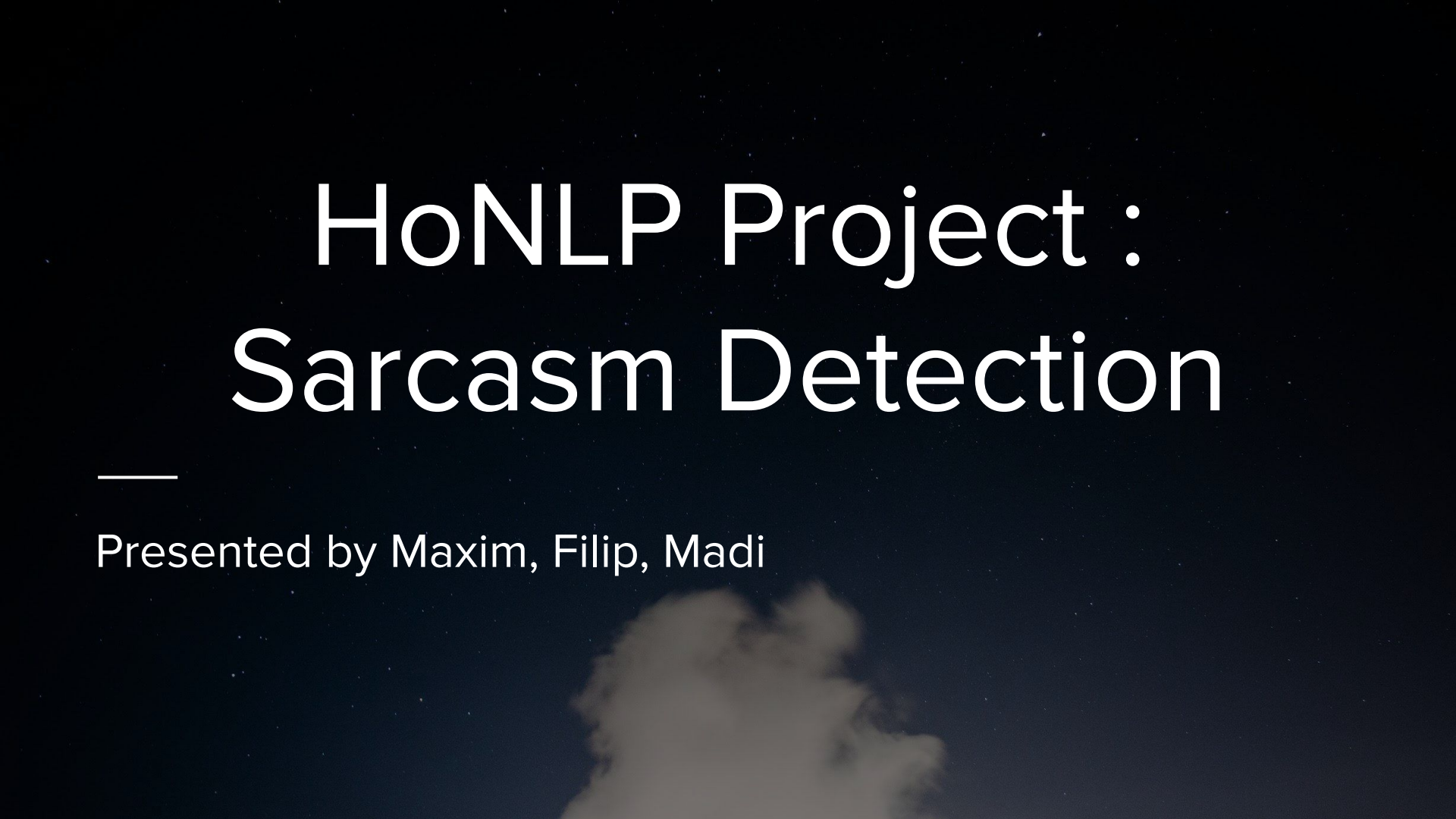
HoNLP Project : Sarcasm Detection

Presented by Maxim, Filip, Madi



Why we chose this topic?

Ex: "I just love it when my computer crashes right in the middle of an important project presentation. It's such a great way to take a break and destress!"

