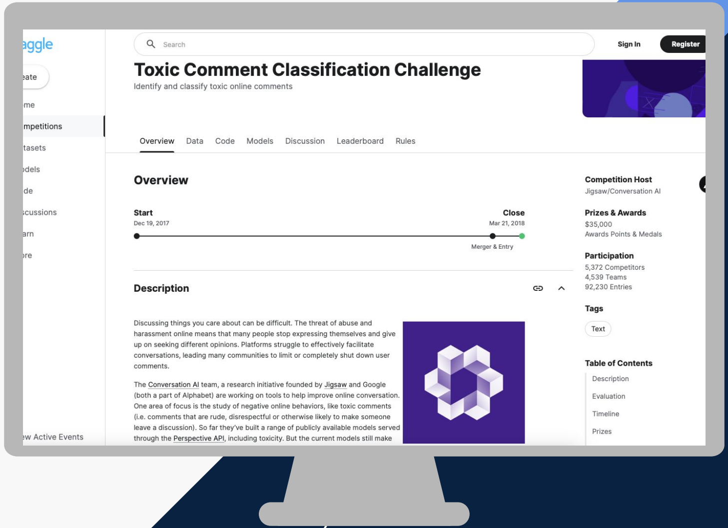
Toxic Comment Classification Project

Riccardo Frigerio 852226
Mattia Napoli 852239
Giulia Tremolada 861144

The Challenge

- Labeling Wikipedia comments based on various categories of toxicity
- Focus on the embedding layer to maximize performance
- Experiments using straightforward architectures



01

Dataset

Data Exploration

id	comment_text	toxic	severe toxic	obscene	threat	insult	Identity hate
0001d958c54c6e35	You, sir, are my hero. Any chance you remember what page that's on?	0	0	0	0	0	0
0002bcb3da6cb337	C**KSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0

Dataset Row Examples

toxic	12238	9.5%
severe toxic	1274	1%
obscene	6734	5.2%
threat	404	0.3%
insult	6263	4.9%
identity hate	1111	0.8%

Label Distribution in the Trainset

Preprocessing

- Removal of punctuation, numbers, non-alphanumeric characters
- Lemmatization
- Stopwords removal

- Tokenization
- Vectorization

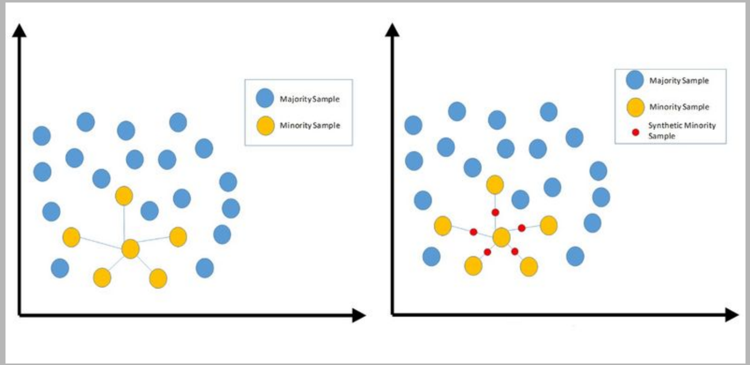
```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

def preprocess(text):
  text = text.lower() # convert text to lowercase
  text = re.sub(r'\d+', '', text) # remove numbers
  text = str(text).replace("\n", " ") # remove newline characters
  text = re.sub(r'[^\w\s]', '', text) #
  text = text.strip() # remove whitespaces

  # remove stop words
  stop_words = set(stopwords.words('english'))
  word_tokens = word_tokenize(text)
  filtered_tokens = [token.lower() for token in word_tokens if token.lower() not in stop_words]

  # Lemmatize the tokens
  lemmatizer=WordNetLemmatizer()
  lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]

  # Join the tokens back into a string
  preprocessed_text = " ".join(lemmatized_tokens)

  return preprocessed_text
```

“*D’aww! He matches this background colour I’m seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)*”

“*daww match background colour im seemingly stuck thanks talk january utc*”

Data Augmentation

- Very high imbalance between labels in the dataset
- Multilabel SMOTE for synthetic sample generation
- Unsatisfying results from this approach



02

Methodological Approach

Embeddings

Keras + positional encoding

Weights learnt from scratch

GloVe

Wikipedia based, 300D

FastText

Wikipedia based, 300D, handles OOV words

Frozen Weights + Fine-tuning

Models - Transformer

- Initial Model for Text Processing
- Multihead Attention Layer
- Maximize Embeddings' representation

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200)]             0

 embedding (Embedding)       (None, 200, 300)          55737900

 transformer_block_1 (Trans  (None, 200, 300)          1264800
 formerBlock)

 global_average_pooling1d_1  (None, 300)               0
  (GlobalAveragePooling1D)

 dropout_6 (Dropout)         (None, 300)               0

 dense_6 (Dense)             (None, 100)               30100

 dropout_7 (Dropout)         (None, 100)               0

 dense_7 (Dense)             (None, 6)                 606

=================================================================
Total params: 57033406 (217.57 MB)
Trainable params: 1295506 (4.94 MB)
Non-trainable params: 55737900 (212.62 MB)
_________________________________________________________________
```

Models - BiLSTM

- Processing of sequential data
- Multiple RNNs examined
- Bidirectional Layer to process Input in both DIrections

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200)]             0

 embedding (Embedding)       (None, 200, 300)          55011900

 bidirectional_1 (Bidirecti  (None, 256)               439296
 onal)

 dropout_2 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 6)                 774

=================================================================
Total params: 55484866 (211.66 MB)
Trainable params: 472966 (1.80 MB)
Non-trainable params: 55011900 (209.85 MB)
_________________________________________________________________
```

