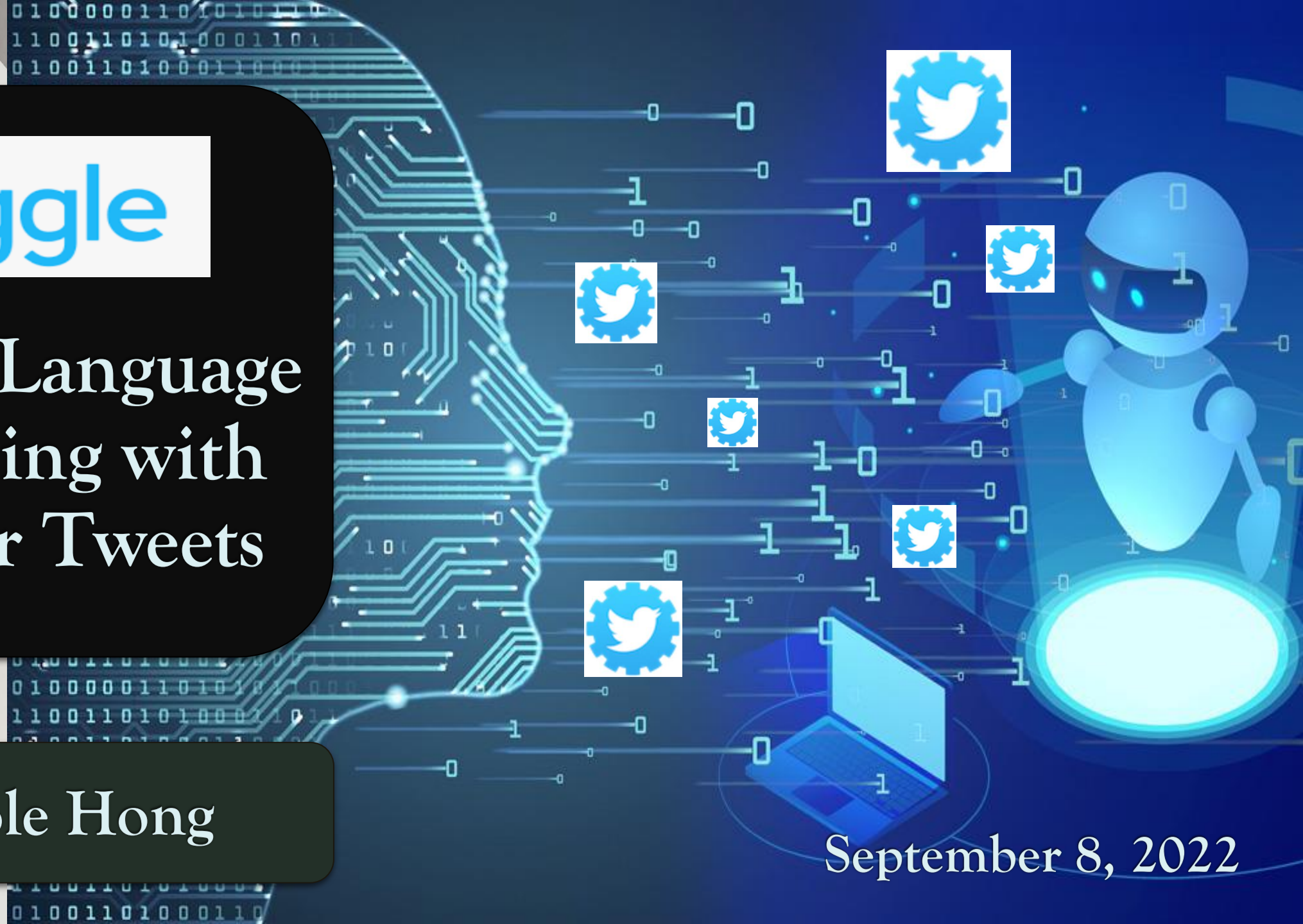


kaggle

Natural Language Processing with Disaster Tweets

Nicole Hong

September 8, 2022

Why this Kaggle?

- On-going, Knowledge-based Competition
- Same evaluation and ranking system as other competitive Kaggle
- Brush up on NLP, while competing
- NLP on Twitter data

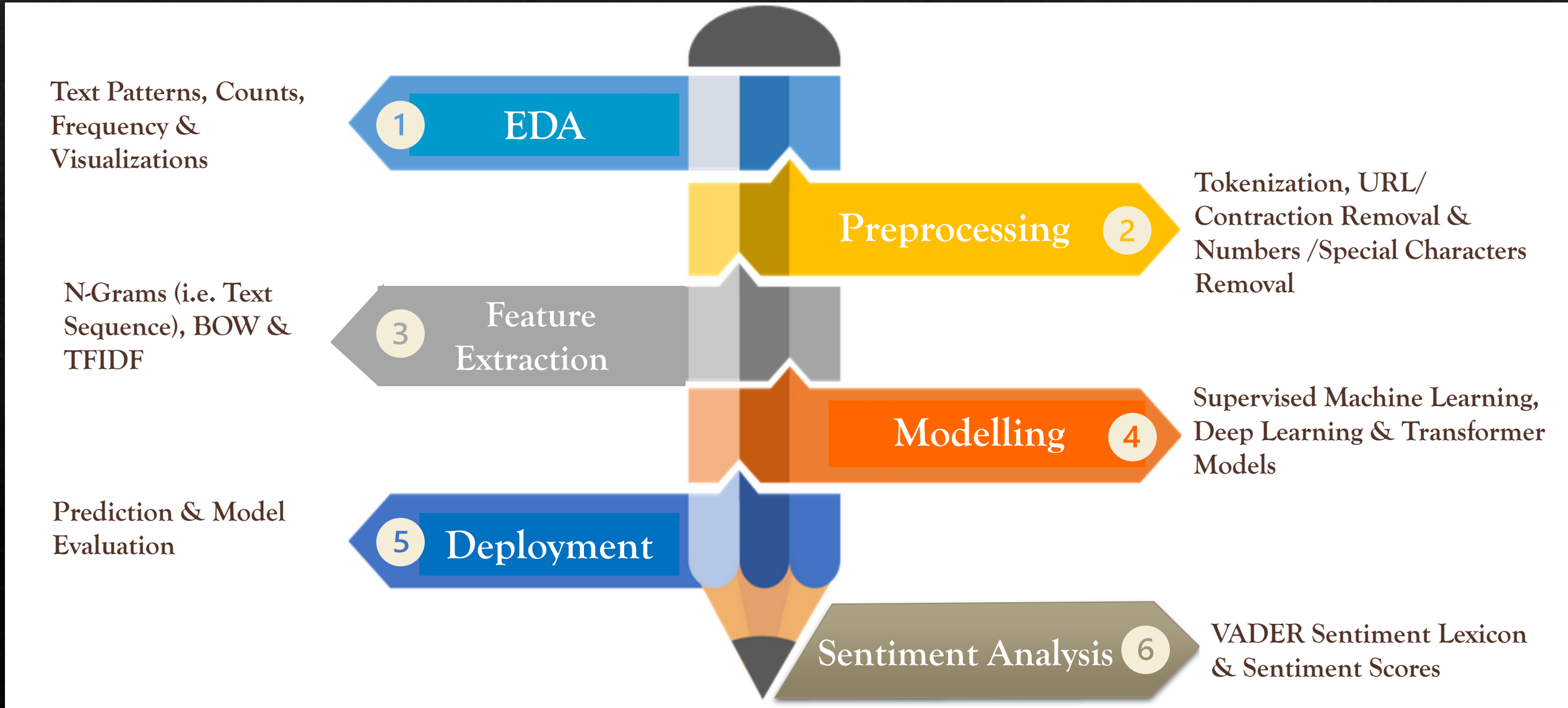
Objective

- Identify disaster related tweets with high accuracy (Classification)
- Determine level of crisis for better response to disasters (Sentiment Analysis)

Why Twitter...?

- Real-time data on experience of those directly affected by a disaster
- Useful sentiment-based data
- Geospatial component, location-based social network
- Developing policy both during and after disasters

Kaggle Workflow



Kaggle Data

train.csv (987.71 kB)

Total 7,613 samples

Detail Compact Column 5 of 5 columns

id	keyword	location	text	target
1 – 10.9k	222 unique values	[null] 33% USA 1% Other (4976) 65%	7503 unique values	0

Tweets in August 2015

test.csv (420.78 kB)

Total 3,263 samples

Detail Compact Column 4 of 4 columns

id	keyword	location	text
0 – 10.9k	222 unique values	[null] 34% New York 1% Other (2120) 65%	3243 unique values

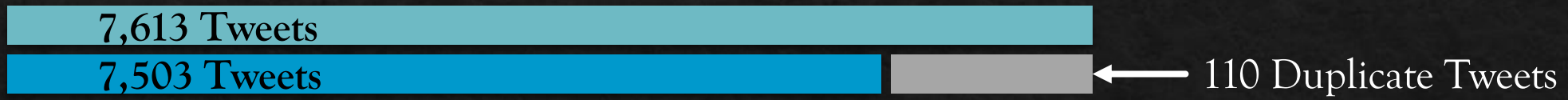
No Target Class

Data Preparation

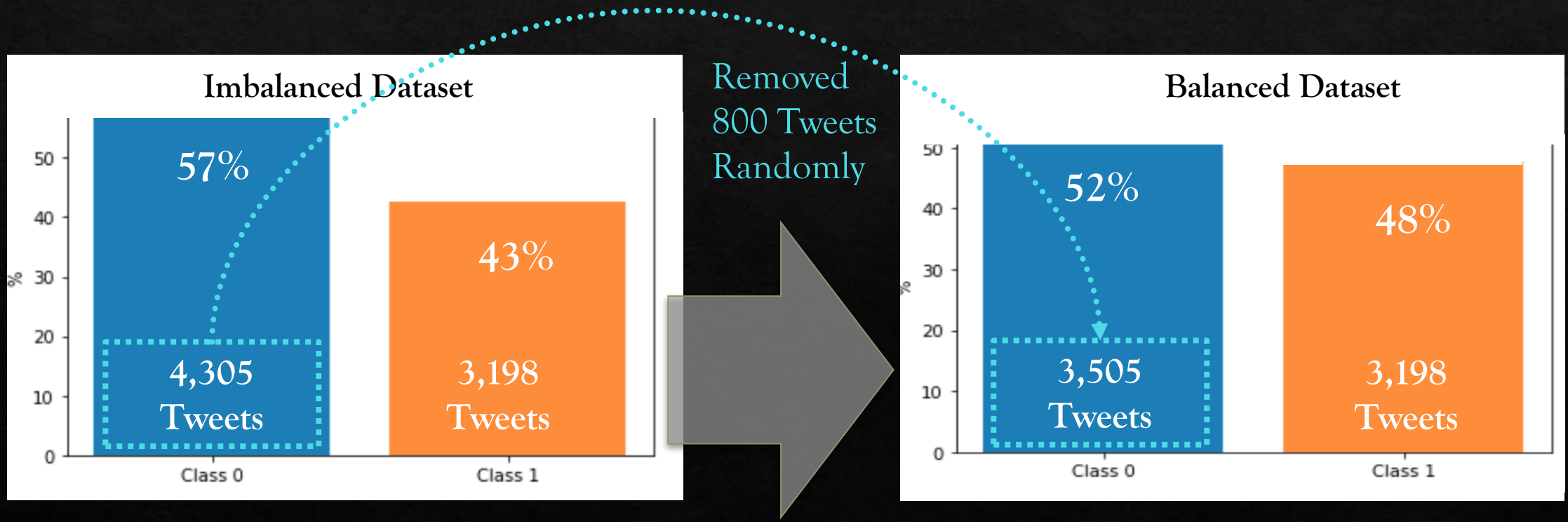
Class 0 Non-disaster related Tweets

Class 1 Disaster related Tweets

(1) Remove Duplicate Tweets



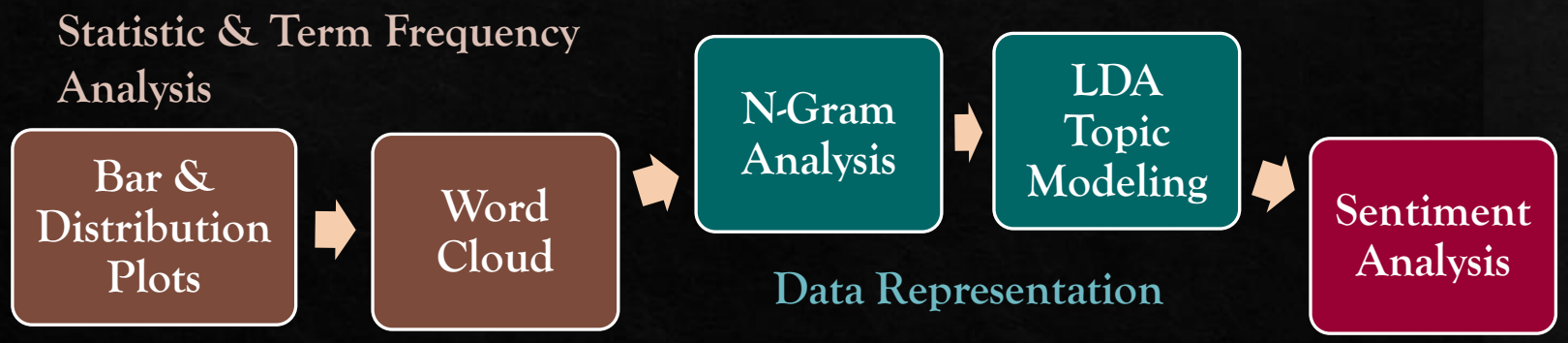
(2) Balance Target Class – Random Re-Sampling Train Data



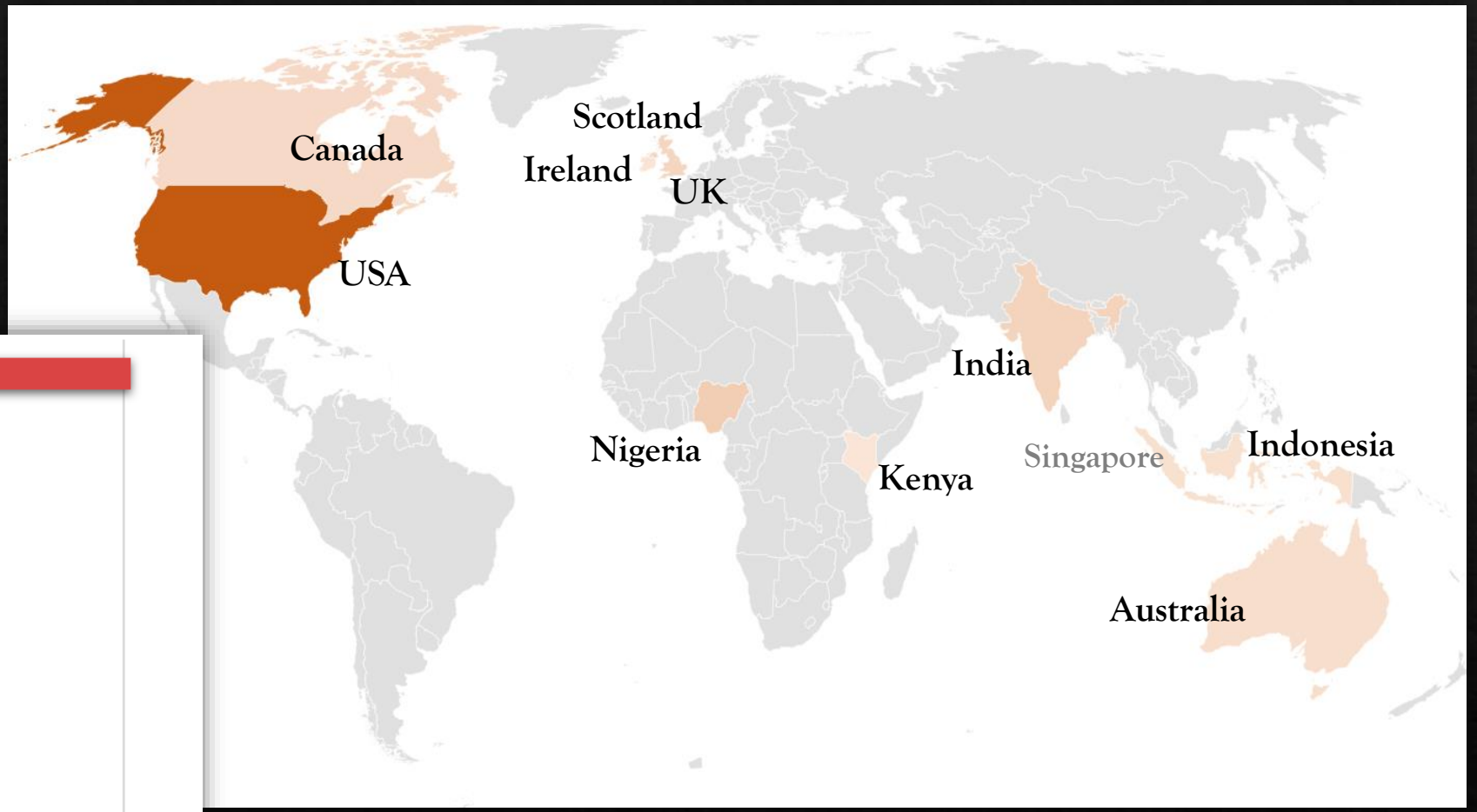
Exploratory Data Analysis (EDA)

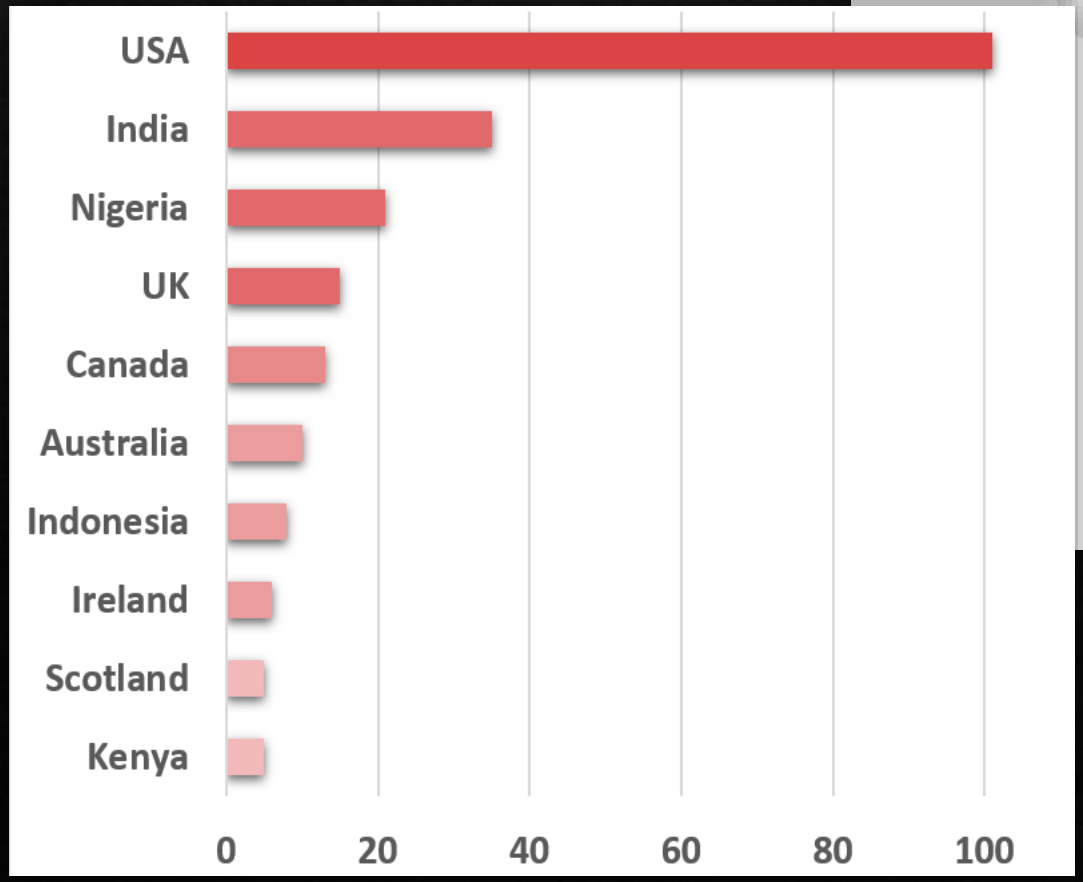
EDA for NLP:

- Descriptive statistics & visualization
- Missing values and value patterns correlated with the target
- Highly correlated, duplicate features or most predictive features for the problem