The challenge

Data

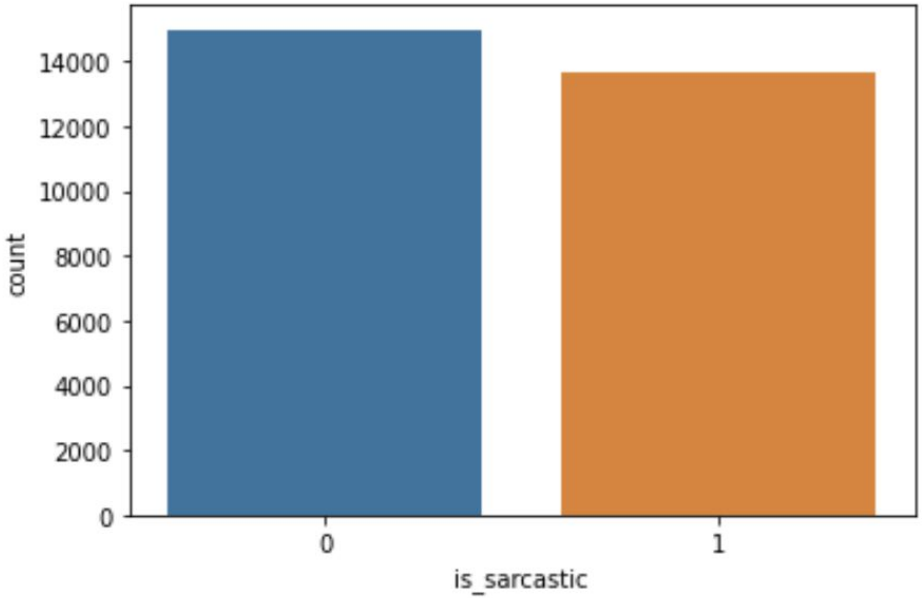
Each record consists of three attributes:

- **is_sarcastic**: 1 if the record is sarcastic otherwise 0
- **headline**: the headline of the news article
- **article_link**: link to the original news article. Useful for collecting supplementary data

Data

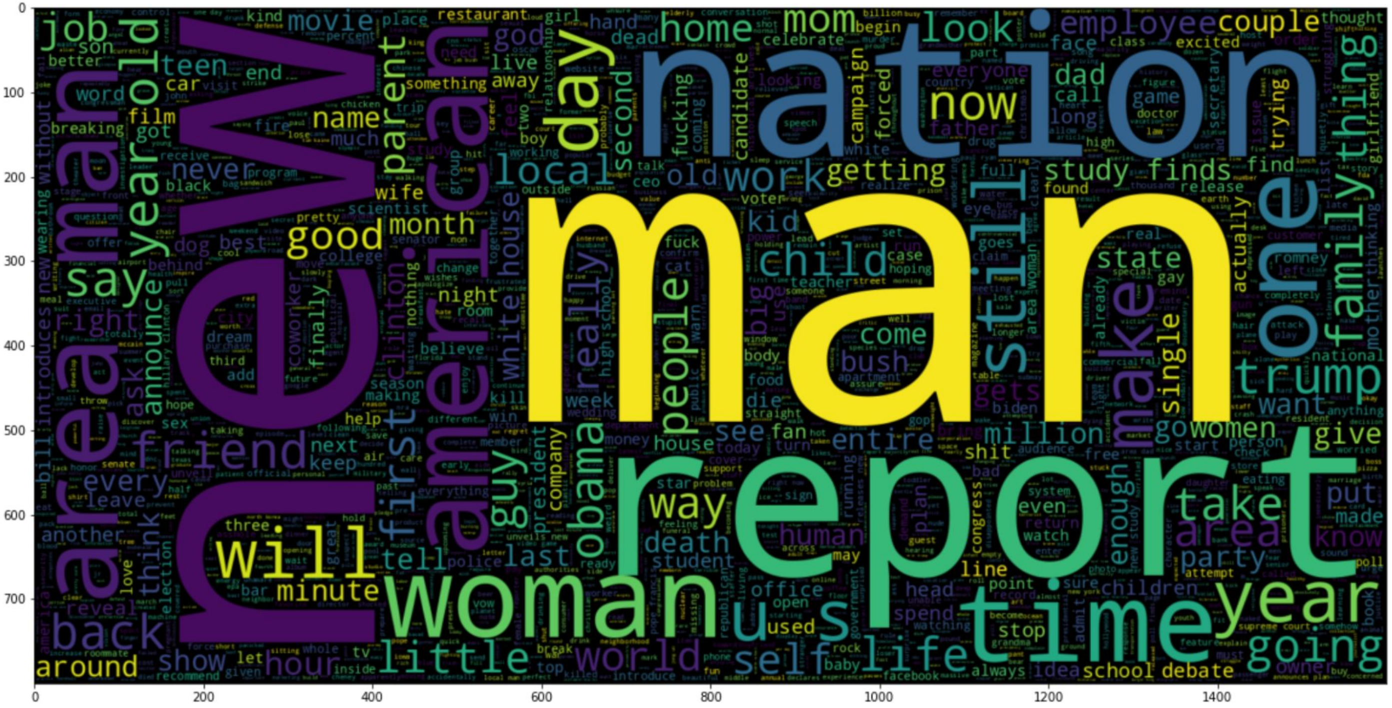
	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snape happy birthday in th...	0

