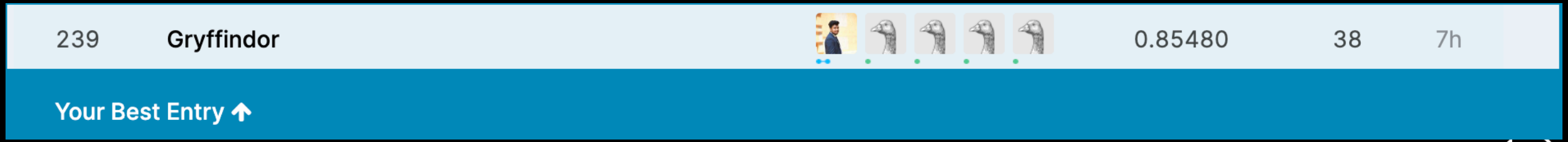
Model Training and Fine Tuning

Training Strategy - 2:

- Fine-tuning and external models:
  - Train BERT for 2-3 epochs
  - Generate embeddings from the trained model
  - Use Embeddings with SVM classifier and XGBoost classifier
- Hyper parameter tuning - Divided between the team with continuous feedback to each other.

Results

Model Specifications	Test Accuracy	Test F1 Score (100% data)	Kaggle public Score (30% data)
SVM	0.7974	0.7271	0.7995
XGBoost	0.7787	0.7303	0.8016
LSTM	0.8139	0.7666	0.8026
Vanilla BERT	0.8461	0.8086	0.8548
BERT + Training Strategy 1	0.8464	0.8082	0.8527
BERT + Training Strategy 2	0.8467	0.8090	0.8517
BERT + Training Strategy 2 + SVM	0.8427	0.8079	0.8456
BERT + Training Strategy 2 + XGBoost	0.8369	0.7995	0.8435



Version History

fileName	date	description	status	publicScore	privateScore
Text-classification with BERT+XGBOOST	2020-04-27 22:21:49	Final BERT Encodings + XGBoost	complete	0.84355	None
Text-classification with BERT+XGBOOST	2020-04-27 22:20:23	Final BERT Encodings + SVM	complete	0.8456	None
Text-classification with BERT+XGBOOST	2020-04-27 22:19:02	Final BERT Encodings + Strategy 2	complete	0.85173	None
Text-classification with BERT+XGBOOST	2020-04-27 21:50:39	BERT Encodings + SVM	complete	0.84253	None
Text-classification with BERT+XGBOOST	2020-04-27 14:49:25	From "Text-classification with BERT+XGBOOST" Script	complete	0.8231	None
Text-classification with BERT+XGBOOST	2020-04-26 08:34:06	From "Text-classification with BERT+XGBOOST" Script	complete	0.84151	None
Text-classification with BERT+XGBOOST	2020-04-24 17:30:27	From "Text-classification with BERT+XGBOOST" Script	complete	0.85173	None
submission_bertnn (1).csv	2020-04-13 04:00:01	None	complete	0.8456	None
submission_bertnn.csv	2020-04-12 23:06:03		complete	0.8456	None
submission.csv	2020-04-08 19:42:57	None	complete	0.8364	None
submission (6).csv	2020-04-07 14:09:34	None	complete	0.79447	None
XGBoost for Text Classification	2020-04-07 13:18:22	From "XGBoost for Text Classification" Script	complete	0.80163	None
submission.csv	2020-04-07 07:09:59	None	complete	0.80163	None
submission.csv	2020-04-07 07:03:48	None	complete	0.79345	None
XGBoost for Text Classification	2020-04-07 06:29:05	From "XGBoost for Text Classification" Script	complete	0.78425	None
submission.csv	2020-04-06 19:35:13	Support vector machines	complete	0.80265	None
submission.csv	2020-04-06 17:24:21	LSTM	complete	0.80265	None
sample_submission.csv	2020-04-06 10:32:03	None	complete	0.57055	None
Tweet Classification_ver1.0.0	2020-04-06 05:39:00	From "Tweet Classification_ver1.0.0" Script	complete	0.83946	None
XGBoost for Text Classification	2020-04-05 18:04:04	From "XGBoost for Text Classification" Script	complete	0.79345	None

Text-classification with BERT+XGBOOST

Python notebook using data from multiple data sources · 88 views · 1d ago · gpu · Edit tags

Version 17	a day ago	Text-classi...	26370.4s	0 B	+17
Version 16	a day ago	Text-classi...	1633.4s	0 B	+45

eet Classification_ver2

on notebook using data from Real or Not? NLP with Disaster Tweets · 157 views · 23d ago · gpu, tpu

25	23 days ago	Tweet Clas...	2212.2s	0 B	+31	-18
24	24 days ago	Tweet Clas...	2242.2s	0 B	+1	-1
23						-38
22						-89
21						
20						
19						