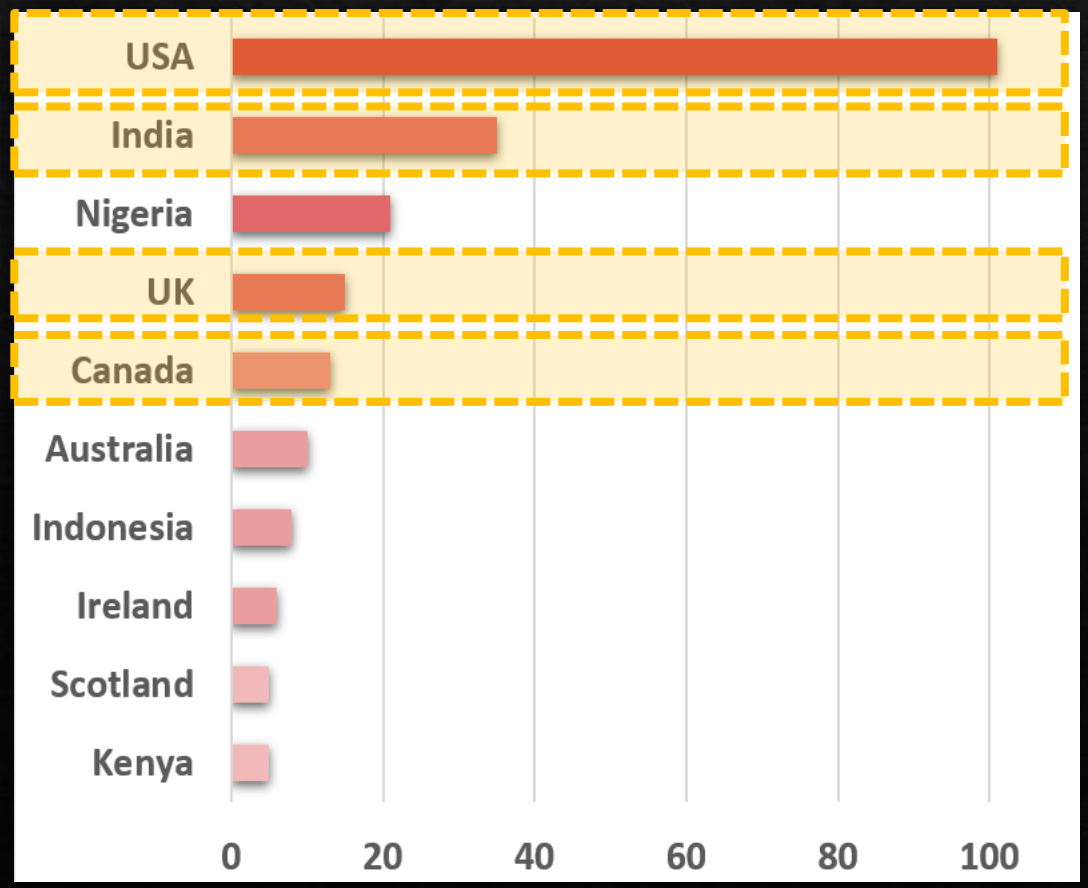


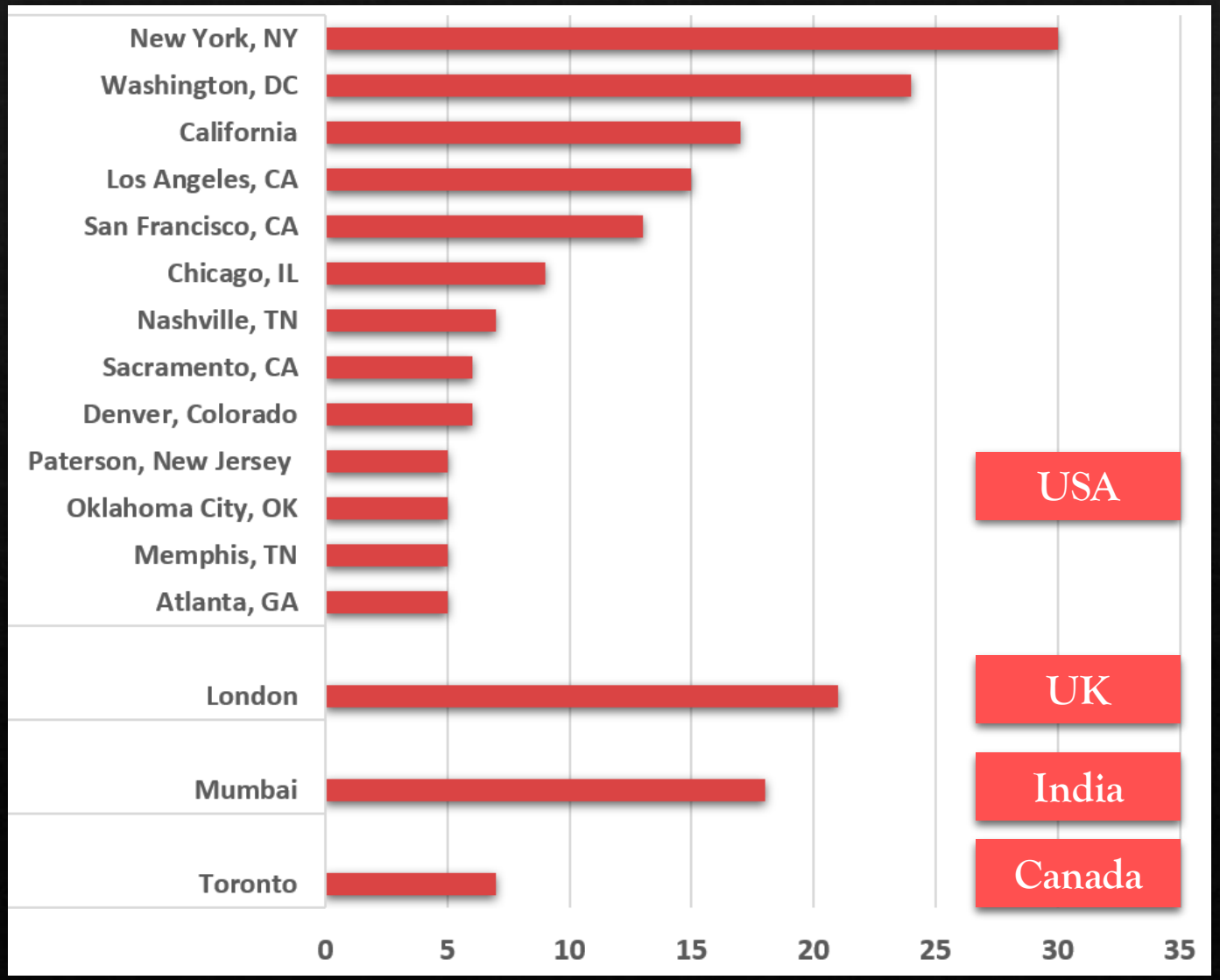
EDA: Location Analysis





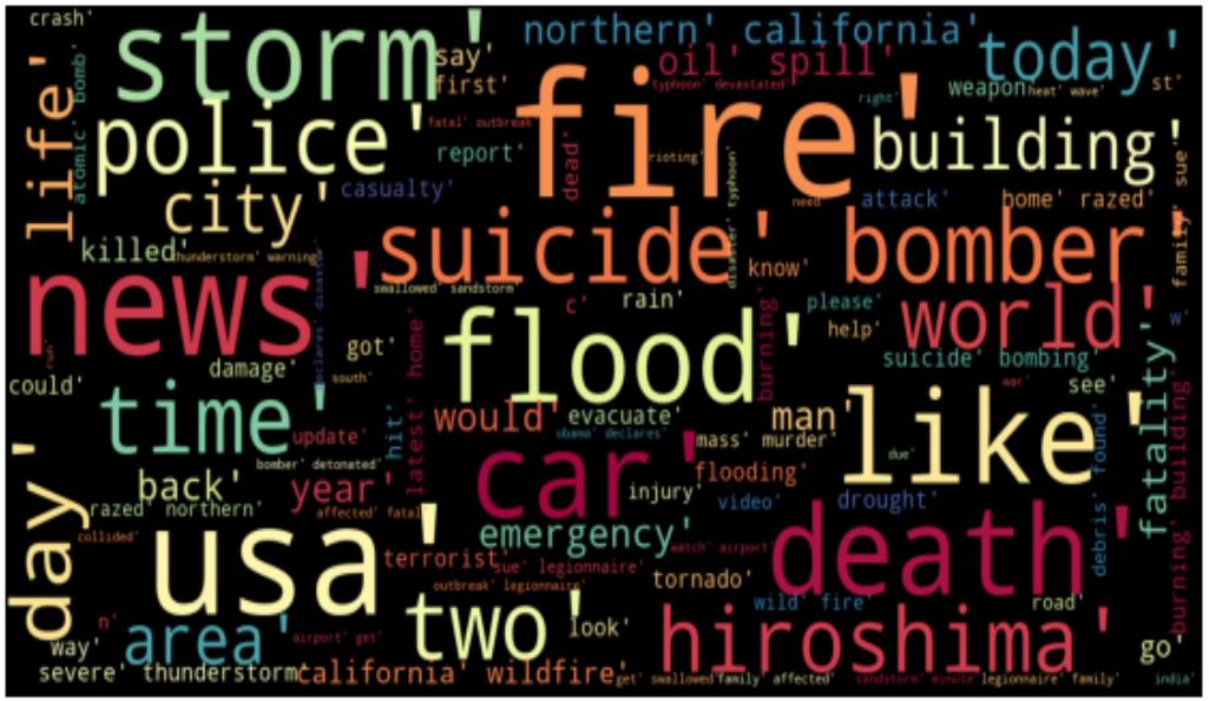
EDA: Location Analysis

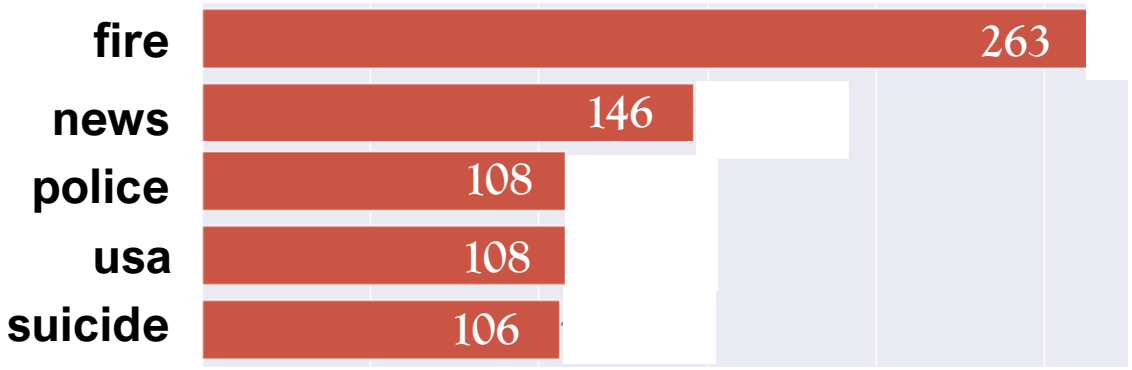




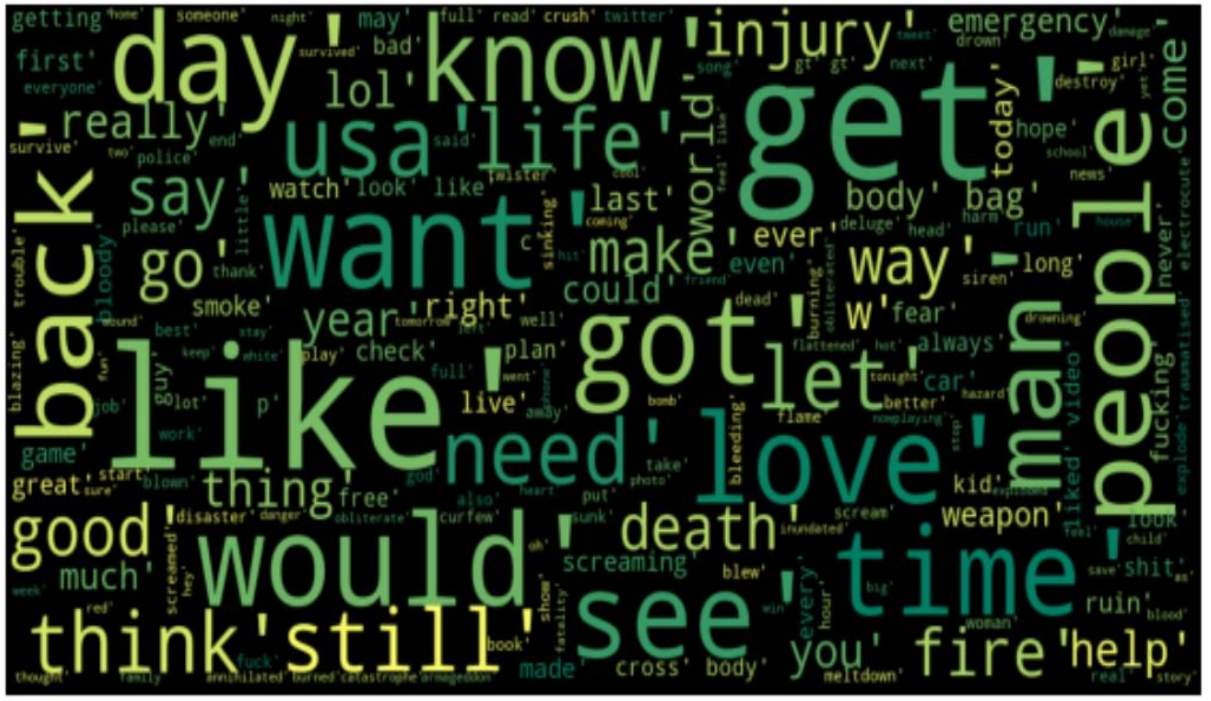
WordCloud: Commonly Used in Tweets

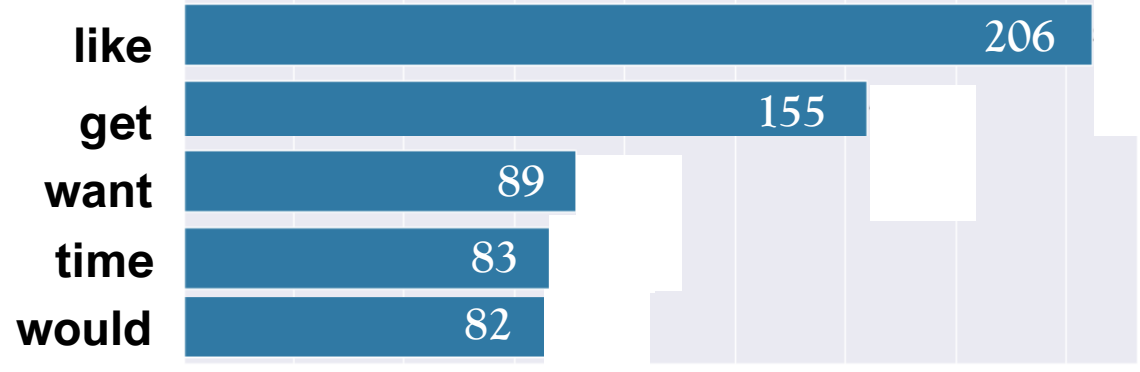
Disaster Tweets | Target = 1



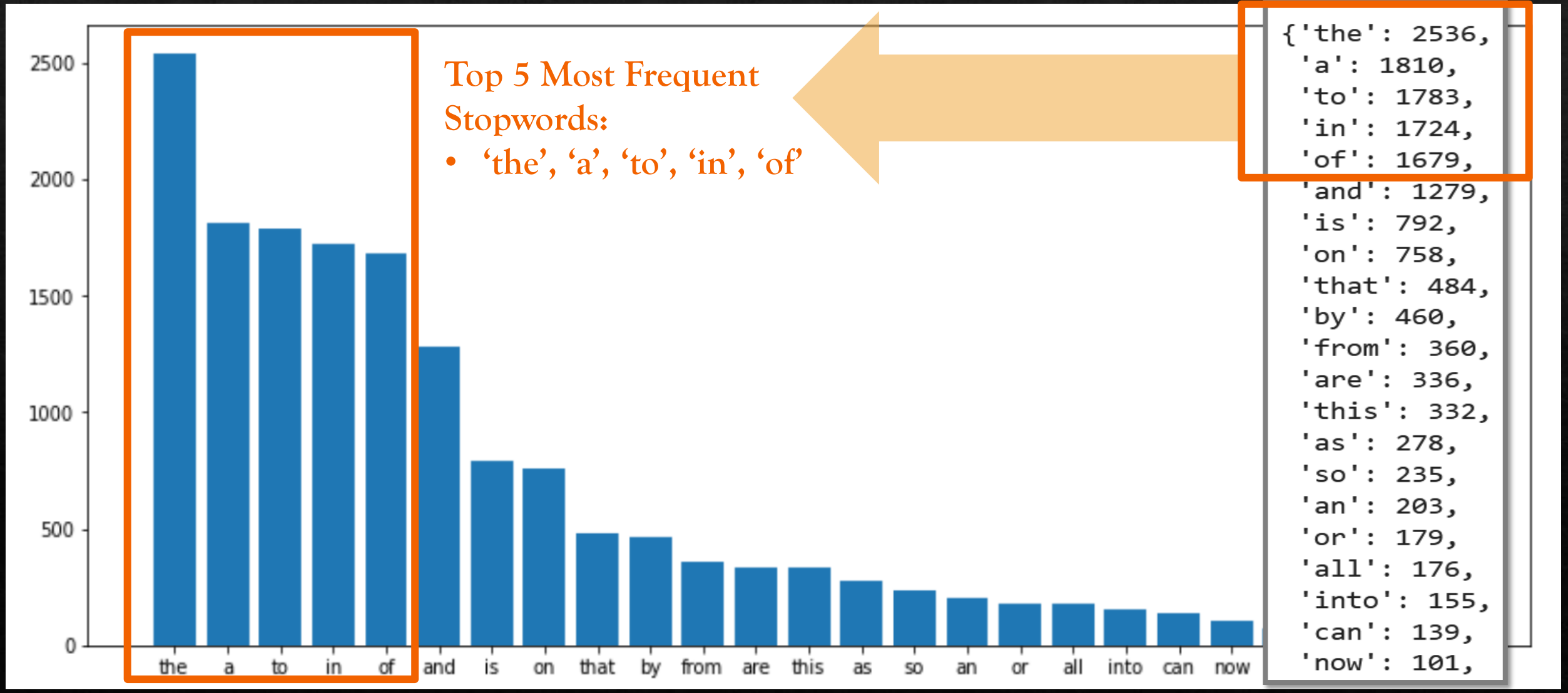


Non-Disaster Tweets | Target = 0





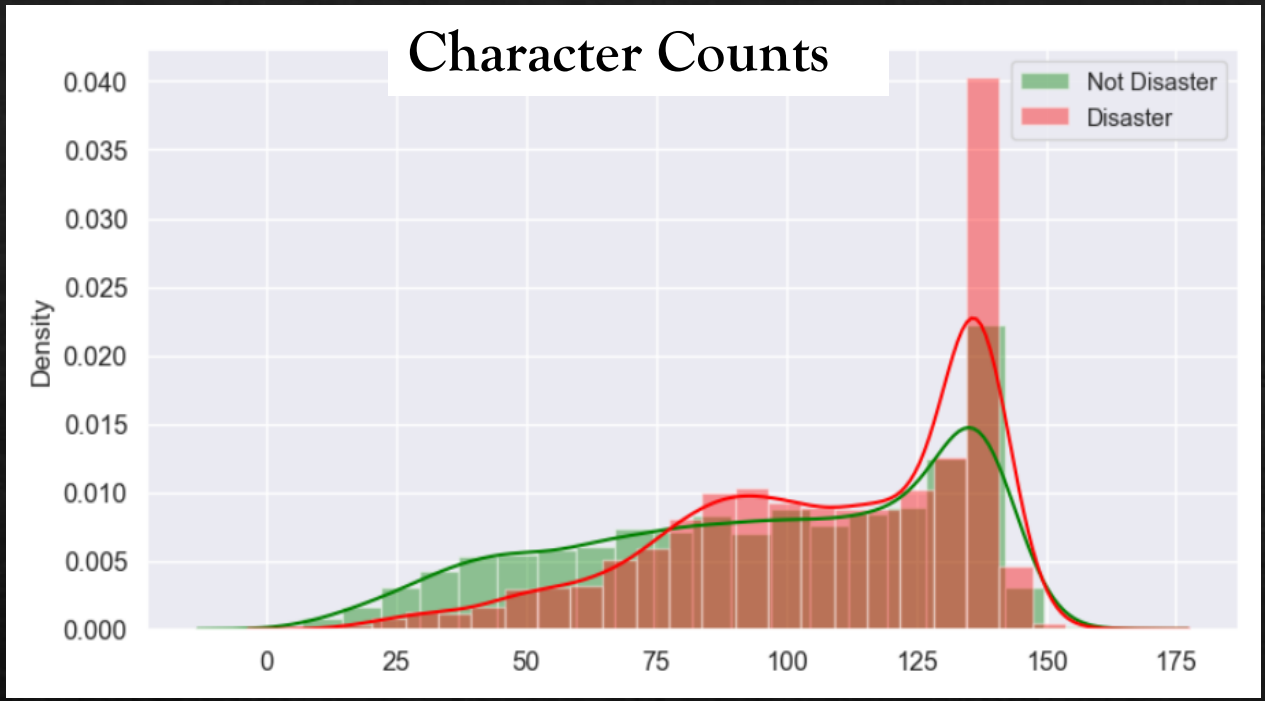
EDA: Stopwords Analysis

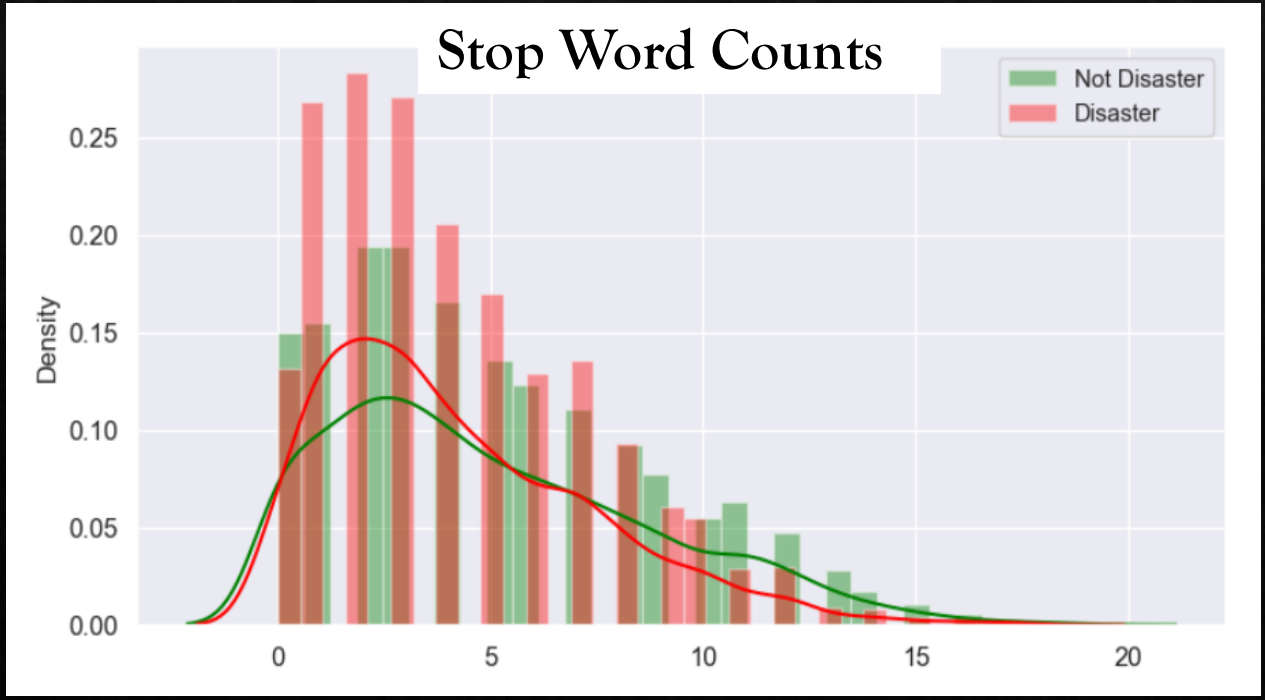


Top 5 Most Frequent Stopwords:

- ‘the’, ‘a’, ‘to’, ‘in’, ‘of’

```
{'the': 2536,
 'a': 1810,
 'to': 1783,
 'in': 1724,
 'of': 1679,
 'and': 1279,
 'is': 792,
 'on': 758,
 'that': 484,
 'by': 460,
 'from': 360,
 'are': 336,
 'this': 332,
 'as': 278,
 'so': 235,
 'an': 203,
 'or': 179,
 'all': 176,
 'into': 155,
 'can': 139,
 'now': 101,
```

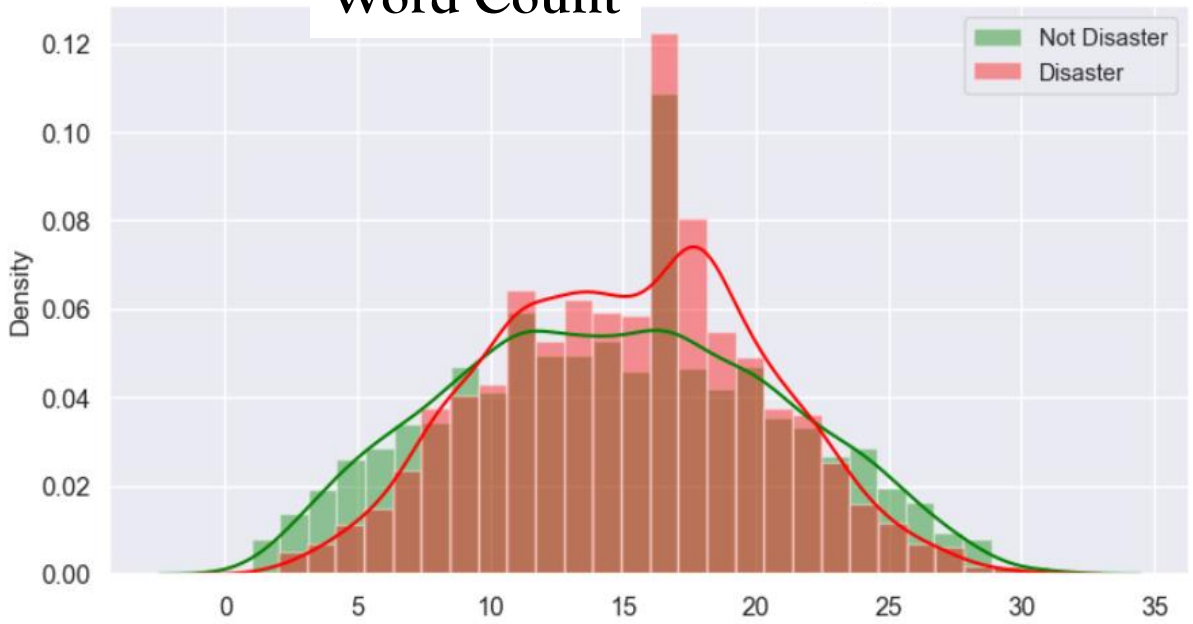




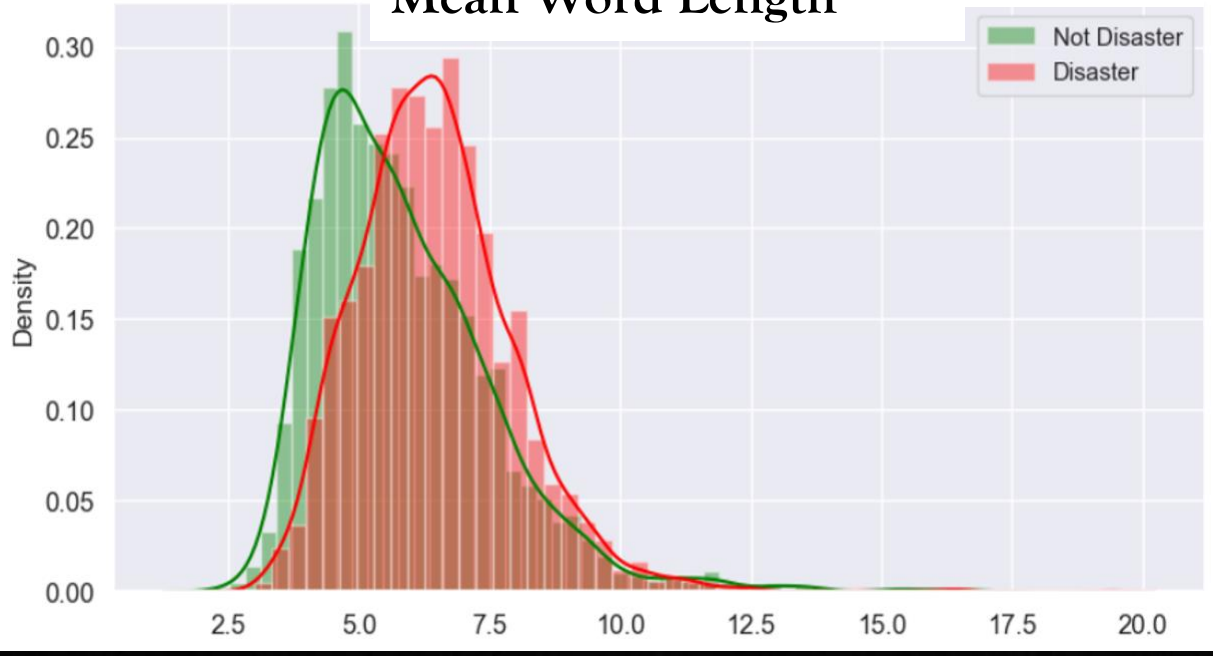
EDA: Distribution Plots #1

EDA: Distribution Plot2 #2

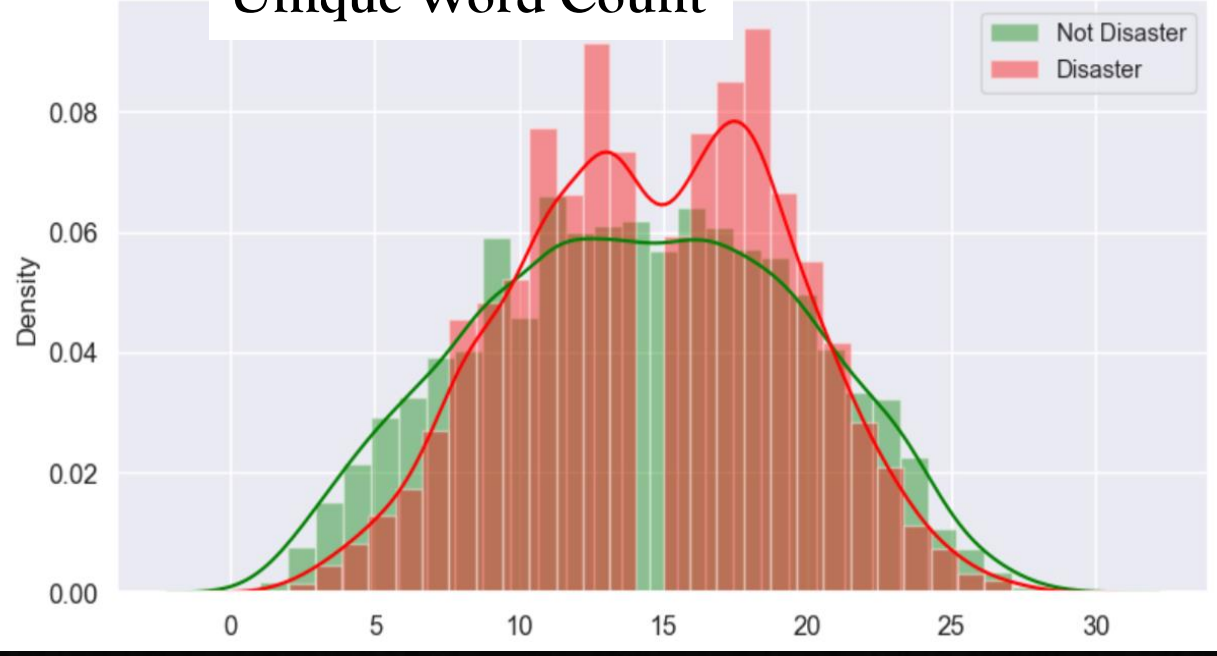
Word Count

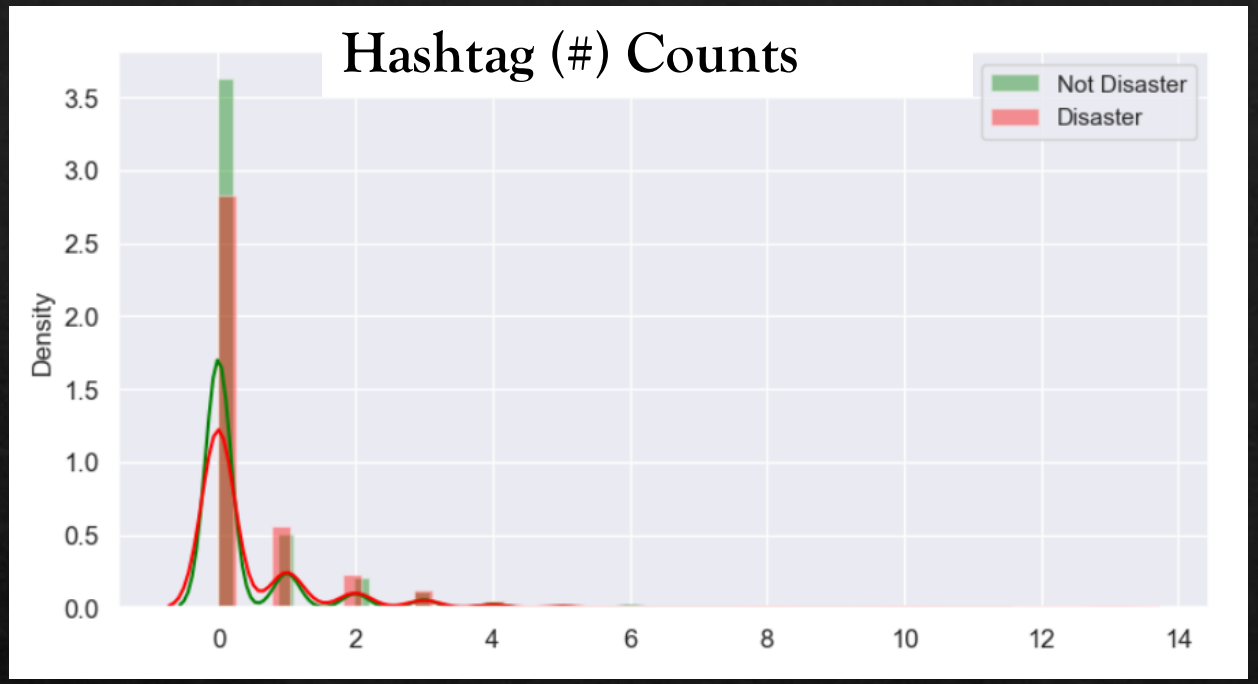


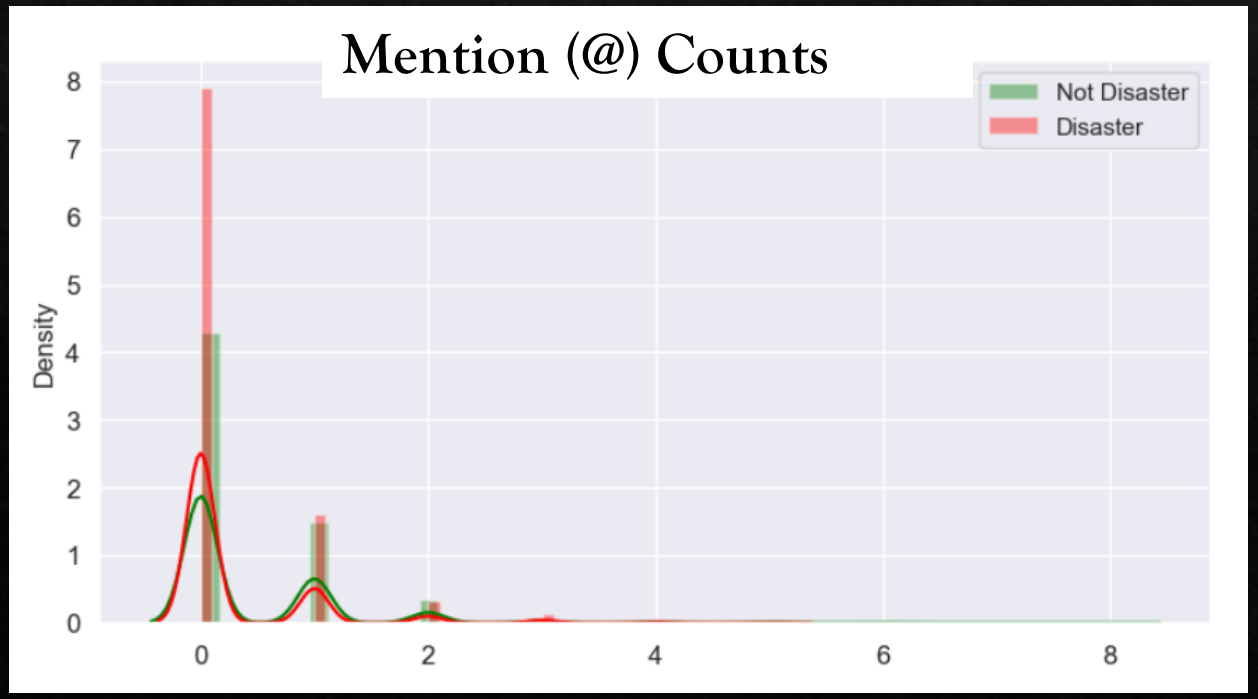
Mean Word Length



Unique Word Count

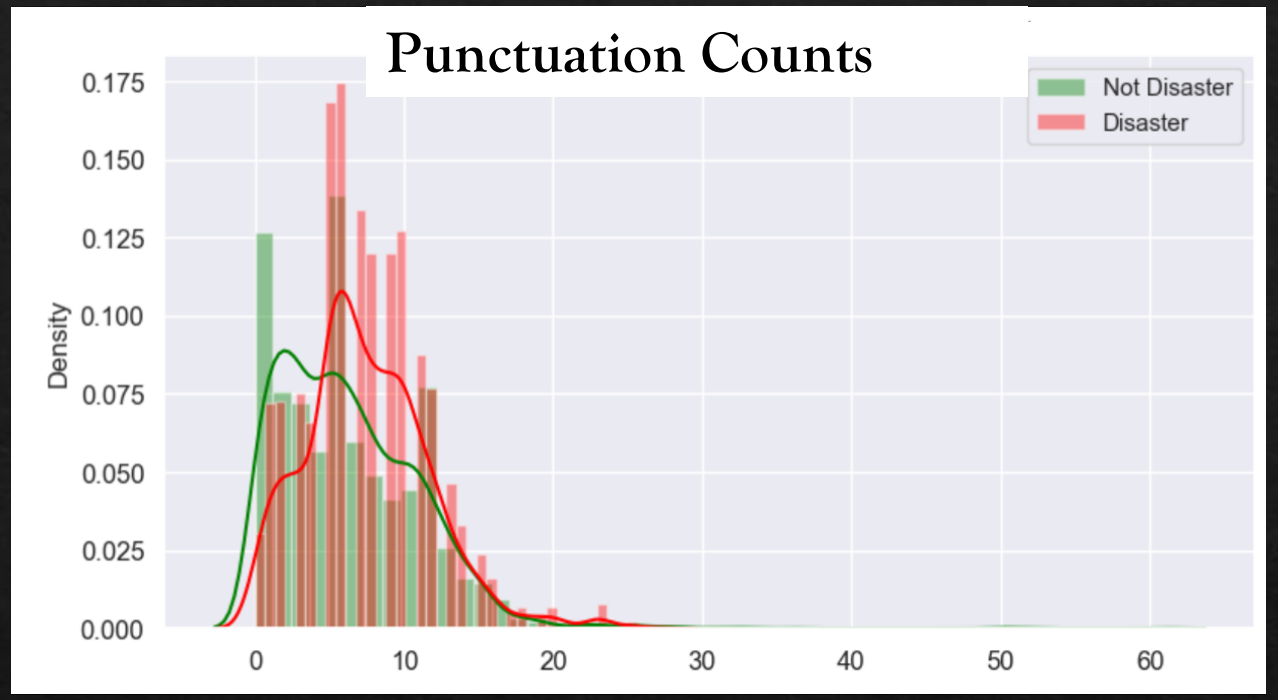


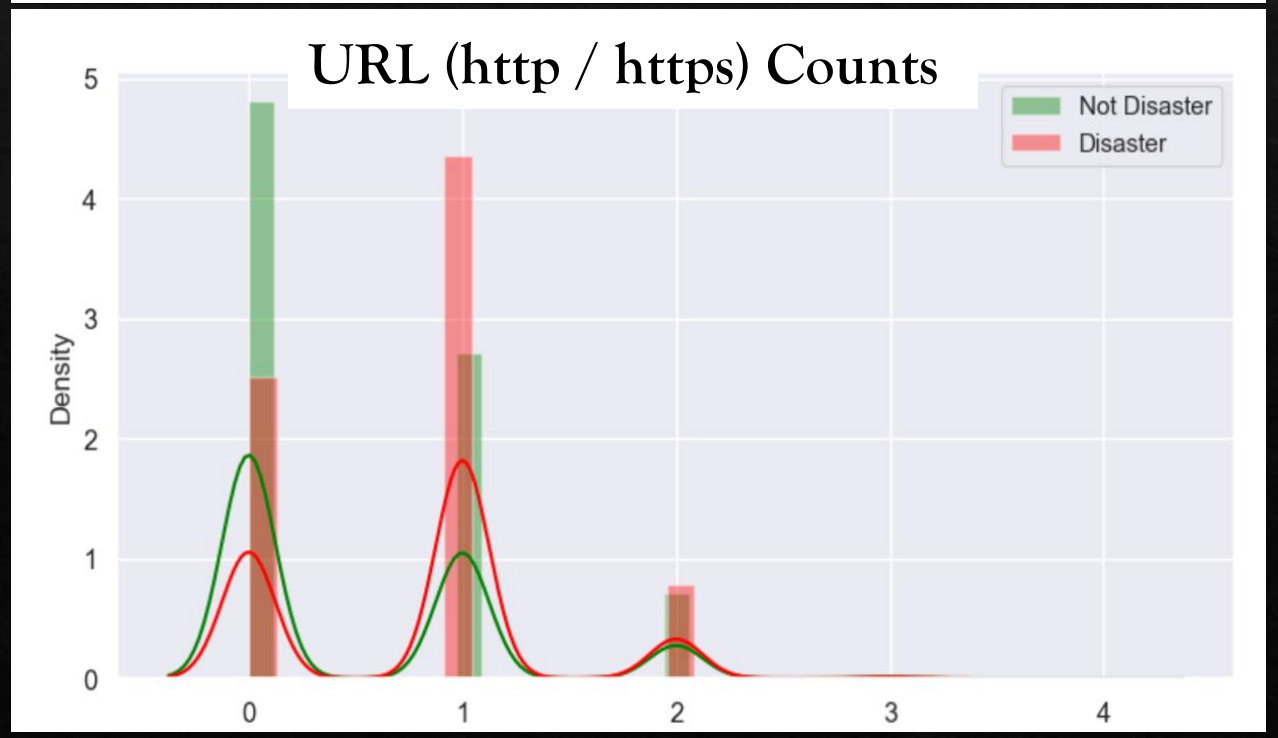




EDA: Distribution Plots #3

EDA: Distribution Plots #4



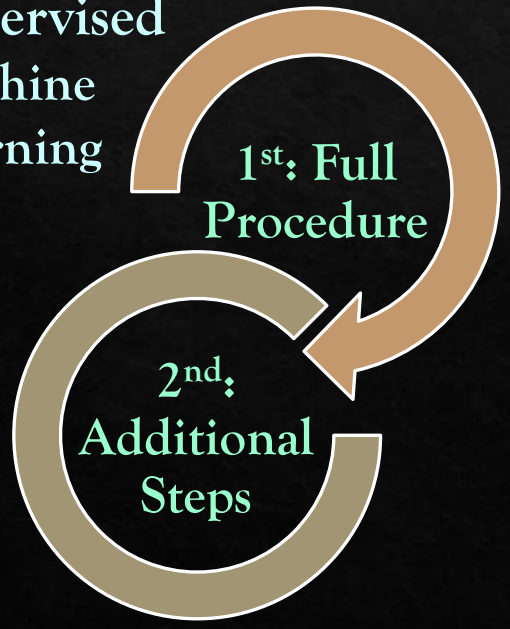


Basic Steps for Text Classification

1

Text Preprocessing

Supervised Machine Learning



Deep Learning / Transformer

TensorFlow / Keras Preprocessing Package

2

Data Representation & Feature Extraction

- Bi-Grams (n = 2)
- Tri-Grams (n = 3)
- LDA Topic Modeling

3

M/L Model Training

Supervised Machine Learning

- Bag of Word (BOW) Vector
- TFIDF Vector

Deep Learning / Transformers