Data Distribution



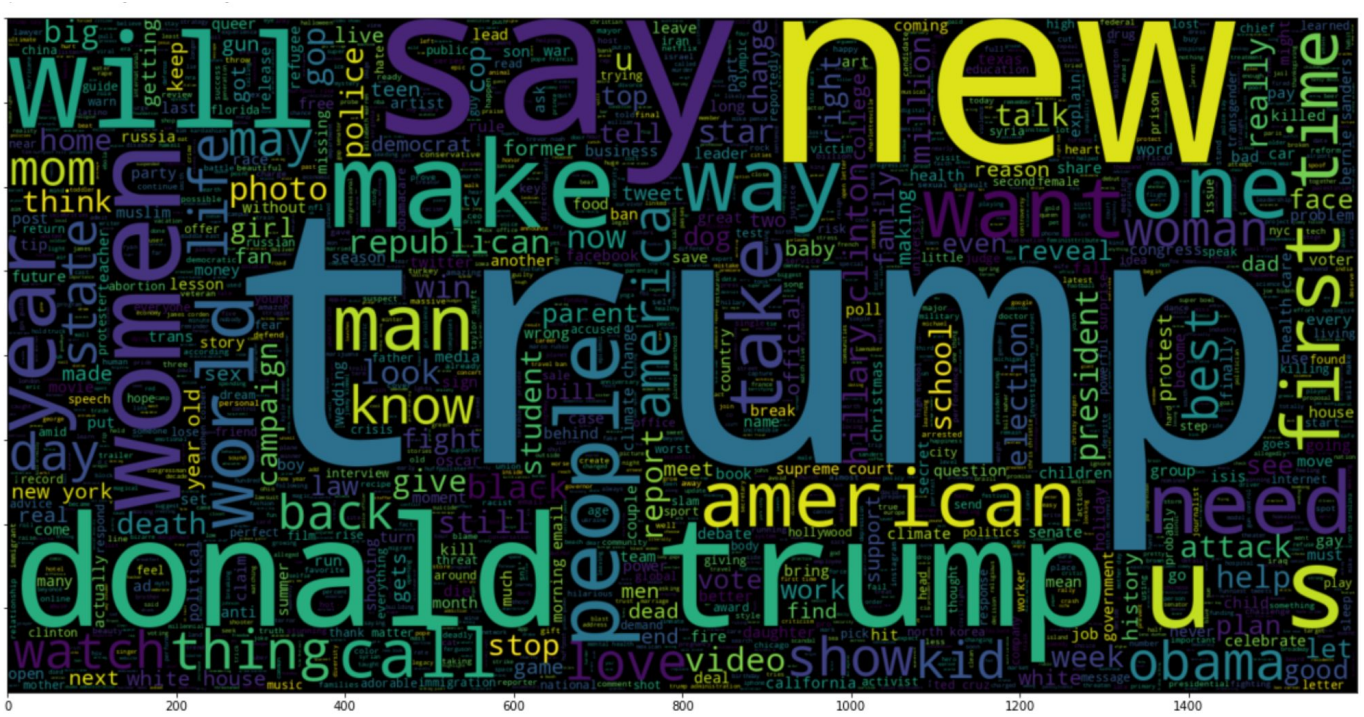
Data Visualization

Word-cloud for text that is sarcastic