Models - CNN

- Introduced to solve difficulties with “Threat” and “Identity Hate”
- Weight sharing and translation invariance
- Most efficient implementation

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200)]             0

 embedding (Embedding)       (None, 200, 300)          55011900

 conv1d_3 (Conv1D)           (None, 196, 500)          750500

 global_max_pooling1d_3 (Gl  (None, 500)               0
 obalMaxPooling1D)

 dense_2 (Dense)             (None, 500)               250500

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 6)                 3006

=================================================================
Total params: 56015906 (213.68 MB)
Trainable params: 1004006 (3.83 MB)
Non-trainable params: 55011900 (209.85 MB)
_________________________________________________________________
```

03

Results & Discussion

AUC and Keras Embeddings Evaluation

Table 1: AUC Metric (in %) of Models

(a) Positional

Model	
Transformer	95.5
Bi-LSTM	96.1
CNN	96.1

(b) GloVe

Model	Frozen	Fine Tuning
Transformer	95.9	96.5
Bi-LSTM	96.6	97.0
CNN	95.9	96.2

(c) FastText

Model	Frozen	Fine Tuning
Transformer	95.3	96.8
Bi-LSTM	96.2	97.1
CNN	96.8	97.3

Kaggle Winner AUC: 98.8%

Table 2: F1-Measure (in %) of Models with Positional Embeddings

	Toxic	*Severe Toxic*	*Obscene*	*Threat*	*Insult*	*Identity Hate*
Transformer	61	5	67	0	58	0
Bi-LSTM	63	36	70	0	61	0
CNN	65	24	68	0	62	0

GloVe Vs FastText Embeddings

Table 3: Evaluation Metrics (in %) of Models with GloVe Embeddings

Label (count)	*Model*	*Frozen*			*Fine Tuning*		
		Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Toxic (6090)	Transformer	54	80	64	55	83	67
	Bi-LSTM	57	80	66	55	84	67
	CNN	54	82	65	53	85	65
Severe Toxic (367)	Transformer	21	5	8	32	33	32
	Bi-LSTM	36	38	37	37	39	38
	CNN	35	13	19	32	48	38
Obscene (3691)	Transformer	65	65	65	63	73	68
	Bi-LSTM	69	64	66	62	74	68
	CNN	66	66	66	62	73	67
Threat (211)	Transformer	0	0	0	0	0	0
	Bi-LSTM	0	0	0	0	0	0
	CNN	45	14	22	38	39	39
Insult (3427)	Transformer	57	67	61	57	72	64
	Bi-LSTM	63	63	63	59	72	65
	CNN	60	63	62	56	72	63
Identity Hate (712)	Transformer	0	0	0	73	29	41
	Bi-LSTM	69	36	47	66	47	55
	CNN	73	31	44	61	53	57

Table 4: Evaluation Metrics (in %) of Models with FastText Embeddings

Label (count)	*Model*	*Frozen*			*Fine Tuning*		
		Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Toxic (6090)	Transformer	51	87	64	50	90	64
	Bi-LSTM	57	79	66	55	86	67
	CNN	60	78	68	54	86	66
Severe Toxic (367)	Transformer	32	33	33	31	54	39
	Bi-LSTM	35	18	24	37	41	39
	CNN	42	27	33	40	42	41
Obscene (3691)	Transformer	74	54	63	58	81	68
	Bi-LSTM	72	64	67	62	79	69
	CNN	71	62	66	60	78	68
Threat (211)	Transformer	0	0	0	0	0	0
	Bi-LSTM	0	0	0	0	0	0
	CNN	32	11	17	34	25	29
Insult (3427)	Transformer	60	48	54	50	74	60
	Bi-LSTM	61	59	60	56	73	63
	CNN	69	57	62	59	72	65
Identity Hate (712)	Transformer	0	0	0	46	13	21
	Bi-LSTM	63	5	10	59	22	32
	CNN	80	16	27	69	45	54

Discussion

Embeddings	Performance of different embeddings reflected our expectations
Fine-tuned GloVe vs Fine-tuned FastText	Similar F1-score, Precision and Recall to but FastText has better AUC. Glove Embeddings appear to be of higher starting quality for the task.
Models	Even performance without clear trends. CNN achieved best results overall

Approach Evaluation

AUC: 97.3% ----▶ AUC: 98.8%

Data Augmentation

Data Cleaning

Sophisticated Models

04

Conclusions

Embeddings

- Fine-tuned pretrained embeddings outperformed their frozen version and Keras version
- Fine-tuned Glove and FastText had similar performances

Models

- CNN was overall the best model because it was only one that was able to recognize all labels

Thank you!

Riccardo Frigerio 852226
Mattia Napoli 852239
Giulia Tremolada 861144