- Layers of Neural Nets
- Pre-trained Features

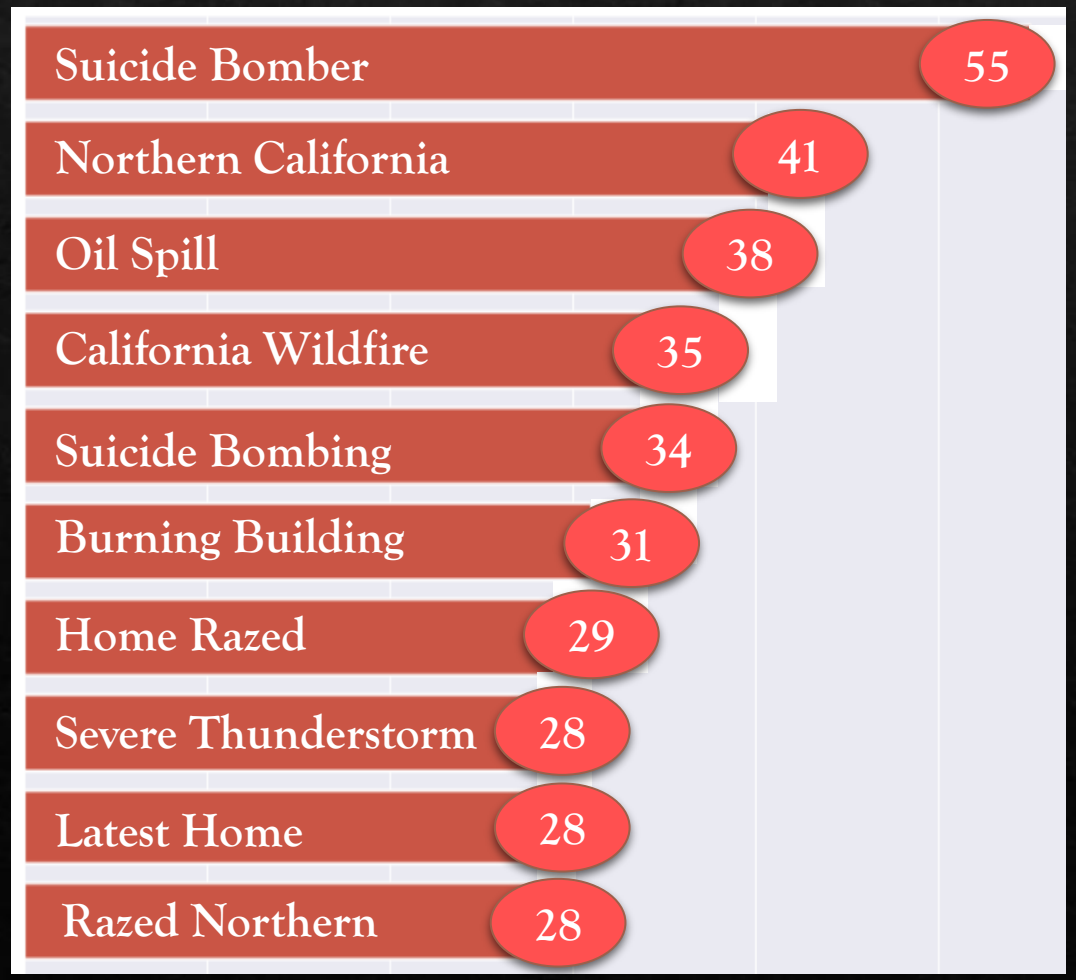
Text Preprocessing

1st Preprocessing:

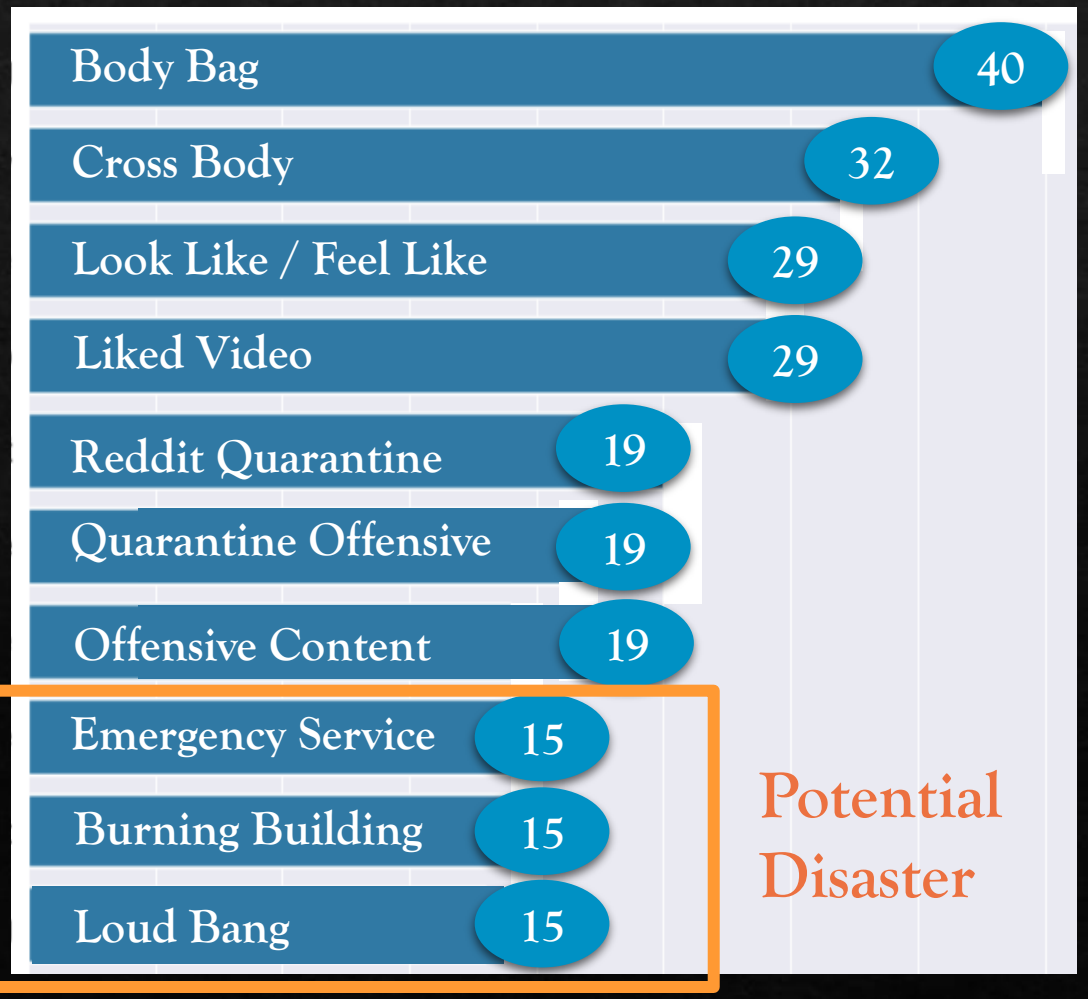
- Replace Abbreviated/ Special Text to Standard Text
- Remove URL
- Remove Contractions
- Tokenize
- Remove Numbers & Unicode
- Remove Punctuation
- Normalize
- Clean Repeated Characters in Text
- Remove Stop Words
- Lemmatize
- Remove Contractions
- Join Tokenized Words to Sentence

Data Representation: Bi-Grams

Disaster Tweets | Target = 1

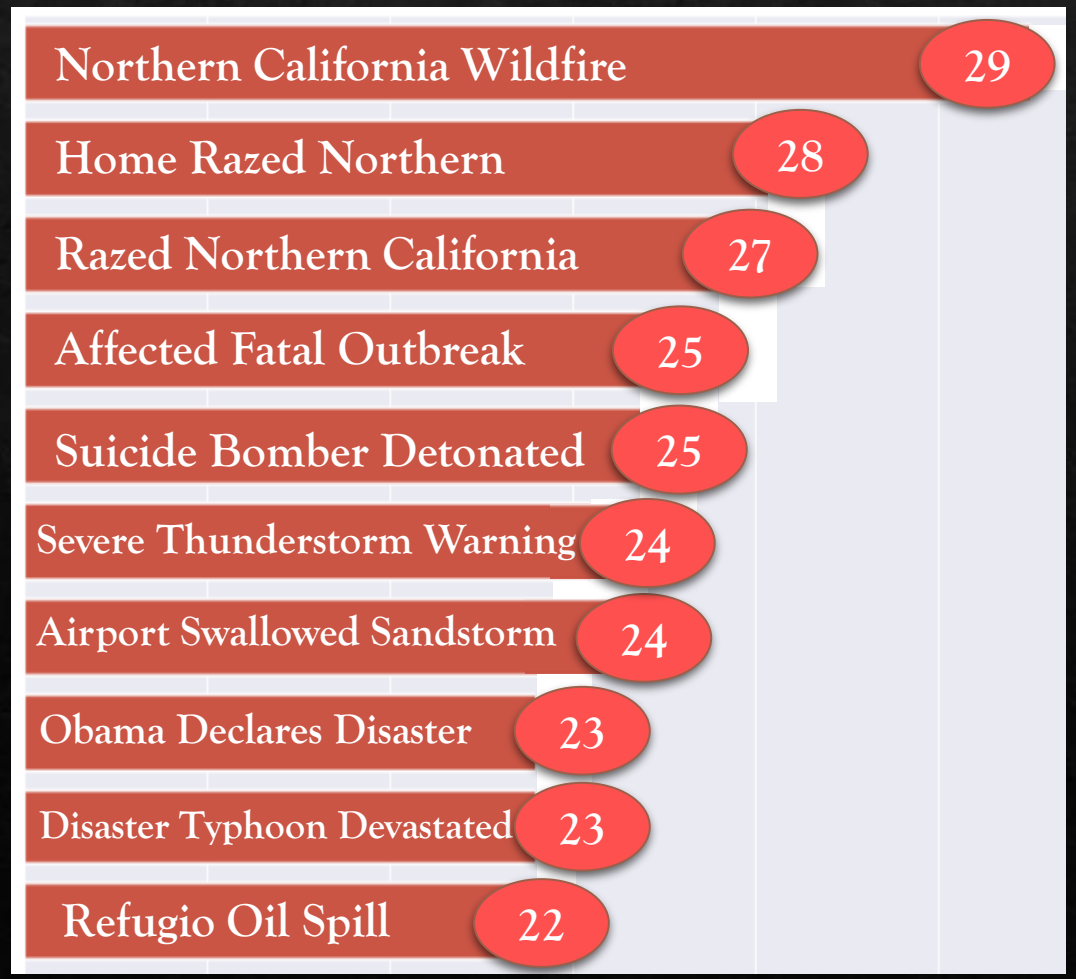


Non-disaster Tweets | Target = 0



Data Representation: Tri-Grams

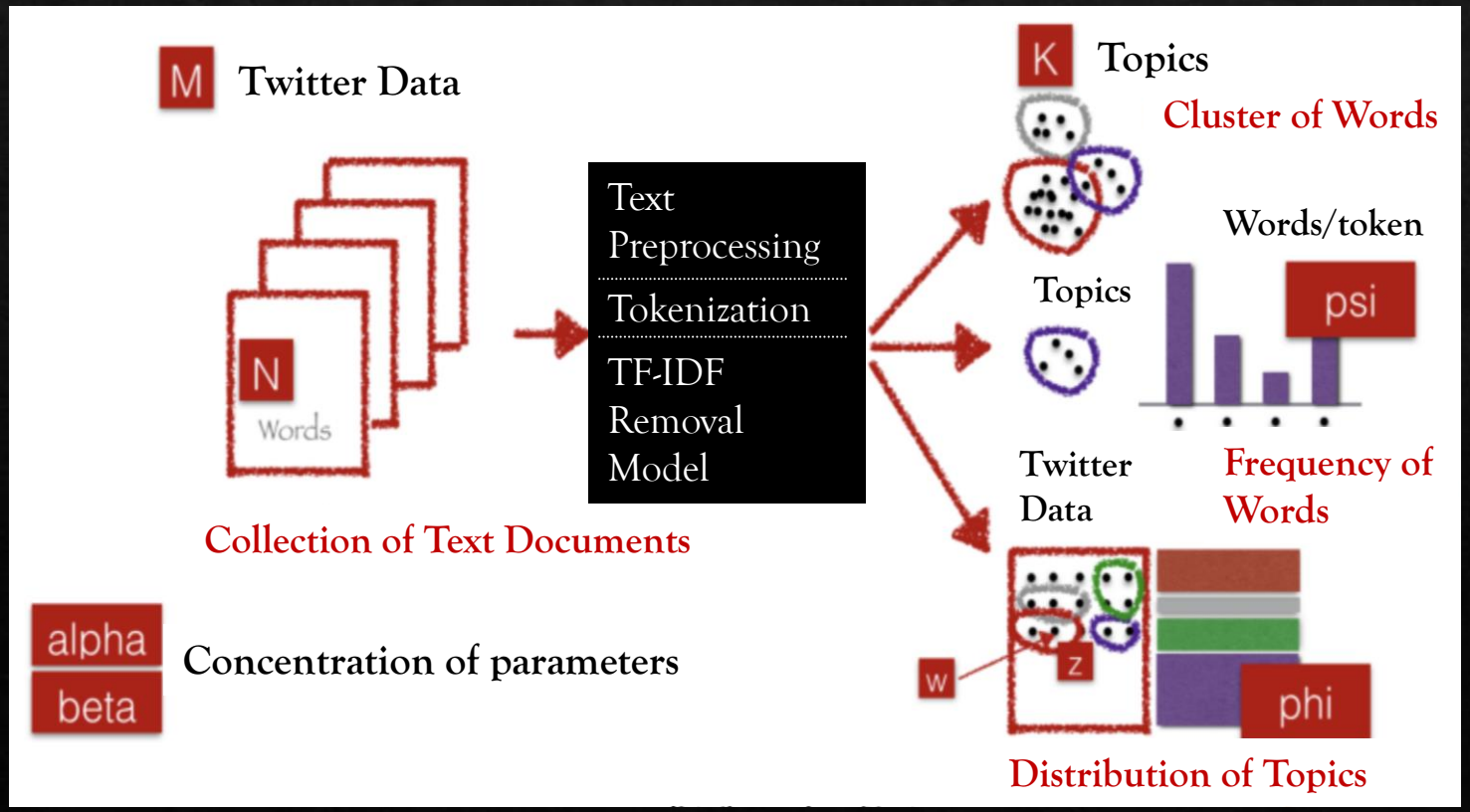
Disaster Tweets | Target = 1



Non-disaster Tweets | Target = 0

LDA Topic Modeling

Latent Dirichlet Allocation (LDA) to determine topics present in Twitter data:



- psi = distribution of words for each topic K
- phi = distribution of topics for each document i

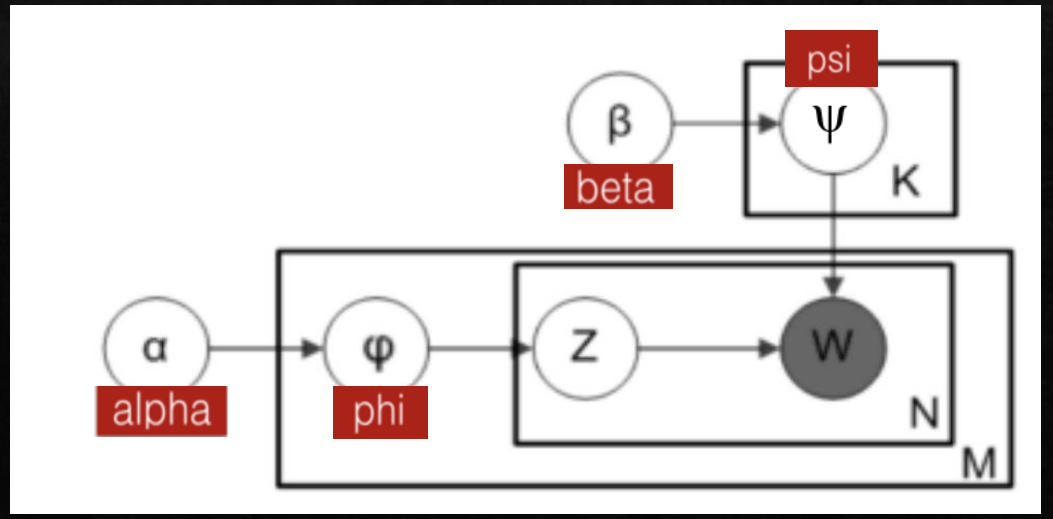


Image Source: https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0

Selected Topic: 0 Previous Topic Next Topic Clear Topic

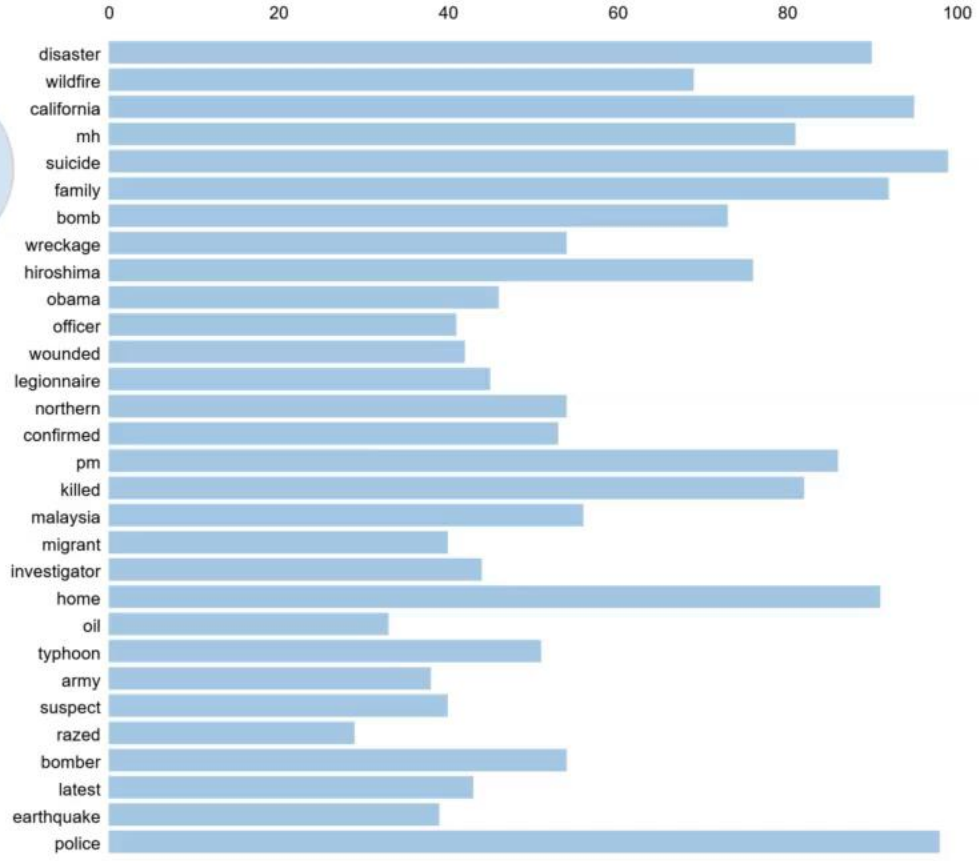
Slide to adjust relevance metric:(2)