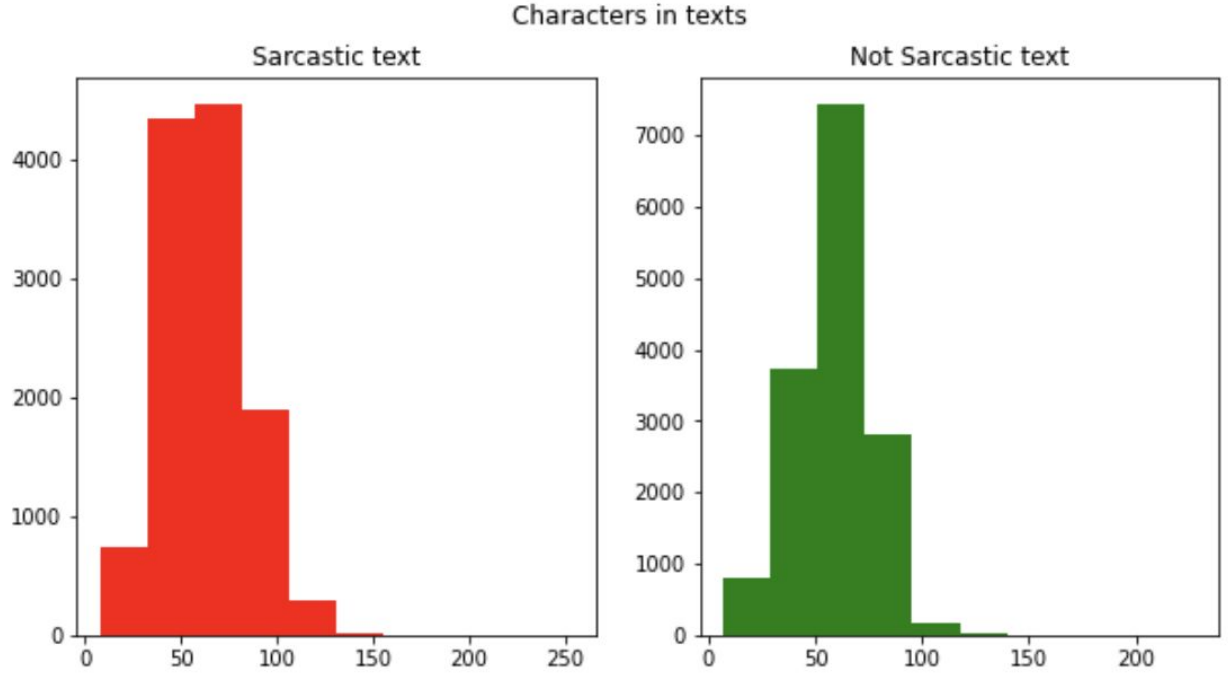


Data Visualization

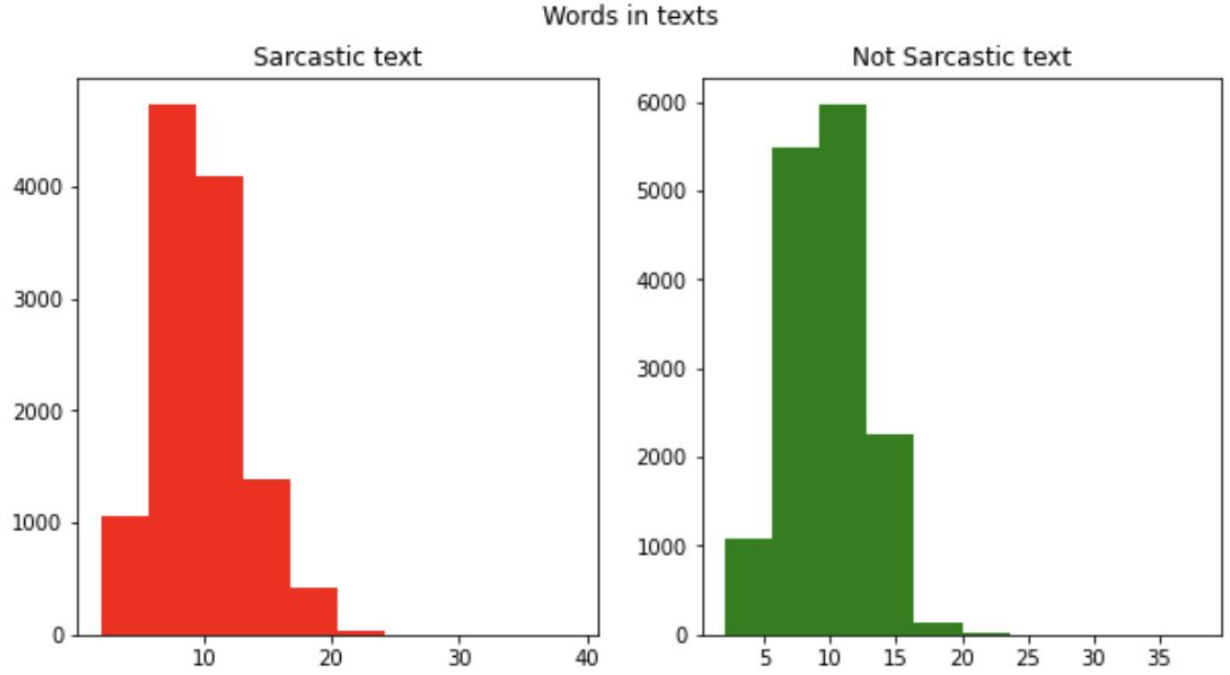
Word-cloud for text that is NOT sarcastic