Tweet Classification_ver2.3.3

Python notebook using data from Real or Not? NLP with Disaster Tweets · 230 views · 1mo ago · gpu · Edit tags

ons

Version 21	a month ago	Tweet Clas...	2194.7s	0 B	+6	-4
Version 20	a month ago	Tweet Clas...	1796.5s	0 B	+69	-78
Version 19	a month ago	Tweet Clas...	2030.6s	0 B	+22	-22
Version 18	a month ago	Tweet Clas...	2052.7s	0 B	+4	-3
Version 17	a month ago	Tweet Clas...	2033.2s	0 B	+1	-2
Version 16	a month ago	Tweet Clas...	2090.2s	0 B	+29	-3

Extensive pre-processing for BERT

Python notebook using data from Real or Not? NLP with Disaster Tweets · 40 views · 1d ago · Edit tags

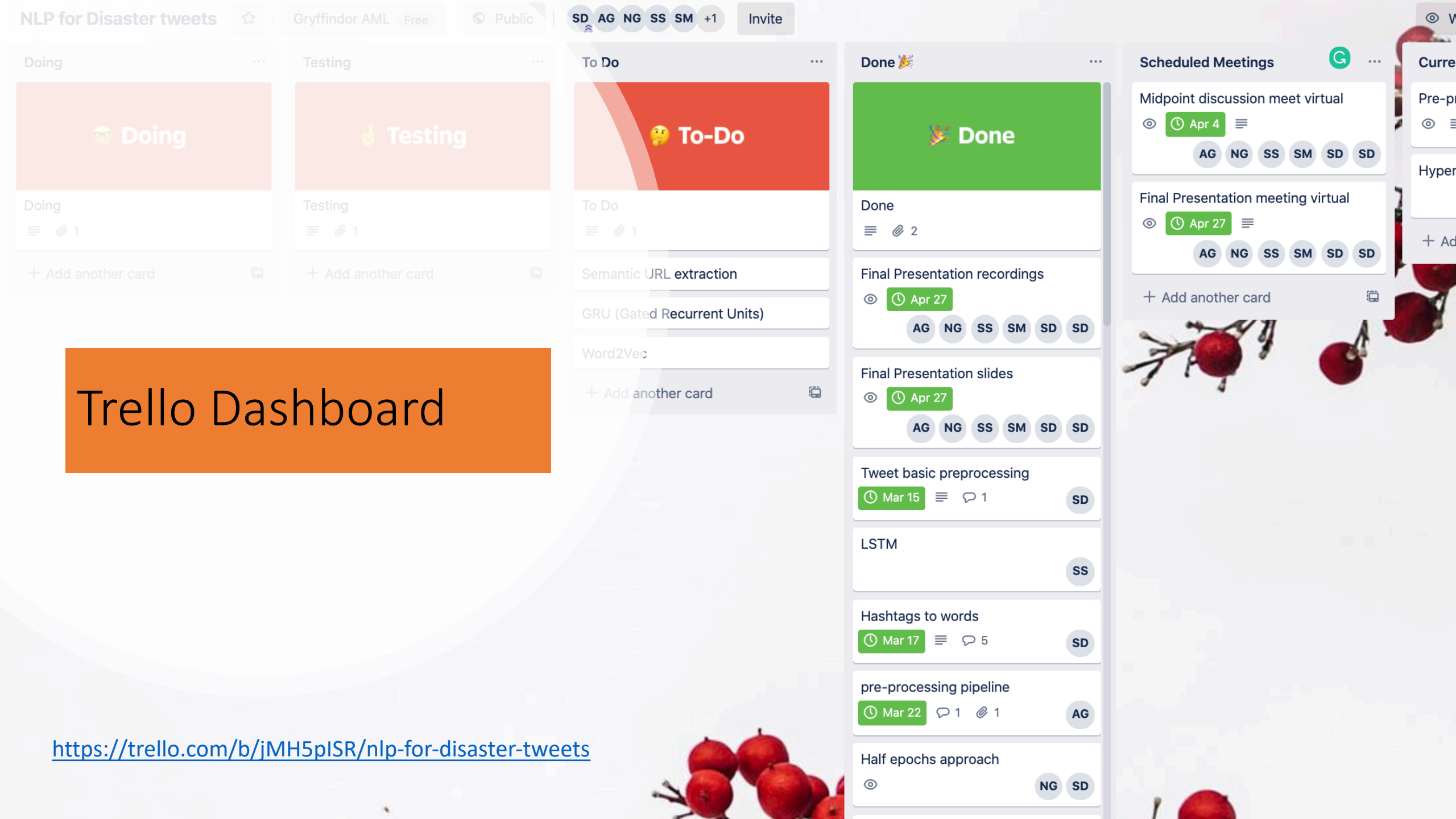
Version 7	a day ago	Extensive ...	767.2s	0 B	0	0
Version 6	a day ago	Extensive ...	3.3s	0 B	+8	-14
Version 5	2 days ago	Extensive ...	791.9s	0 B	+1	-1

Tweet Classification_ver1.0.0

Python notebook using data from Real or Not? NLP with Disaster Tweets · 9 views · 22d ago · gpu · Edit tags

Versions

Version 1	22 days ago	Tweet Clas...	1708.8s	0 B	0
Forked from	a month ago	Tweet Clas...	1722.7s	3.75 GB	0



Trello Dashboard

https://trello.com/b/jMH5pISR/nlp-for-disaster-tweets

Thank you

Questions?