λ = 1

0.0 0.2 0.4 0.6 0.8 1.0

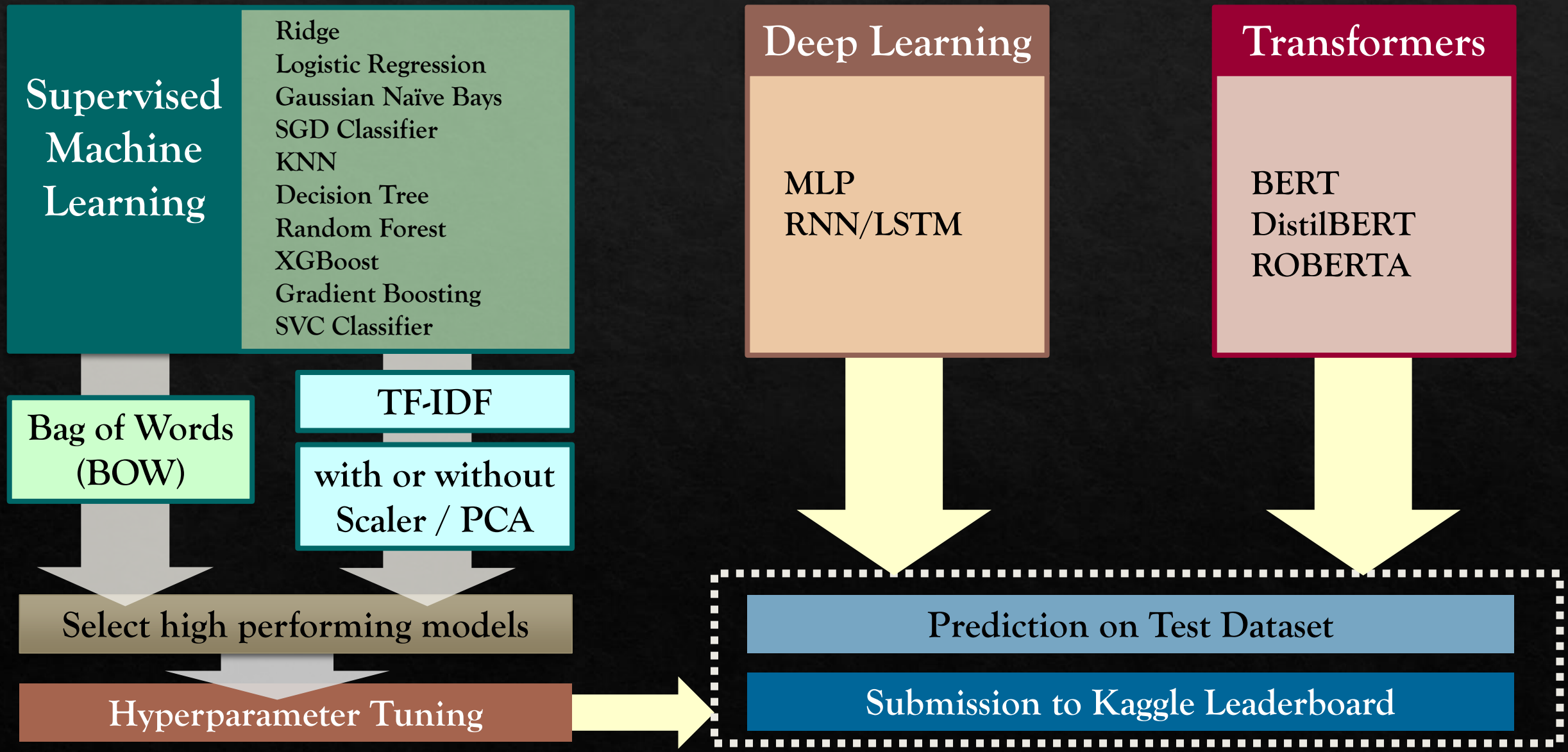
Intertopic Distance Map (via multidimensional scaling)

Top-30 Most Salient Terms1

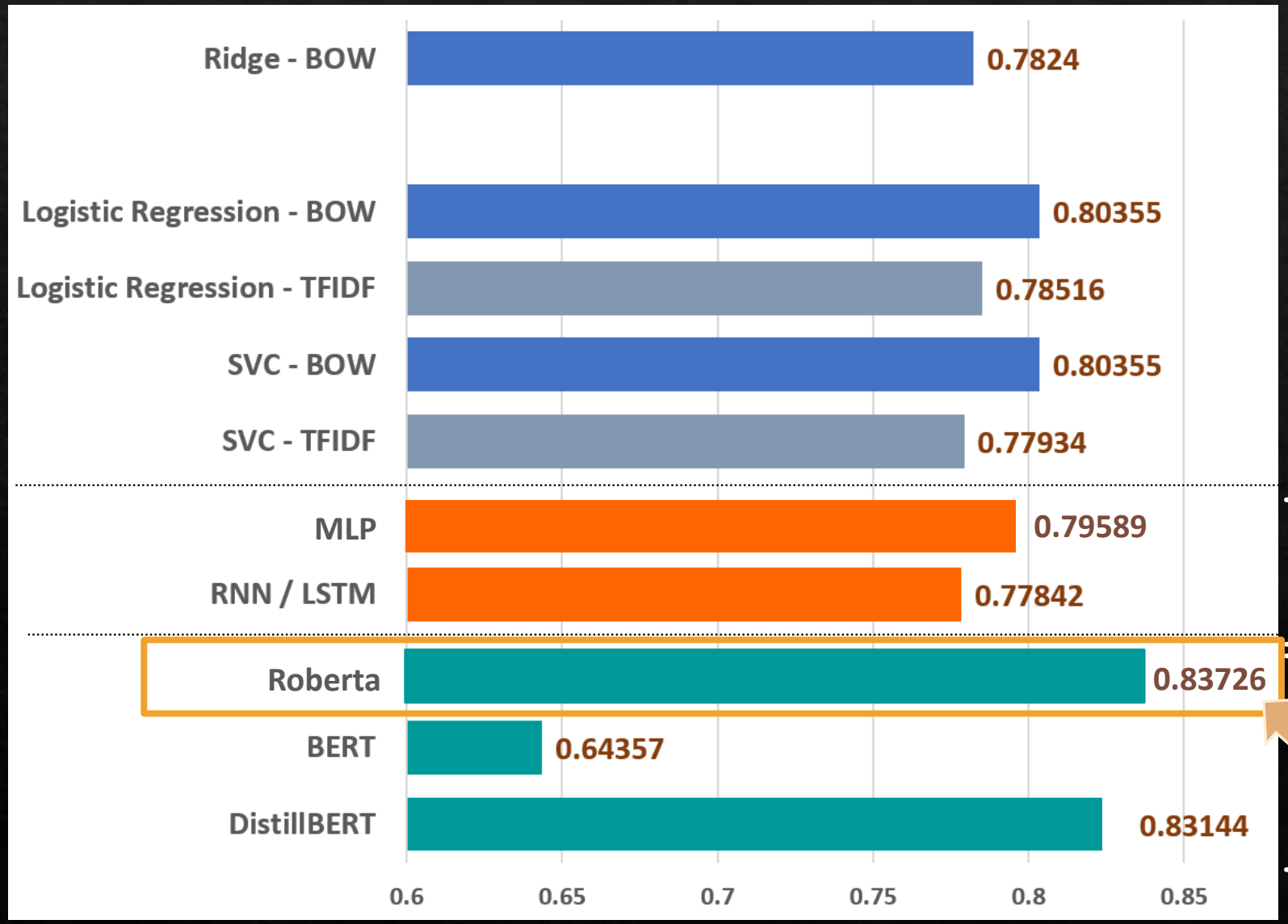


1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Machine Learning – Training Process



Machine Learning – Classification Performance



Kaggle – Best F1 Scores

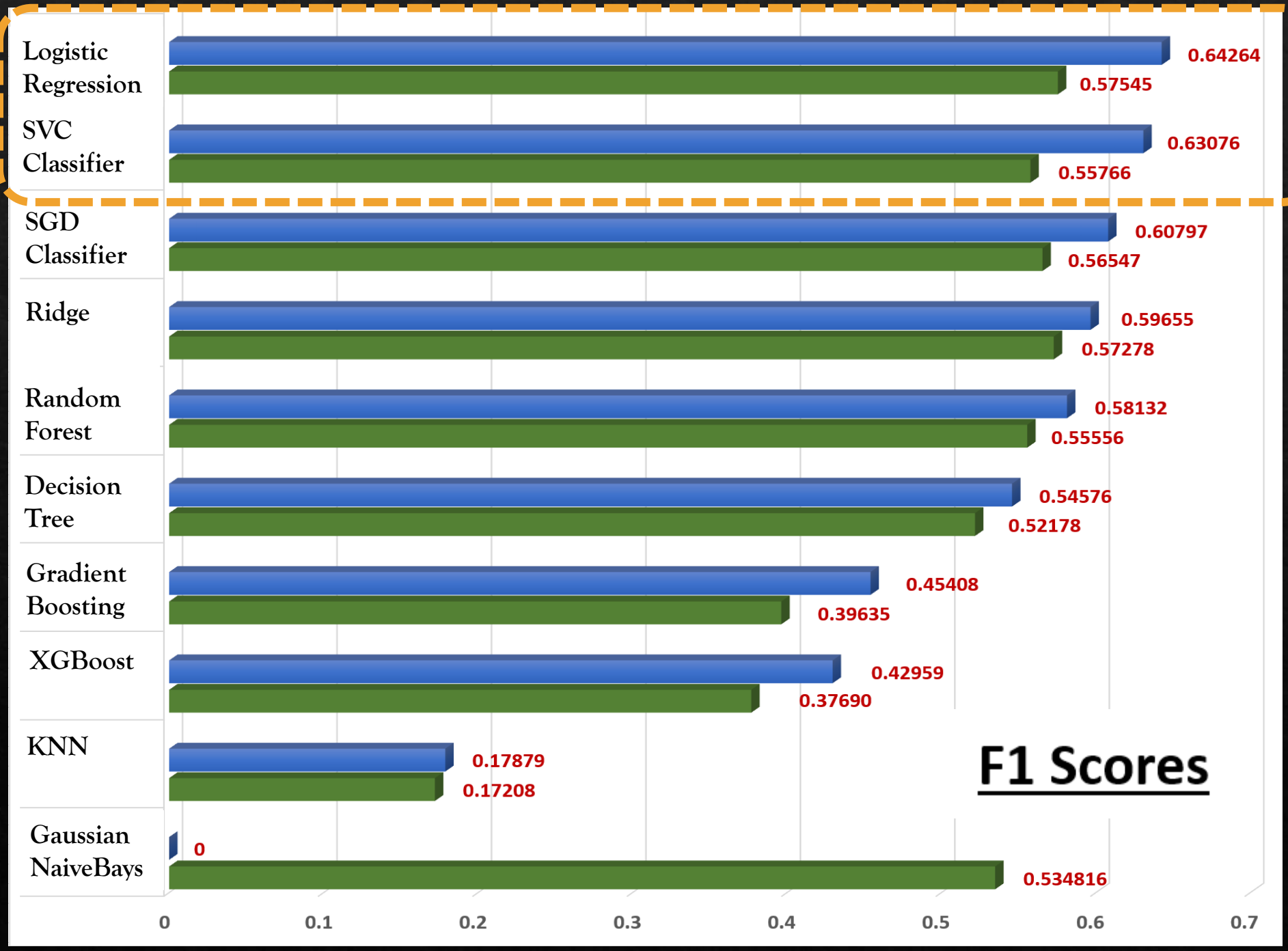
Supervised Machine Learning

Deep Learning

Transformers

Best Performing Model

Supervised M/L: Performance Summary

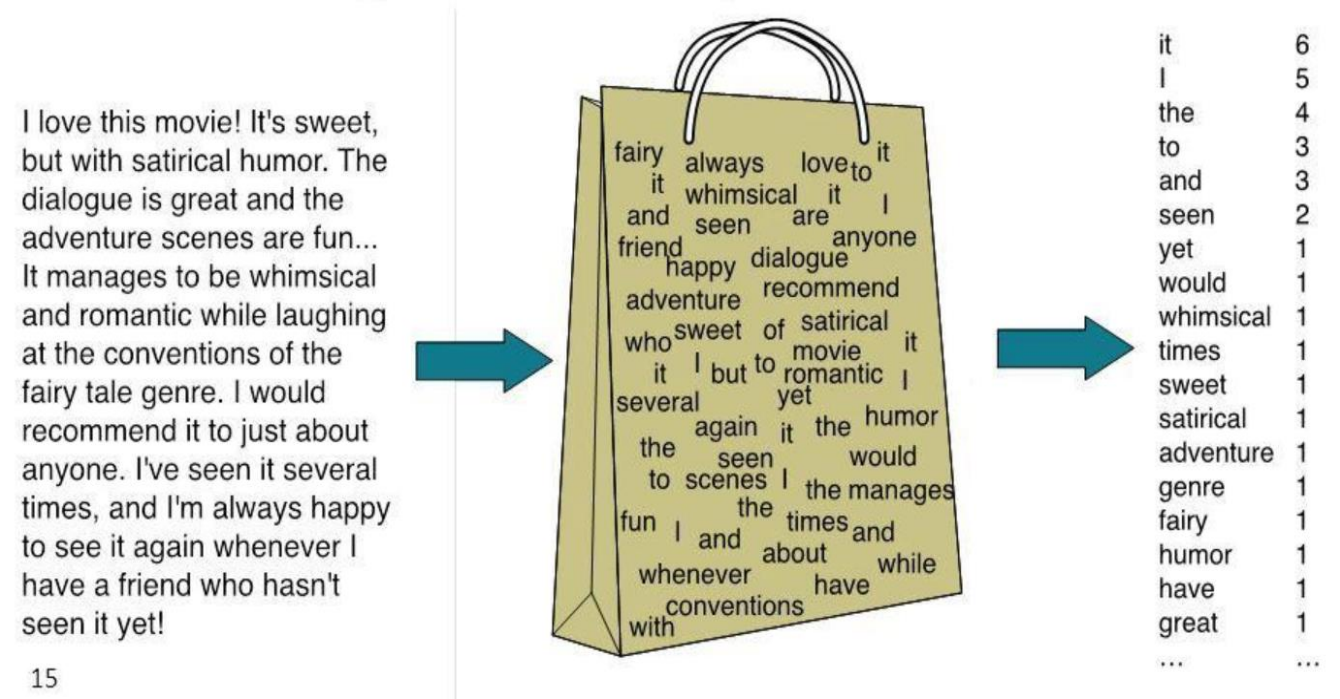


Models Selected for Hyperparameter Tuning

=> BOW

=> TF-IDF

Feature Extraction: Bag of Words (BOW)



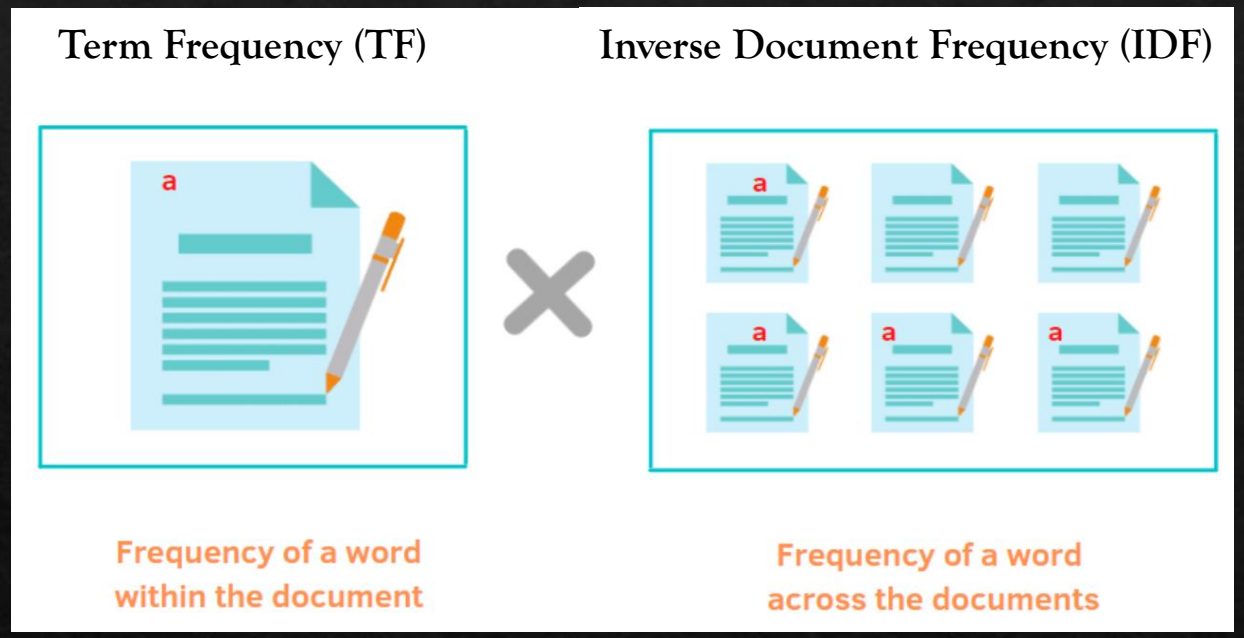

```
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = feature_extraction.text.CountVectorizer()

train_vectors = count_vectorizer.fit_transform(train_df['text_preprocessed_2nd'])
```

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

- Extraction of text features for use in machine learning algorithm, where the representation of text describes the occurrence of words within a document
- Extraction involves: vocabulary of known words & measure of presence of known words
- Converts variable-length texts into a fixed-length vector of numbers

Image Source: https://sep.com/blog/a-bag-of-words-levels-of-language/

Feature Extraction: TF-IDF



- Numerical metric reflecting the importance of a word in a collection of corpus
- Higher TF-IDF score for frequent words in a document but not across documents

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(ngram_range=(1, 2), max_features=1200)
X = tfidf_vect.fit_transform(train_df['text_preprocessed_2nd'])
target = pd.DataFrame(X.toarray(), columns=tfidf_vect.get_feature_names())
```

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

$$TF_{x,y} = \frac{f_{x,y}}{n_y}$$

$f_{x,y}$ = frequency of term x in document y
n_y = total number of words in document y

$$IDF_x = 1 + log\left(\frac{N}{df_x}\right)$$

Image Source: http://filotechnologia.blogspot.com/2014/01/a-simple-java-class-for-tfidf-scoring.html
https://ted-mei.medium.com/demystify-tf-idf-in-indexing-and-ranking-5c3ae88c3fa0
https://medium.com/codex/document-indexing-using-tf-idf-189afd04a9fc

Dimensionality Reduction & Data Normalization

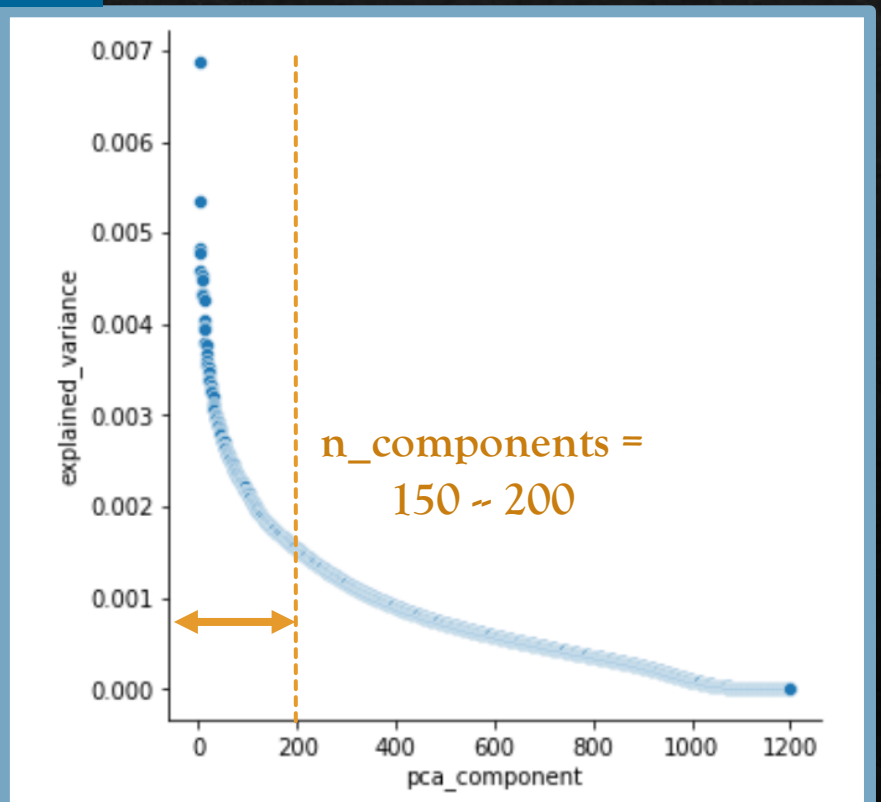
Data Normalization: MinMaxScaler

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

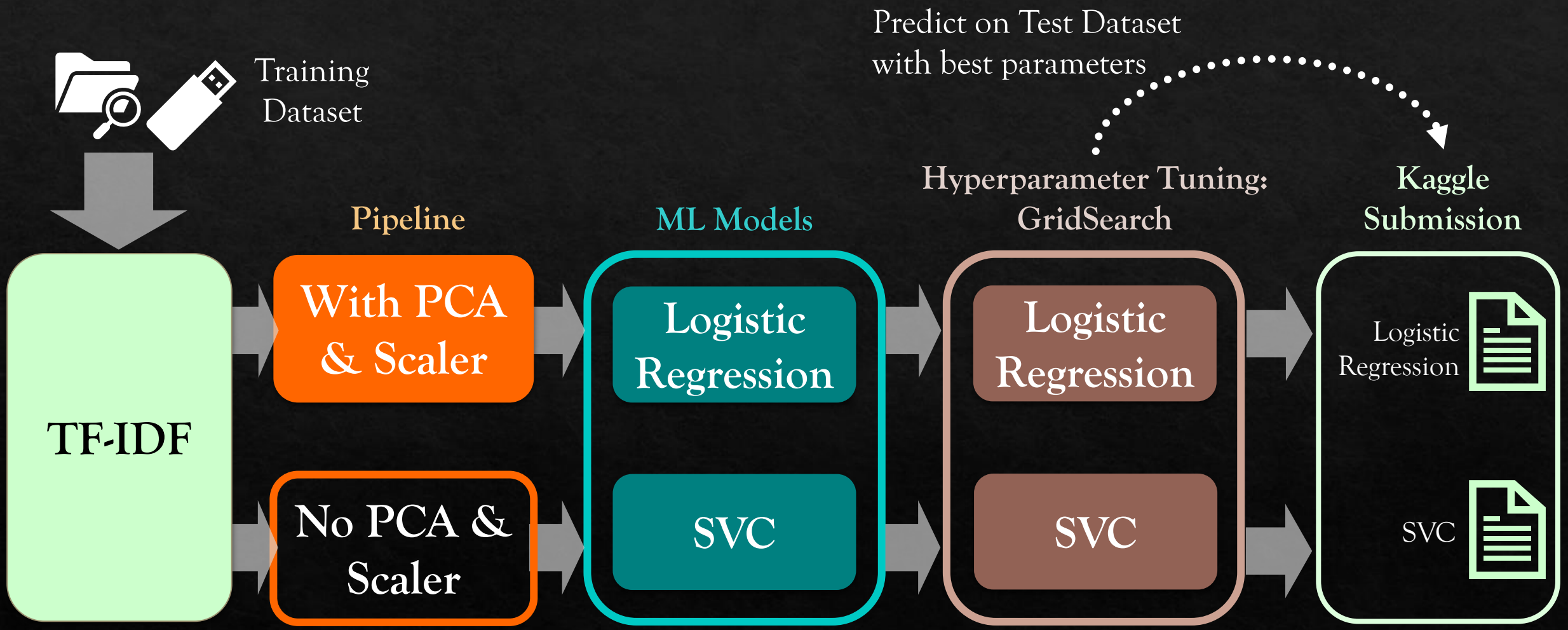
- Normalized Twitter datasets, using the scikit-learn object MinMaxScaler
- Default scale for Normalization: to rescale variables into the range [0,1]

Dimensionality Reduction (PCA)

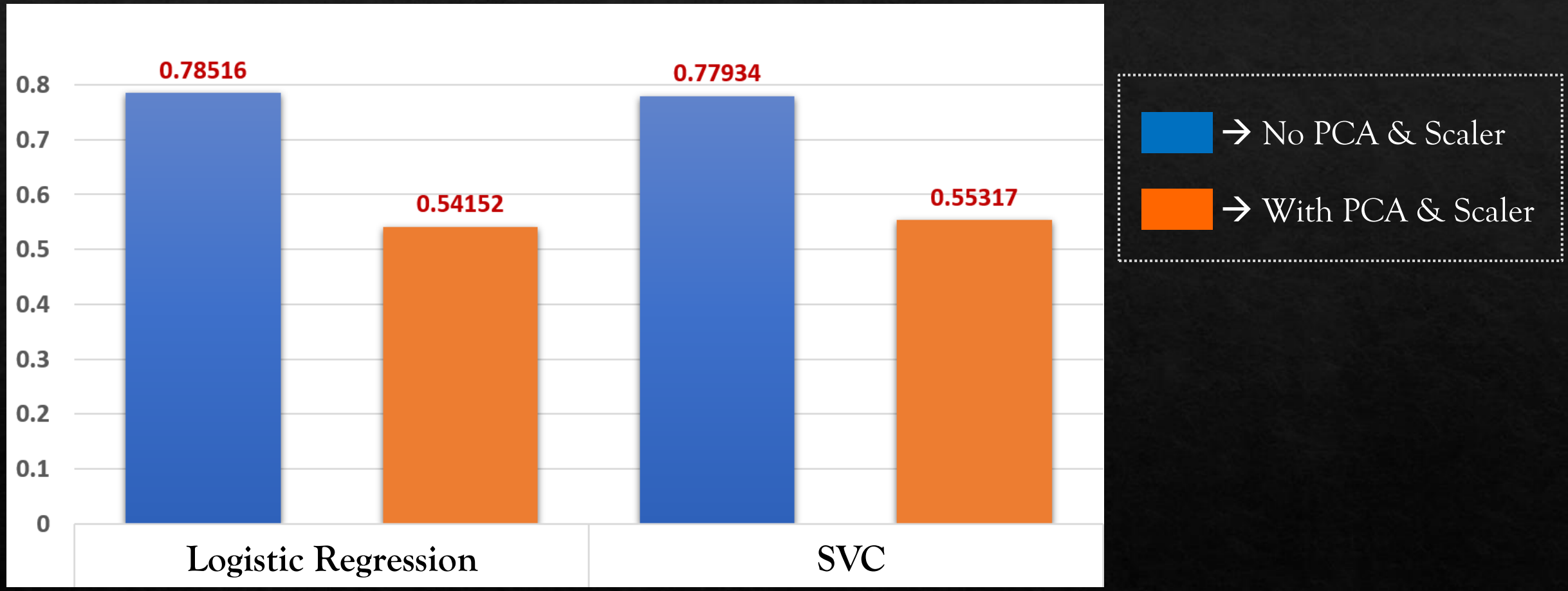
	explained_variance	pca_component	pca_100%
0	6.874733e-03	1	0.6874733
1	5.339239e-03	2	0.5339239
2	4.840314e-03	3	0.4840314
3	4.774374e-03	4	0.4774374
4	4.596527e-03	5	0.4596527
...	...	...	...
1195	6.827159e-36	1196	6.827159e-34
1196	3.230210e-36	1197	3.230210e-34
1197	1.398170e-36	1198	1.398170e-34
1198	1.039025e-36	1199	1.039025e-34
1199	4.694233e-38	1200	4.694233e-36



TF-IDF: Scaler and PCA



TF-IDF: Scaler and PCA -- Kaggle F1 Scores



Hyperparameter Tuning: Logistic Regression

With PCA & Scaler — 2.31 minutes