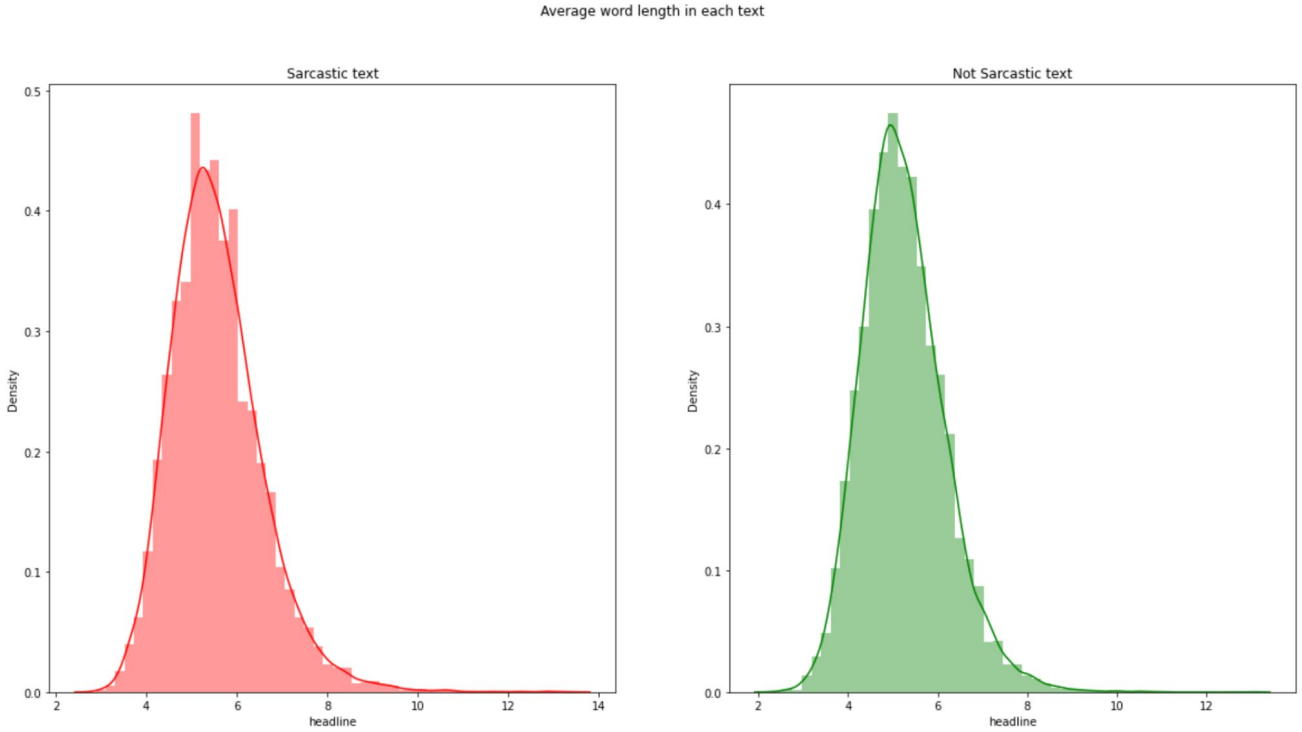
Data Visualization



Data Visualization



Data Visualization



Pre-processing

```
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

def remove_url(text):
    return re.sub('http\S+', '', text)

def remove_stopwords(text):
    final_text = []
    for i in text.split():
        if (i.strip().lower() not in stop):
            final_text.append(i.strip())
    return " ".join(final_text)

def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    text = remove_stopwords(text)
    return text
```

```
def remove_punctuation(text):
    return re.sub(r'[^\w\s]', '', text)
```

CountVectorizer

```python
def algos_test_CountVectorizer(train_xs, test_xs, train_ys, test_ys):
    vectorizer = CountVectorizer()
    train_xs = vectorizer.fit_transform(train_xs)
    test_xs = vectorizer.transform(test_xs)

    algos = [
        LogisticRegression(),
        MultinomialNB(),
        DecisionTreeClassifier(),
        RandomForestClassifier(),
        SVC(),
        xgb.XGBClassifier()
    ]
    results = []
    for algo in algos:
        print(algo)
        %timeit -n 1 -r 1 algo.fit(train_xs, train_ys)
        pred_ys = algo.predict(test_xs)
        results.append(
            {
                "algo": algo,
                "accuracy": accuracy_score(test_ys, pred_ys),
                "precision": score(test_ys, pred_ys, average="macro")[0],
                "recall": score(test_ys, pred_ys, average="macro")[1],
                "f1": score(test_ys, pred_ys, average="macro")[2],
            }
        )
    return pd.DataFrame.from_records(results)

results_CountVectorizer = algos_test_CountVectorizer(x_train, x_test, y_train, y_test)
```

CountVectorizer

	algo	accuracy	precision	recall	f1
0	LogisticRegression()	0.794584	0.795300	0.785976	0.788773
1	MultinomialNB()	0.803819	0.802650	0.797662	0.799508
2	DecisionTreeClassifier()	0.727942	0.724009	0.723677	0.723837
3	(DecisionTreeClassifier(max_features='sqrt', r...	0.754773	0.767401	0.736998	0.740165
4	SVC()	0.791089	0.803113	0.775958	0.780513
5	XGBClassifier(base_score=None, booster=None, c...	0.722326	0.756285	0.695473	0.693120

TfidfVectorizer

```
def algos_test_TfidVectorizer(train_xs, test_xs, train_ys, test_ys):
    vectorizer = TfidfVectorizer()
    train_xs = vectorizer.fit_transform(train_xs)
    test_xs = vectorizer.transform(test_xs)

    algos = [
        LogisticRegression(),
        MultinomialNB(),
        DecisionTreeClassifier(),
        RandomForestClassifier(),
        SVC(),
        xgb.XGBClassifier()
    ]
    results = []
    for algo in algos:
        print(algo)
        %timeit -n 1 -r 1 algo.fit(train_xs, train_ys)
        pred_ys = algo.predict(test_xs)
        results.append(
            {
                "algo": algo,
                "accuracy": accuracy_score(test_ys, pred_ys),
                "precision": score(test_ys, pred_ys, average="macro")[0],
                "recall": score(test_ys, pred_ys, average="macro")[1],
                "f1": score(test_ys, pred_ys, average="macro")[2],
            }
        )
    return pd.DataFrame.from_records(results)

results_TfidVectorizer = algos_test_TfidVectorizer(x_train, x_test, y_train, y_test)
```

TfidfVectorizer

	algo	accuracy	precision	recall	f1
0	LogisticRegression()	0.790715	0.794471	0.779658	0.783240
1	MultinomialNB()	0.792088	0.806503	0.776097	0.780818
2	DecisionTreeClassifier()	0.717210	0.713200	0.710382	0.711399
3	(DecisionTreeClassifier(max_features='sqrt', r...	0.758767	0.763998	0.744812	0.748086
4	SVC()	0.796331	0.800844	0.785128	0.788904
5	XGBClassifier(base_score=None, booster=None, c...	0.716461	0.749430	0.689298	0.686236