```
scaler  = MinMaxScaler()
pca  = PCA(n_components = 200)
```

```
param_grid = {
  'solver': [ 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' ],
  'penalty': [ 'l2' ],
  'C': [ 1e-05, 0.001, 0.1, 1.0, 10.0, 100.0 ],
  'random_state': [ 10, 25, 35, 42, 53 ]  }
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3)
```

```
grid_lg = GridSearchCV(clf_lg, param_grid, n_jobs = -1,
                       scoring='f1', cv=cv, verbose = 3)
```

```
clf_lg2 = LogisticRegression(C=100, penalty='l2',
                             random_state=35, solver='saga')
```

Kaggle F1 Score: **0.54152**

No PCA & Scaler — 8.42 minutes

```
scaler → None
pca → None
```

```
param_grid = {
  'solver': [ 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' ],
  'penalty': [ 'l2' ],
  'C': [ 1e-05, 0.001, 0.1, 1.0, 10.0, 100.0 ],
  'random_state': [ 10, 25, 35, 42, 53 ]  }
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3)
```

```
grid_lg = GridSearchCV(clf_lg, param_grid, n_jobs = -1,
                       scoring='f1', cv=cv, verbose = 3)
```

```
clf_lg3 = LogisticRegression(C=10.0, penalty='l2',
                             random_state=42, solver='sag')
```

Kaggle F1 Score: **0.78516**

Hyperparameter Tuning: SVC

With PCA & Scaler — 2.33 hours

```
scaler  = MinMaxScaler()
pca  = PCA(n_components = 200)
```

```
param_grid = {
   'C': [ 1e-05, 0.001, 0.1, 1.0, 10.0, 100.0 ],
   'gamma': [ 1, 0.1, 0.01, 0.001, 0.0001 ],
   'kernel': [ 'linear' 'rbf' ],
   'max_iter': [ -1 ],
   'random_state': [ 10, 25, 35, 42, 53 ]  }
```

```
grid_svc = GridSearchCV( clf_svc, param_grid,
                         refit = True, verbose = 3)
```

```
clf_svc = SVC( C=100, gamma = 0.01,  kernel = 'rbf',
               max_iter = -1, random_state=10 )
```

Kaggle F1 Score **0.55317**

No PCA & Scaler — 15.78 minutes

```
scaler → None
pca → None
```

```
param_grid = {
   'C': [ 1e-05, 0.001, 0.1, 1.0, 10.0, 100.0 ],
   'gamma': [ 1, 0.1, 0.01, 0.001, 0.0001 ],
   'kernel': [ 'linear' 'rbf' ],
   'max_iter': [ -1 ],
   'random_state': [ 10, 25, 35, 42, 53 ]  }
```

```
grid_svc2 = HalvingGridSearchCV( clf_svc, param_grid,
                                 refit = True, verbose = 3)
```

```
clf_svc2 = SVC( C=0.1, gamma = 0.1,  kernel = 'linear',
                max_iter = -1, random_state=10 )
```

Kaggle F1 Score **0.77943**

Deep Learning: MLP

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification

mlp2=MLPClassifier(random_state=0, early_stopping=True, max_iter=100, verbose=2)
```

Multi-layer Perceptron (MLP): supervised learning algorithm that learns a function, $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where:

→ m = the number of dimensions for input

→ o = the number of dimensions for output

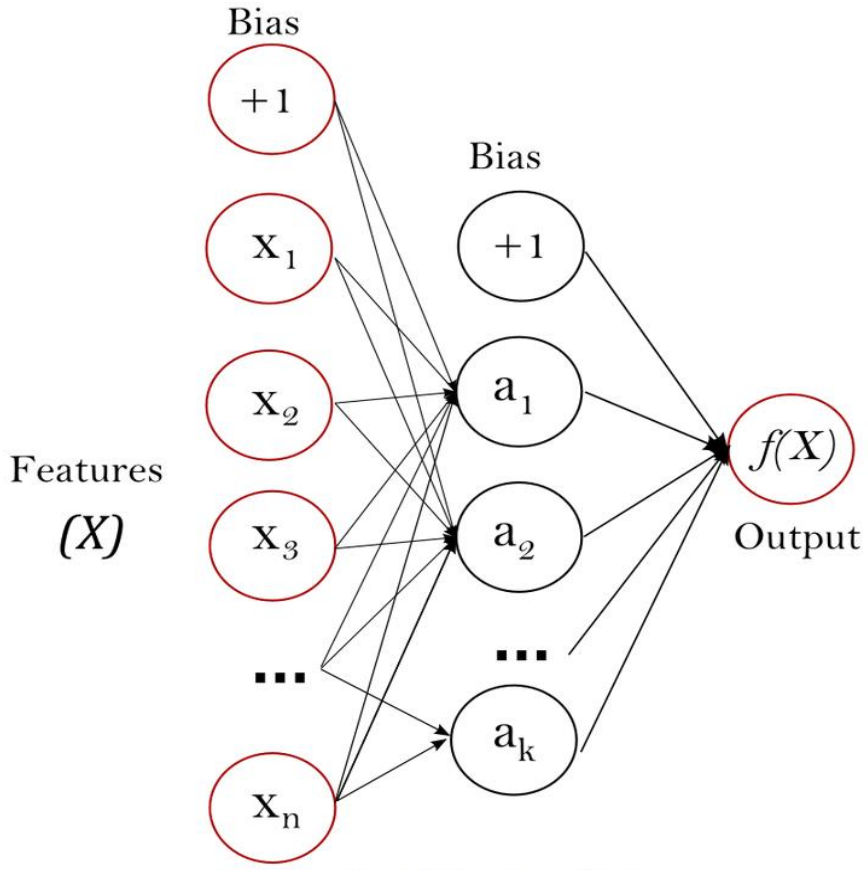
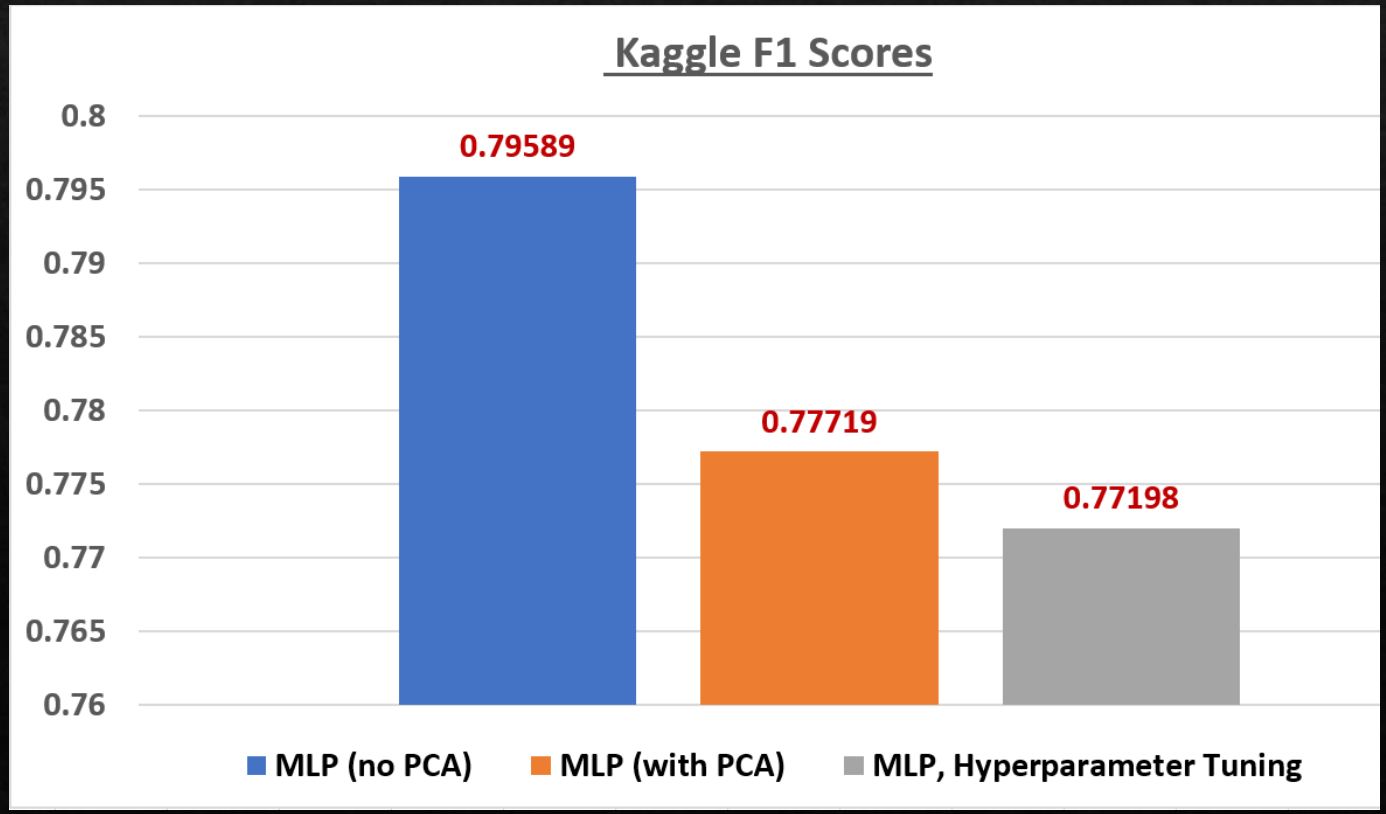
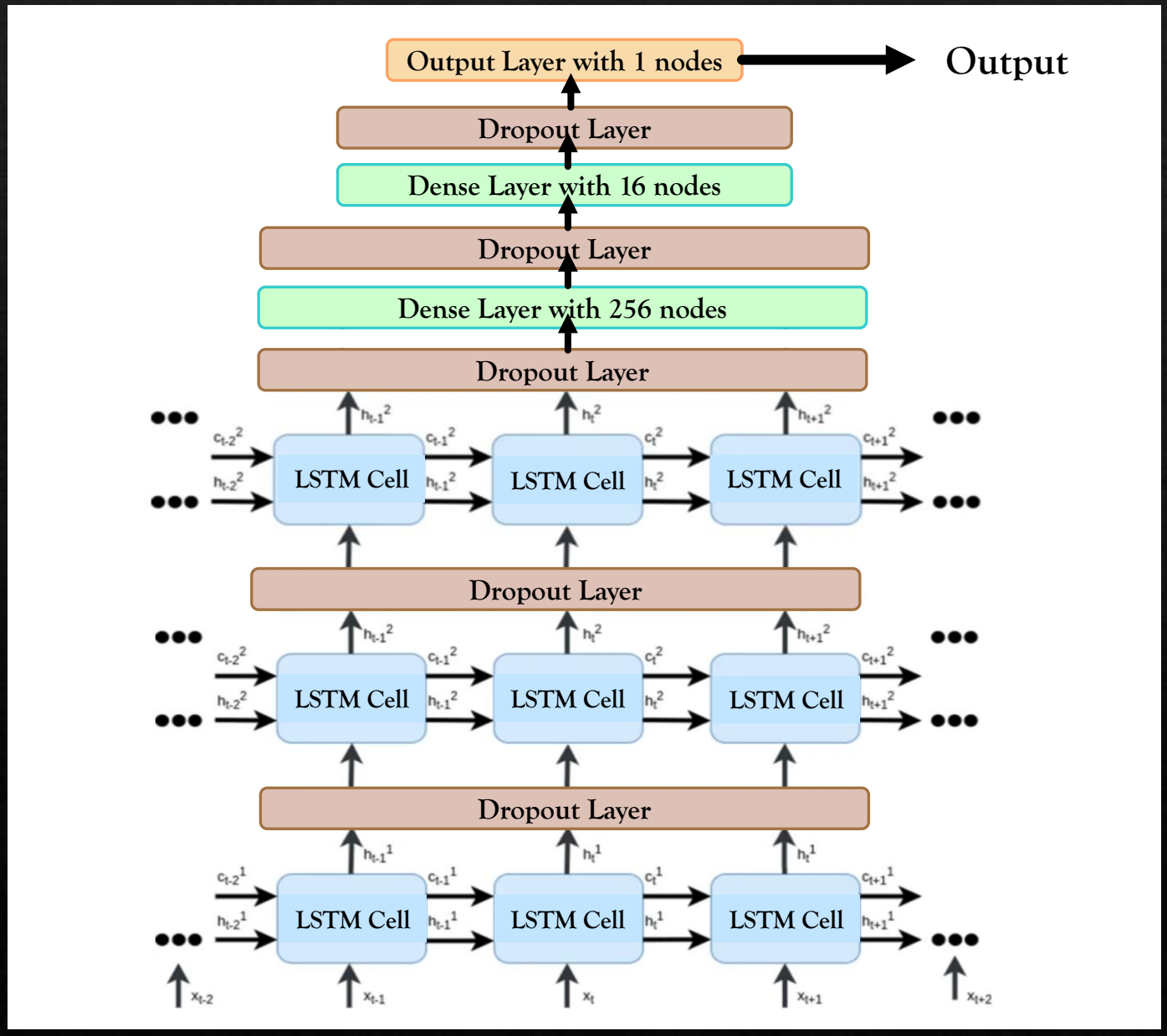


Figure 1 : One hidden layer MLP.



Deep Learning: Bidirectional LSTM based RNN

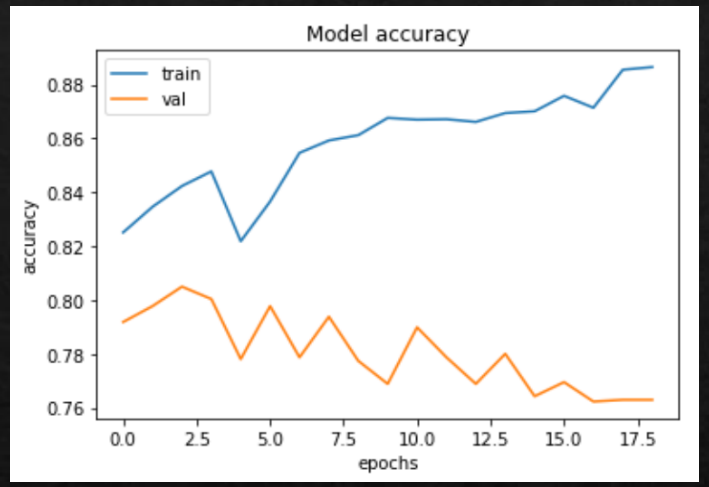


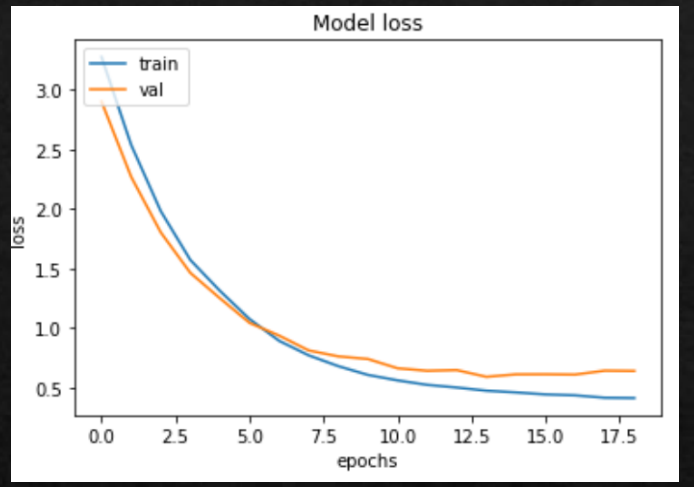

```
CPU times: user 14 µs, sys: 0 ns, total: 14 µs
Wall time: 28.6 µs
Model: "Bidirectional_RNN"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_7 (Embedding)      (None, 256, 256)          256000

 bidirectional (Bidirectiona  (None, 256, 256)          394240
 l)

 dropout (Dropout)            (None, 256, 256)          0

 bidirectional_1 (Bidirectio  (None, 256, 128)          164352
 nal)

 dropout_1 (Dropout)          (None, 256, 128)          0

 bidirectional_2 (Bidirectio  (None, 64)                41216
 nal)

 dropout_2 (Dropout)          (None, 64)                0

 dense_7 (Dense)              (None, 256)               16640

 dropout_3 (Dropout)          (None, 256)               0

 dense_8 (Dense)              (None, 16)                4112

 dropout_4 (Dropout)          (None, 16)                0

 dense_9 (Dense)              (None, 1)                 17

=================================================================
Total params: 876,577
Trainable params: 876,577
Non-trainable params: 0
```