Word embedding

Capturing context of words, semantic, syntactic similarity

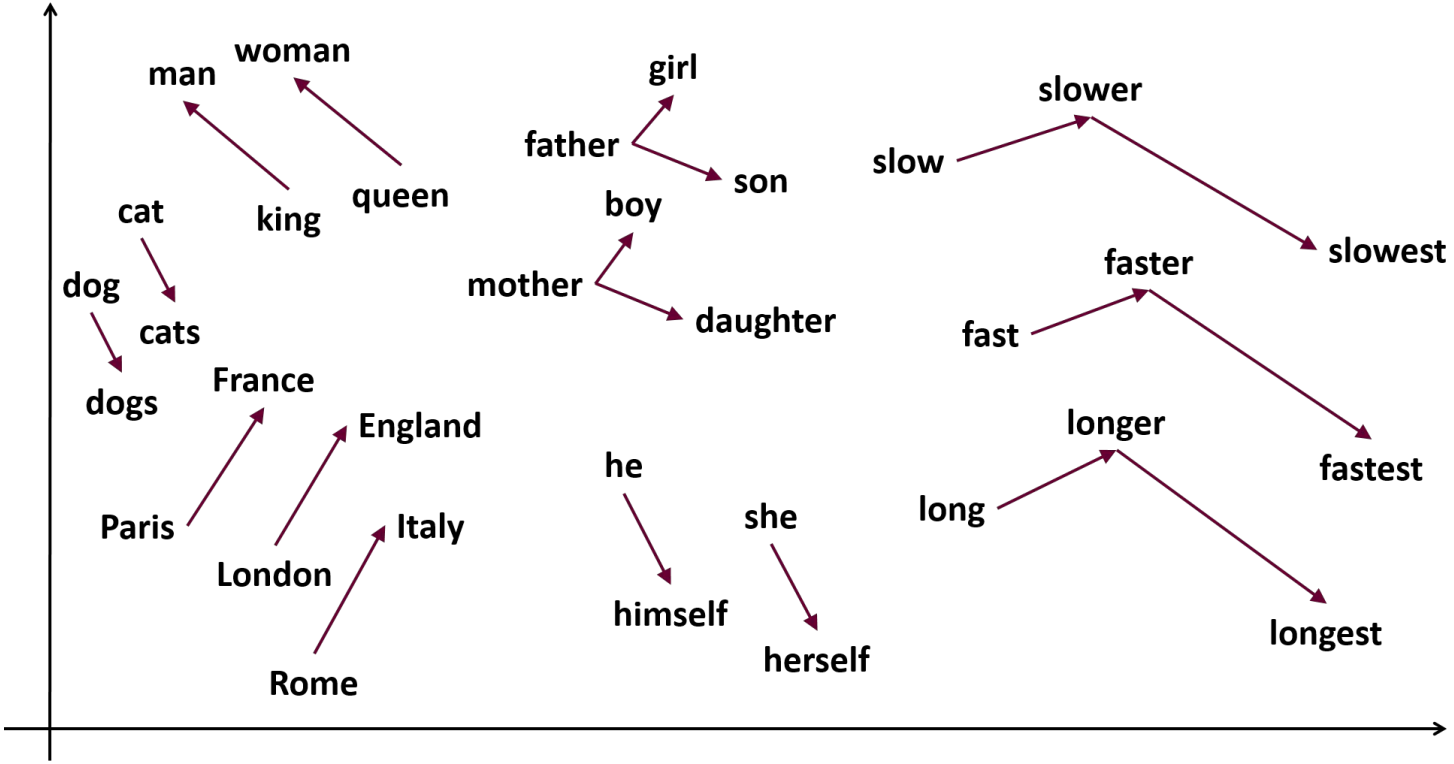
and relation with other words !

Cows lose their jobs as milk prices drop **is sarcastic**

Sheeps lose their jobs as wool prices drop **should be sarcastic too !**

Word2Vec

How ? Cosine Similarity between vectors !
cos(O) close to 1 if angle close to 0



Word2Vec

Put it in the correct format for gensim

```
words = []
for i in df.headline.values:
    words.append(i.split())
words[:5]
```

```
[['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'secret',
  'black',
  'code',
  'minority',
  'shoppers'],
 ['roseanne',
  'revival',
  'catches',
  'thorny',
  'political',
  'mood',
  'better',
  'worse'],
```

Word2Vec

Create a vector for each words

```
import gensim
#Dimension of vectors we are generating
EMBEDDING_DIM = 200

#Creating Word Vectors by Word2Vec Method (takes time...)
w2v_model = gensim.models.Word2Vec(sentences = words , vector_size=EMBEDDING_DIM , window = 5 , min_count = 1)
```

```
#vocab size
len(w2v_model.wv)
#We have now represented each of 28359 words by a 200dim vector.
```

```
28359
```

Word2Vec

Create the embedding matrix

```
# Function to create weight matrix from word2vec gensim model
def get_weight_matrix(model, vocab):
    # total vocabulary size plus 0 for unknown words
    vocab_size = len(vocab) + 1
    # define weight matrix dimensions with all 0
    weight_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
    # step vocab, store vectors using the Tokenizer's integer mapping
    for word, i in vocab.items():
        weight_matrix[i] = model[word]
    return weight_matrix
```

Word2Vec

Create the model : Recurrent Neural Network (RNN)

```
#Defining Neural Network
model = Sequential()
#Non-trainable embeddidng layer
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM, weights=[embedding_vectors], input_length=20, trainable=True))
#LSTM
model.add(Bidirectional(LSTM(units=128 , recurrent_dropout = 0.3 , dropout = 0.3,return_sequences = True)))
model.add(Bidirectional(GRU(units=32 , recurrent_dropout = 0.1 , dropout = 0.1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.01), loss='binary_crossentropy', metrics=['acc'])
```

Trainable embedding matrix
LSTM : long-short-term memory
GRU : gated recurrent unit
Activation layer