Deep Learning: Bidirectional LSTM based RNN

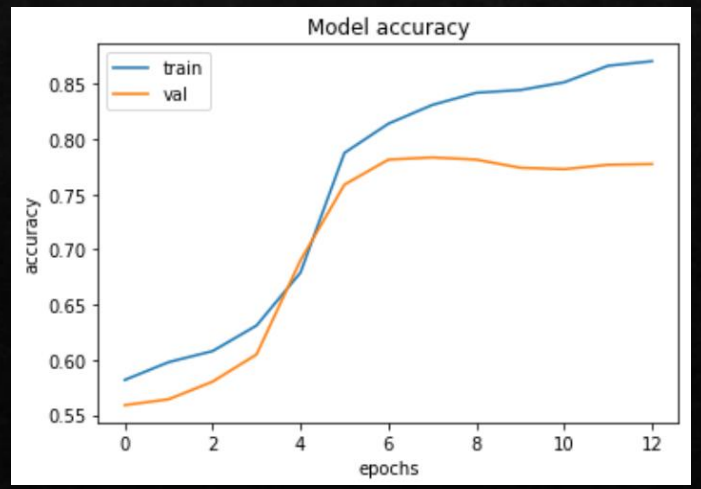
Kaggle F1 Scores

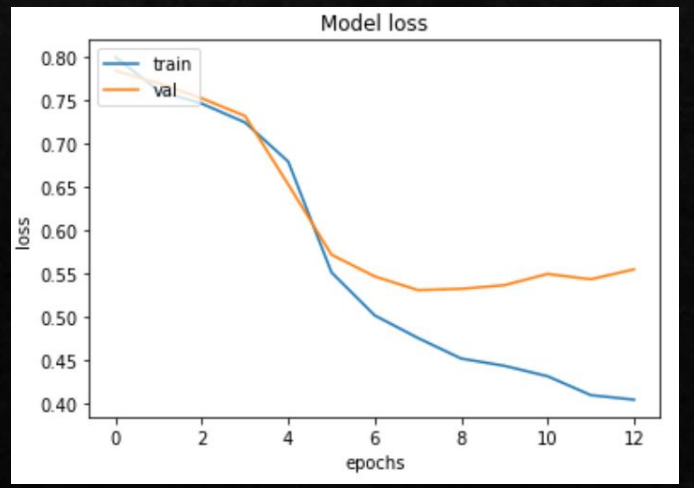
1 Trained on Unpreprocessed Train Dataset





2 Trained on Preprocessed Train Dataset





Transformer-Based Models

Timeline for Projects After BERT

List of Transformers:
https://github.com/huggingface/transformers

BERT
October 11, 2018 — Google
Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin et al

GPT-2
February 14, 2019 — OpenAI
Language Models are Unsupervised Multitask Learners

XLNet
June 19, 2019 — Google/CMU
Generalized Autoregressive Pre-training for Language Understanding

CTRL
September 11, 2019 — Salesforce
Conditional Transformer Language Model for Controllable Generation

DistilBERT
October 2, 2019 — HuggingFace
Smaller and Faster BERT using Model Distillation

Transformer-XL
Google/CMU — January 9, 2019
Attentive Language Models Beyond a Fixed-Length Context

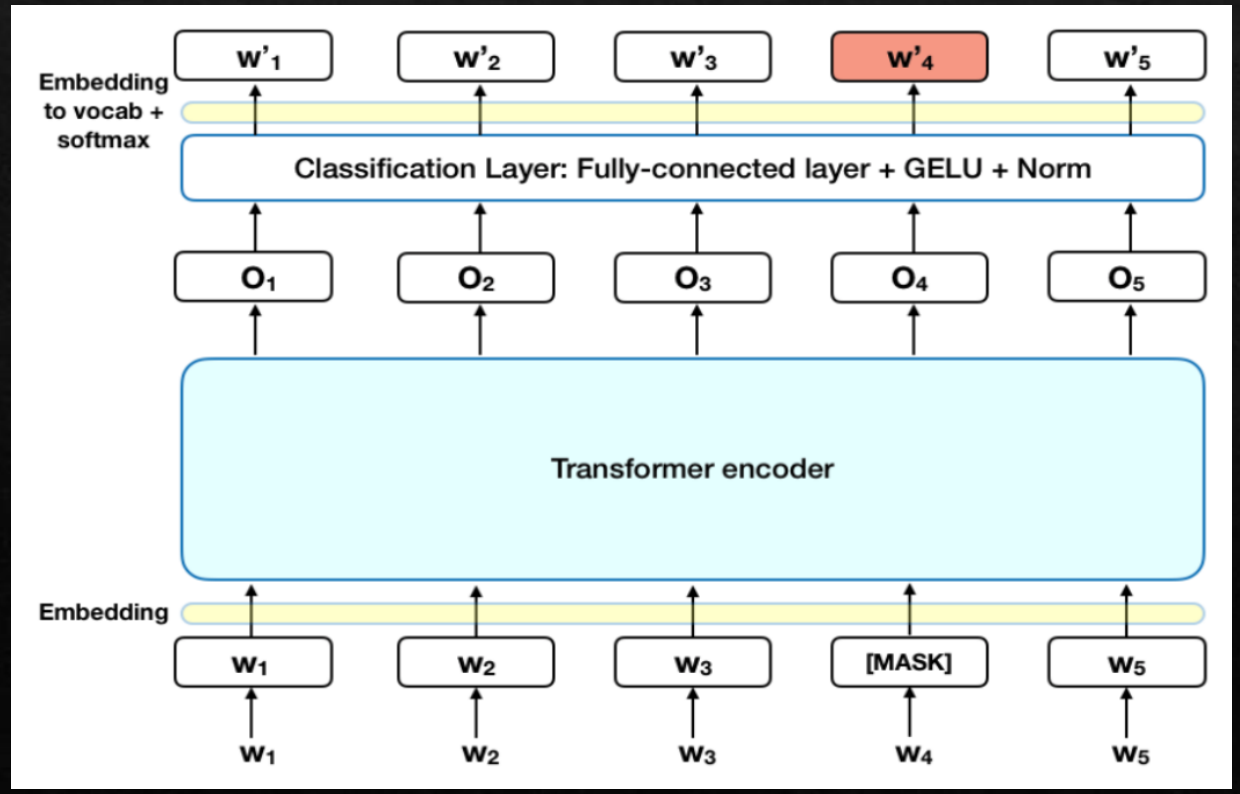
ERNIE
Baidu Inc. — April 19, 2019
Enhanced Representation through Knowledge Integration

RoBERTa
Facebook AI — July 26, 2019
Robustly Optimized BERT Pre-training Approach

ALBERT
Google Research & Toyota — September 26, 2019
Lite BERT for Self-supervised Learning of Language Representations

Transformer: BERT

Bidirectional Encoder Representations from Transformers (BERT): transformer-based architecture, 'designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers'

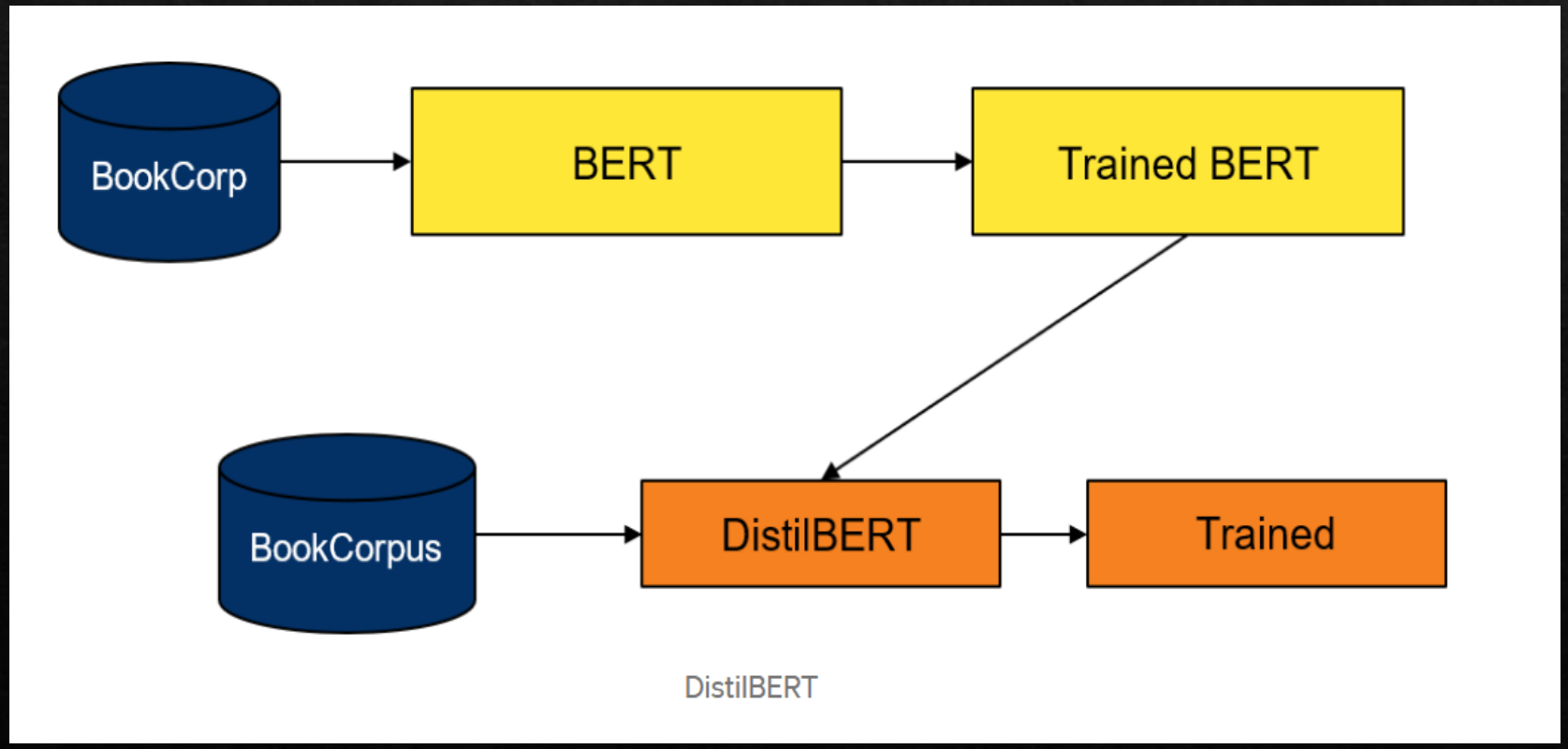


- Fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks
- Two different architectures: BERT_Base & BERT_Large
- GLUE Score (at GLUE Benchmark) 80.5%

Image Source: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Transformer: DistilBERT

BERT Distillation: smaller version of BERT developed and open-sourced by the team at HuggingFace

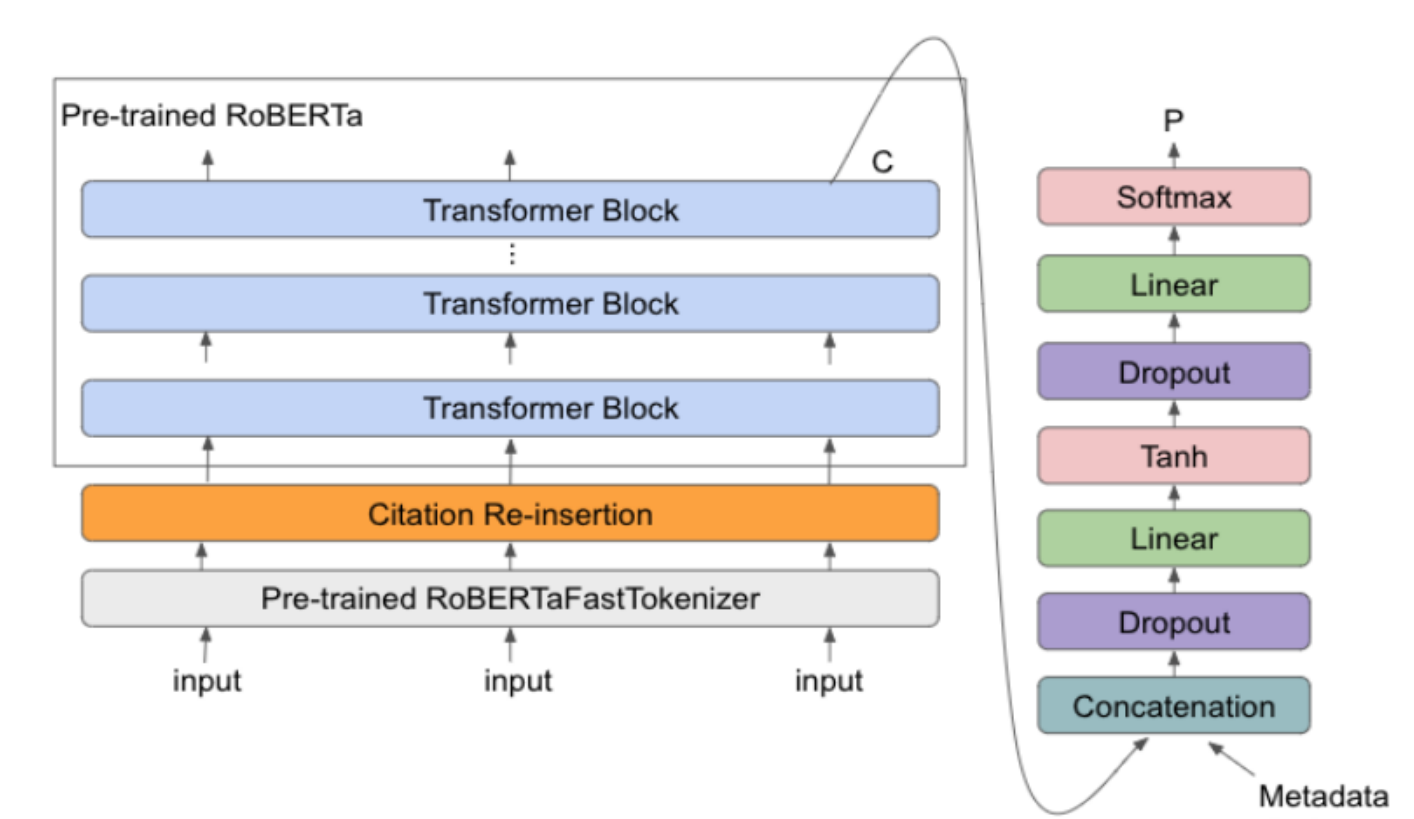


- 40% less parameters than BERT-base-uncased, runs 60% faster, preserving over 95% of BERT's performances
- Implementation on TensorFlow/Keras, Trained DistilBERT used to generate sentence embedding
- Basic NN Architecture (with Dense and Dropout layers)

Image Source: https://swatimeena989.medium.com/distilbert-text-classification-using-keras-c1201d3a3d9d

Transformer: RoBERTa

Robustly Optimized BERT Pretraining Approach (RoBERTa): developed by Facebook AI & Washington University. Improves on the BERT model by modifying key hyperparameters and pretraining on a larger corpus



The RoBERTa model architecture.

- **Modifications to BERT with the following changes:**
  - ➔ Removing the Next Sentence Prediction (NSP) objective
  - ➔ Training with bigger batch sizes & longer sequences
  - ➔ Dynamically changing the masking pattern
- **Datasets used to train:**
  - ➔ BOOK CORPUS and English Wikipedia dataset (16 GB)
  - ➔ CC-NEWS (76 GB)
  - ➔ OPENWEBTEXT (38 GB)
  - ➔ STORIES (31 GB)

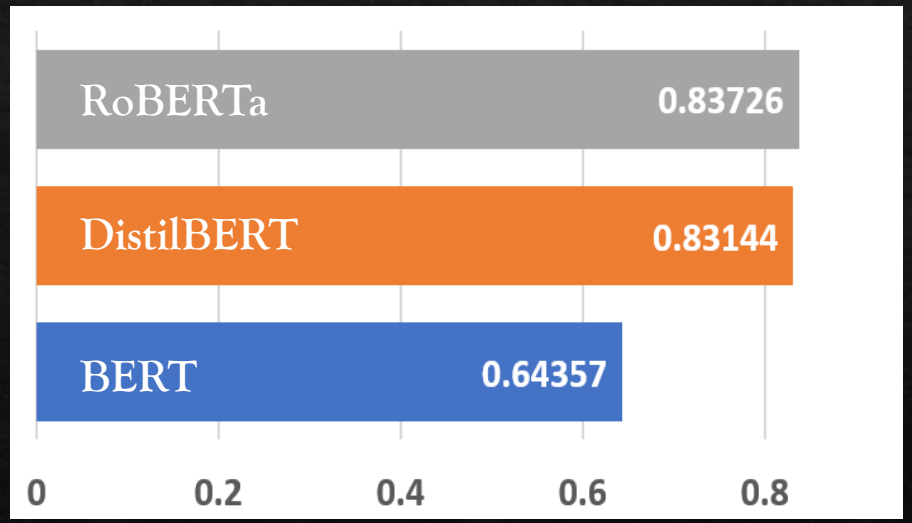
Image Source: Research Paper, "Context-Aware Legal Citation Recommendation using Deep Learning" by Huang et al, Krass et al & Grabmair et al, June 2021

Transformers: Comparison

	BERT	RoBERTa	DistilBERT
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base**: 66
Training Time	**Base**: 8 x V100 x 12 days* **Large**: 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large**: 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation

Image Source: https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8

Kaggle F1 Scores



Easy deployment of the model for a variety of NLP tasks with a few lines of code through the transformers library by Hugging Face (https://huggingface.co/docs/transformers/index)

kaggle

Discussion | Leaderboard | Rules | Team | My Submissions | Submit Predictions | ...

#	Team	Members	Score	Entries	Last	Code	Join
1	Andrej Marinchenko		1.00000	270	17h		
2	Moshi Wei		1.00000	1	2mo		
3	Jewel Liu		1.00000	2	2mo		
4	DinoKing		1.00000	1	2mo		
5	Wonhyeong Seo		1.00000	1	1mo		
6	💥Tesla, Inc.💥		1.00000	1	1mo		
52	nicsli		0.83726	9	1mo		
53	**Nicole Hong**		0.83726	2	23d		
54	rookie #2		0.83665	5	7d		

Total 718 Teams

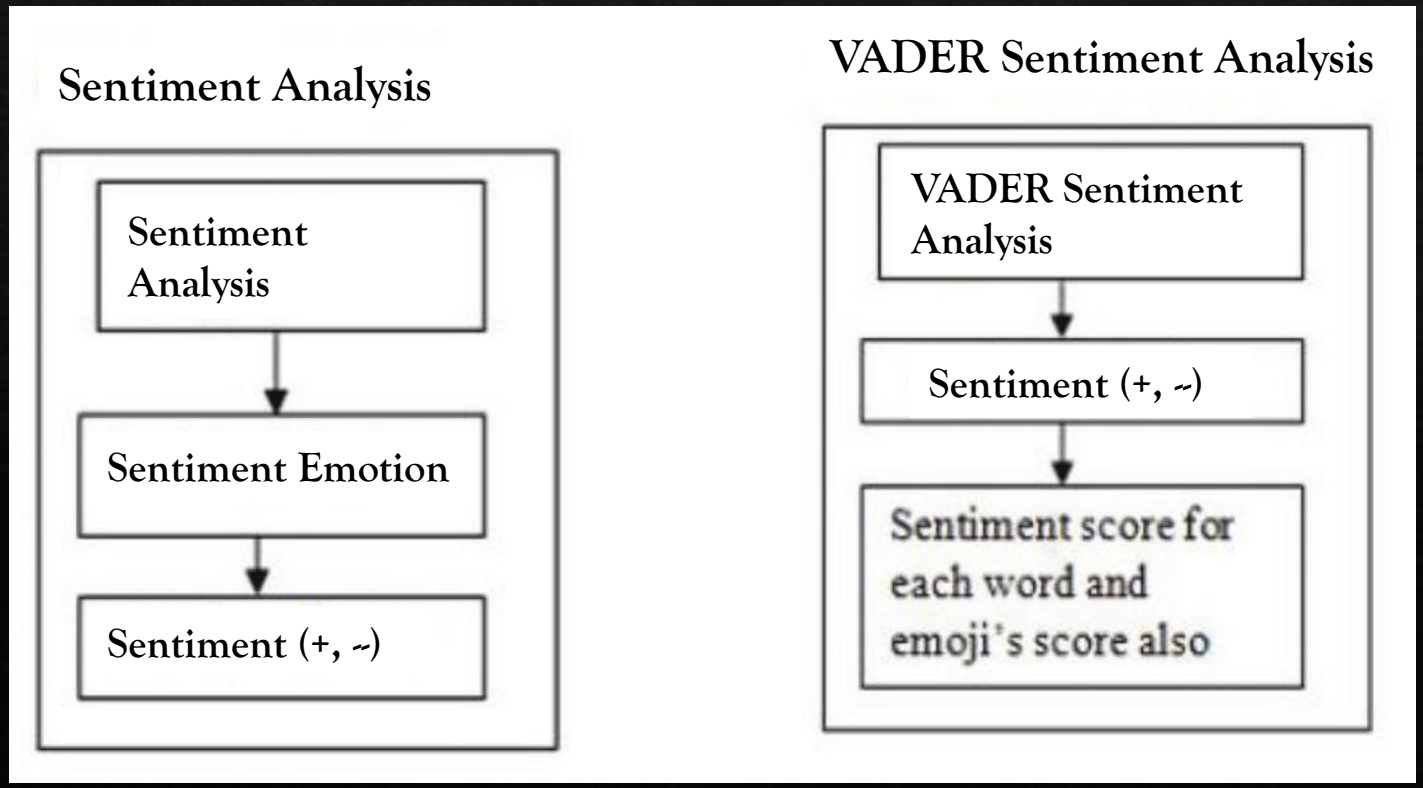
Me!

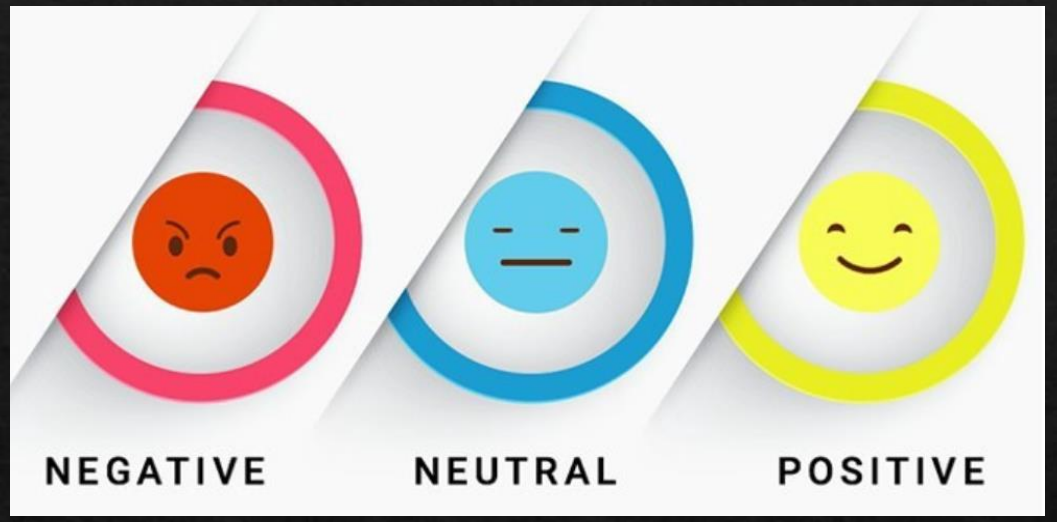
Your Best Entry!
Your most recent submission scored 0.83726, which is an improvement of your previous score of 0.83144. Great job!

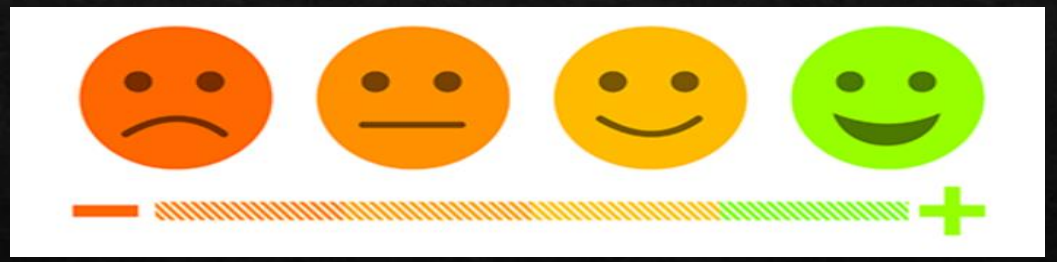
Tweet this

VADER Sentiment Analysis

Valence Aware Dictionary and sEntiment Reasoner (VADER): lexicon and rule-based sentiment analysis tool, specifically attuned to sentiments expressed in social media



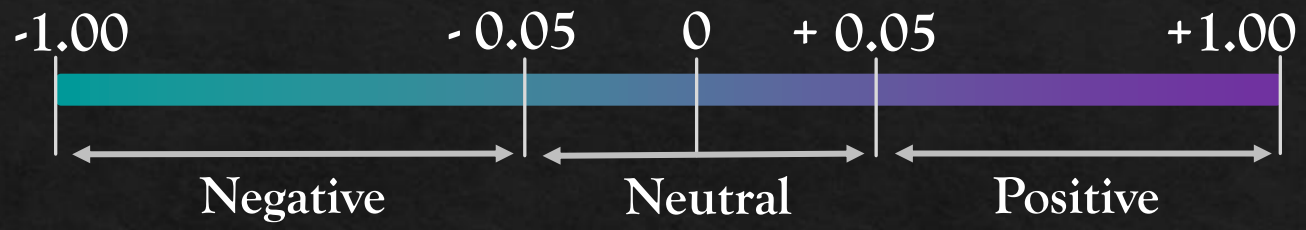




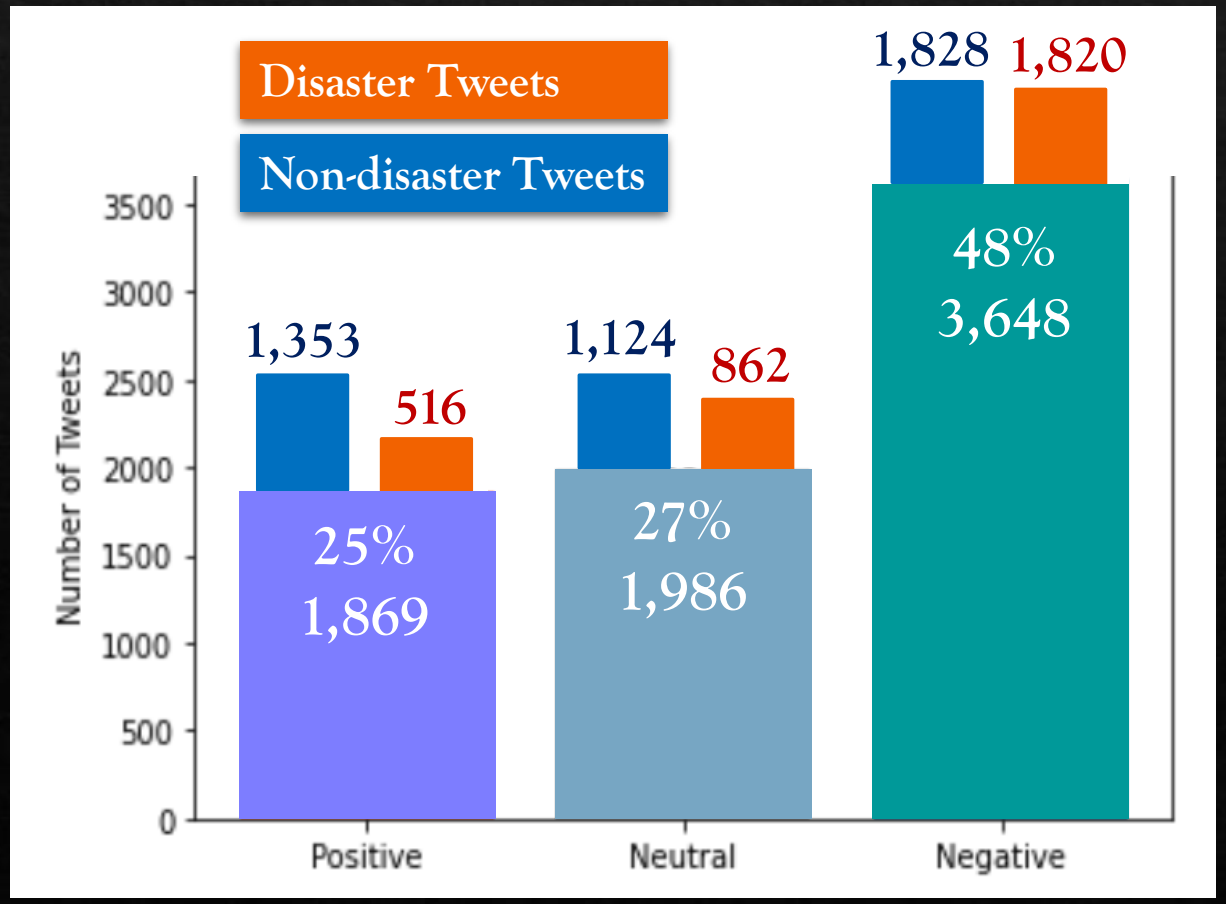
- Uses a combination of A sentiment lexicon in a list of lexical features
- Provides Positive, Negative, Neutral Scores of the sentence

Image Source: https://medium.com/@urvisoni/vader-a-parsimonious-rule-based-model-for-sentiment-analysis-of-social-media-text-34380204b96f

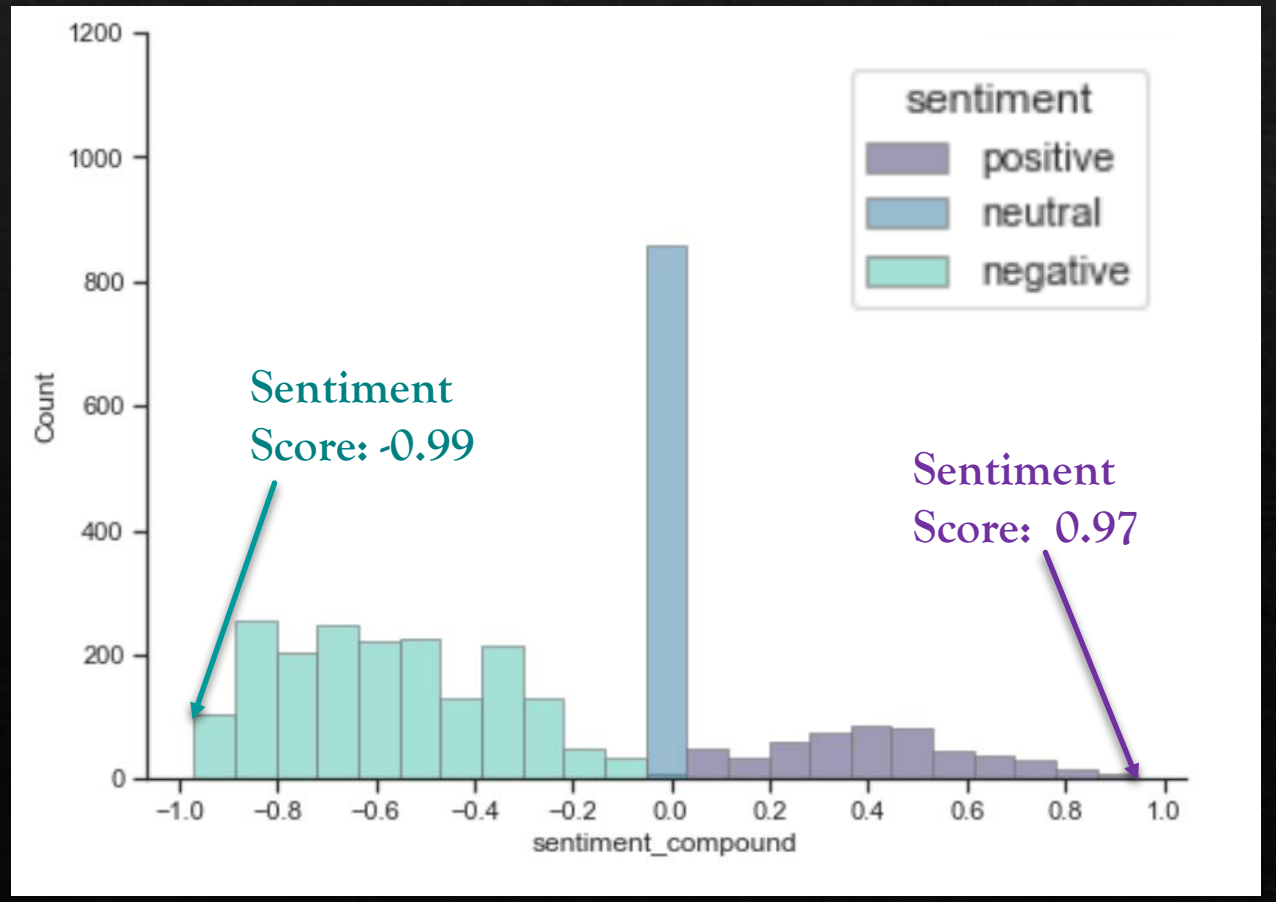
Sentiment Analysis: Result Summary



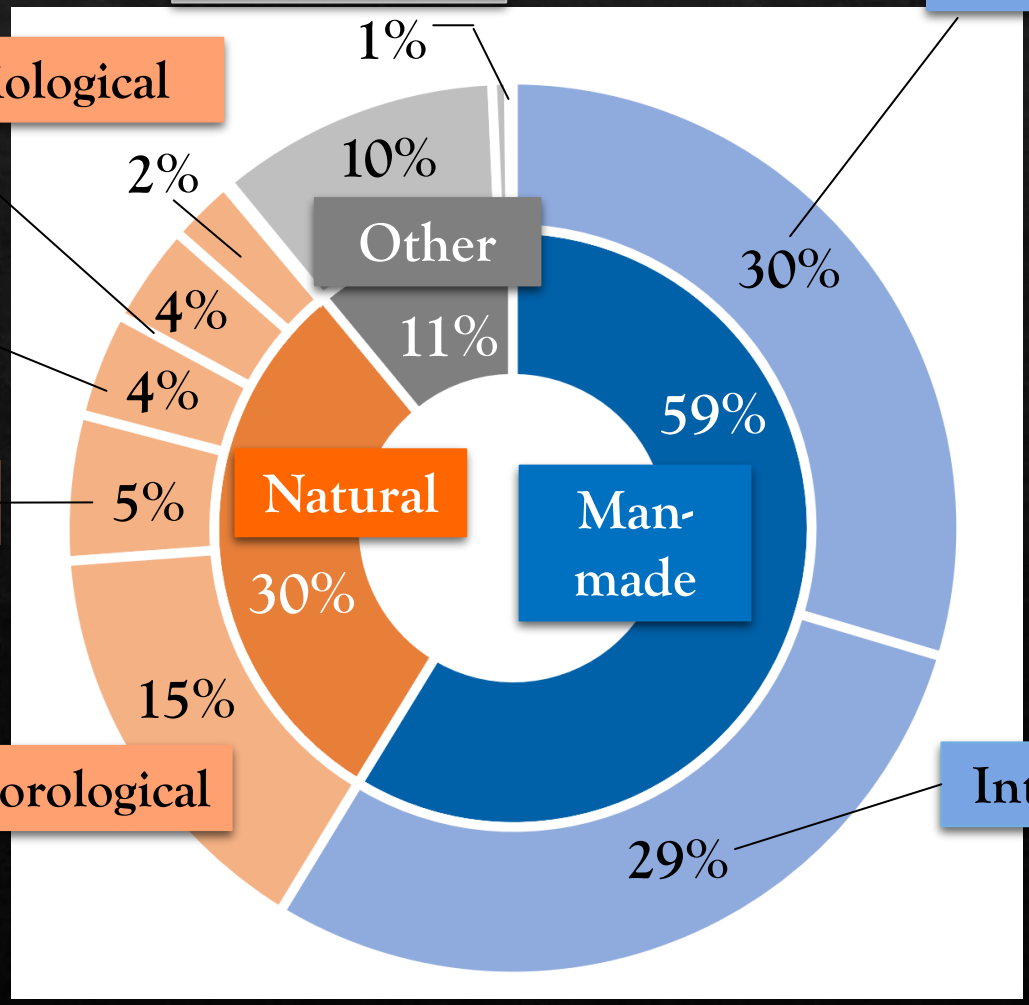
Counts of Sentiment Classes in All Tweets



Sentiment Score Distribution – Disaster Tweets



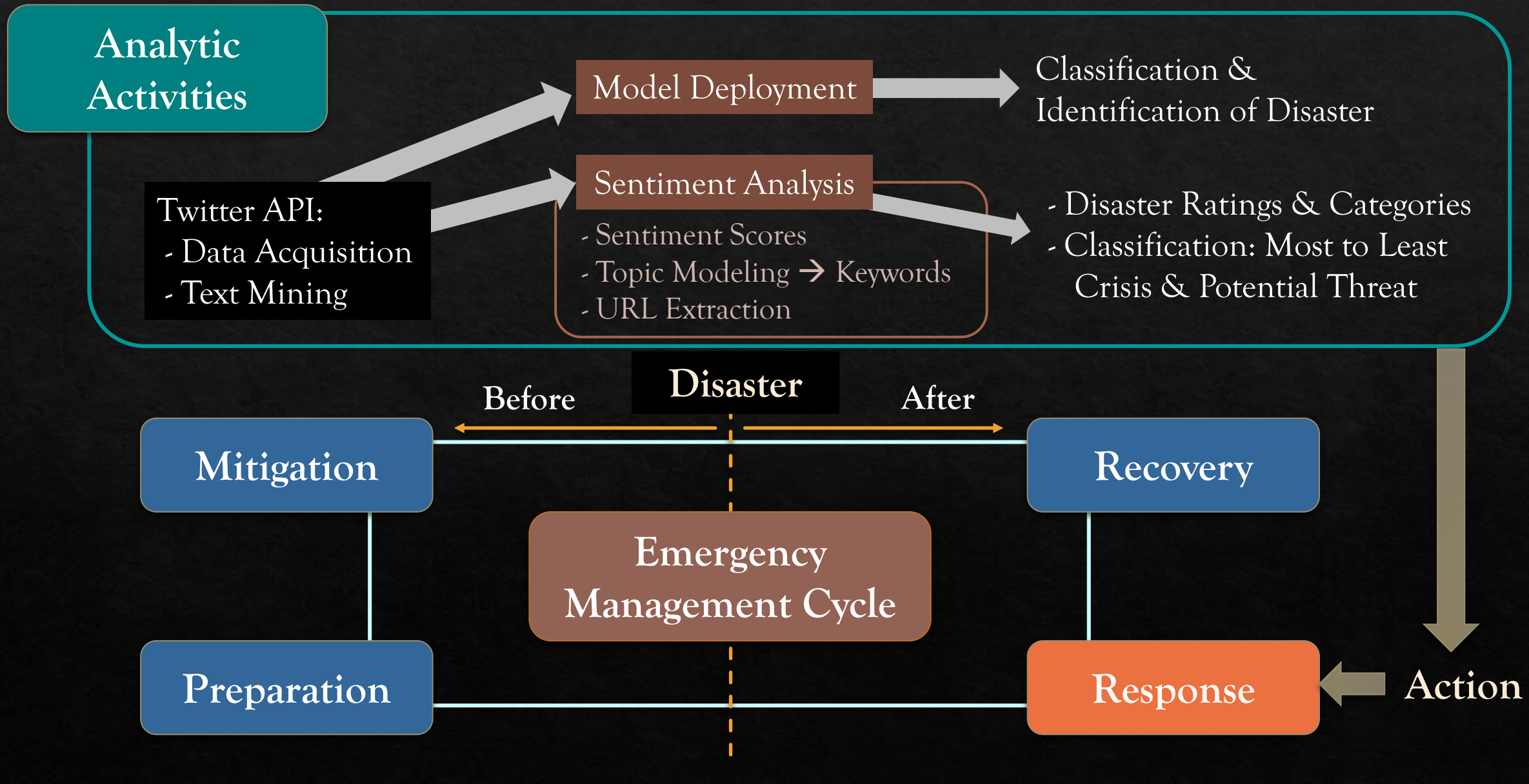
Disaster Tweets

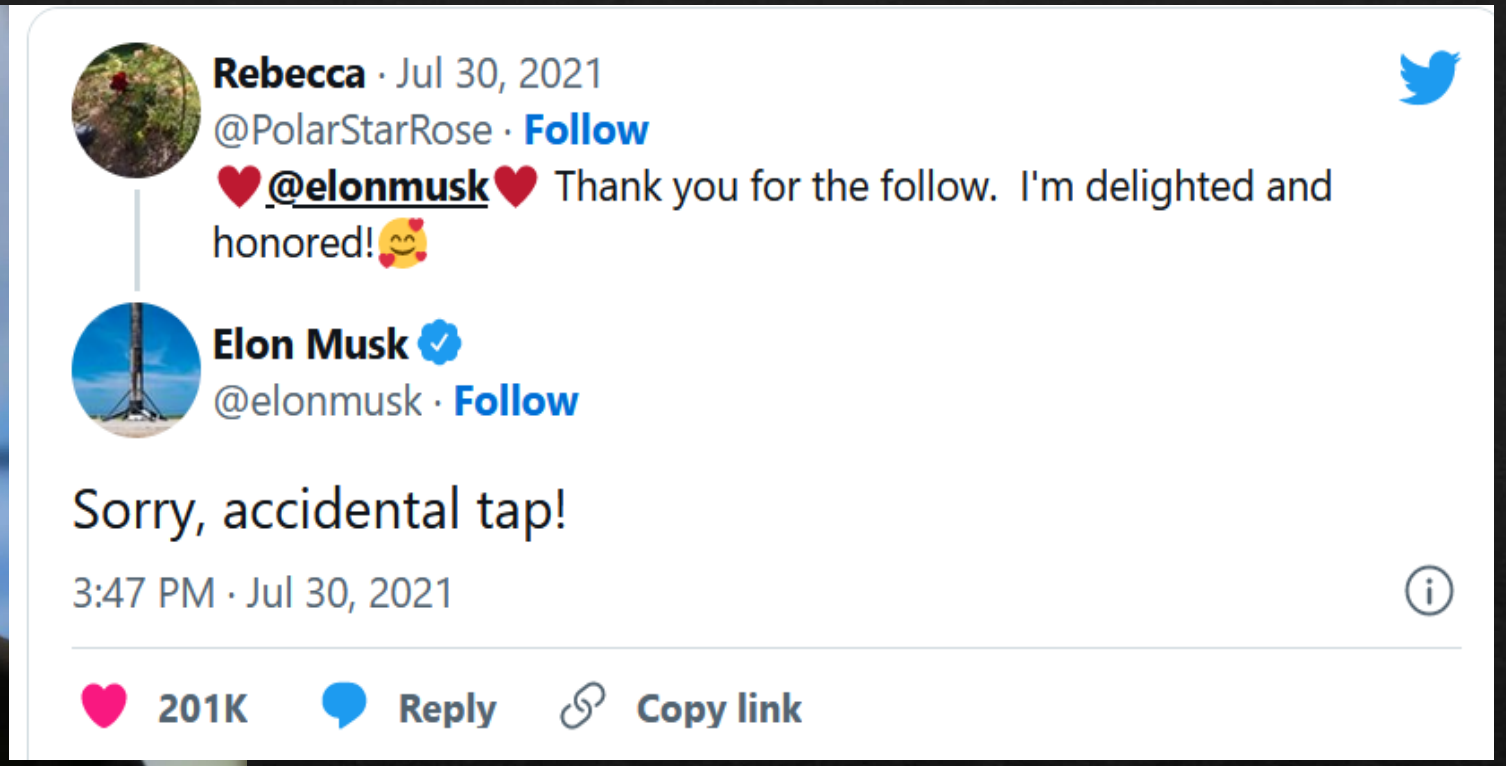


Disaster Class & Categories

Class	Sub-class
Natural	Meteorological Hydrological Geophysical Climatological Biological
Man-made	Intentional Accidental
Other	Non-disaster related

Emergency Response Management





n.hong@queensu.ca
www.linkedin.com/in/nicolehhong
https://github.com/Nicole-Hong

Thank You!