} solve the exploding and vanishing gradient problem

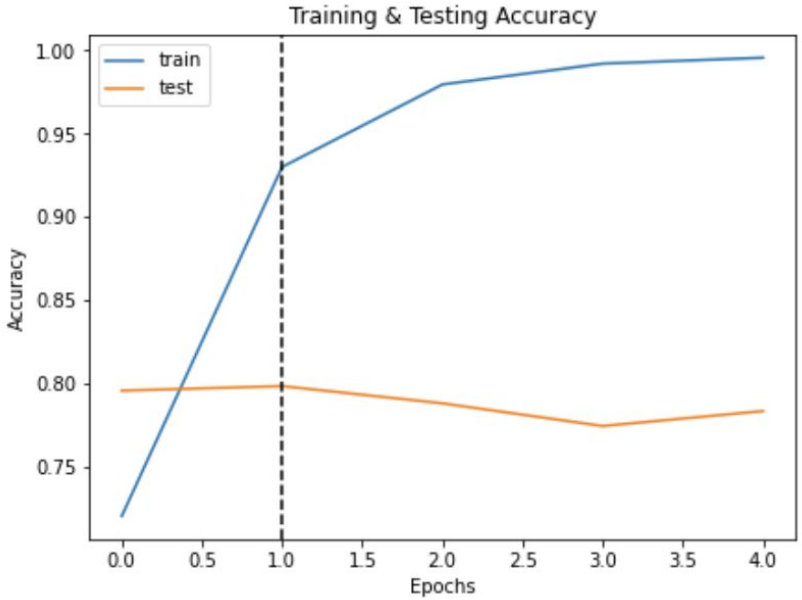
Word2Vec

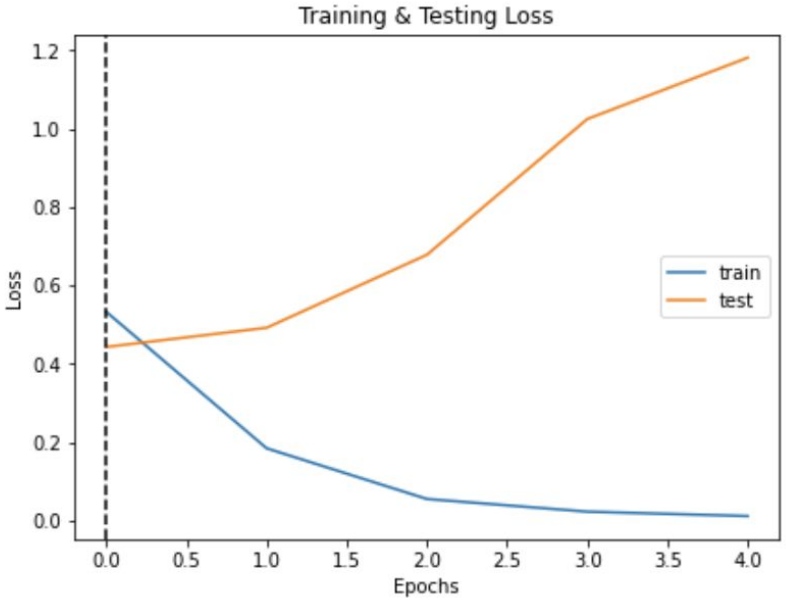
Tokenizing and padding before train/test split

```
tokenizer = text.Tokenizer(num_words=35000)
tokenizer.fit_on_texts(words)
tokenized_train = tokenizer.texts_to_sequences(words)
x = pad_sequences(tokenized_train, maxlen = 20)
```

Word2Vec

Results





Word2Vec

	precision	recall	f1-score	support
Not Sarcastic	0.79	0.83	0.81	4483
Sarcastic	0.77	0.73	0.75	3530
accuracy			0.78	8013
macro avg	0.78	0.78	0.78	8013
weighted avg	0.78	0.78	0.78	8013

Glove

derive semantic relationships between words from the co-occurrence matrix

The cat sat on the mat

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

How ? Global statistics !

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

(>1) or close to 1 or (<1)

Glove

Create the embedding matrix : Transfer learning !

```
EMBEDDING_FILE = '../input/glove-twitter/glove.twitter.27B.200d.txt'
```

```
def get_coefs(word, *arr):
    return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.rstrip().rsplit(' ')) for o in open(EMBEDDING_FILE))
```

```
word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
#change below line if computing normal stats is too slow
embedding_matrix = embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

Glove

Create the model : Recurrent Neural Network (RNN)

```
#Defining Neural Network
model = Sequential()
#Non-trainable embeddidng layer
model.add(Embedding(nb_words, output_dim=embed_size, weights=[embedding_matrix], input_length=200, t
rainable=True))
#LSTM
model.add(Bidirectional(LSTM(units=128 , recurrent_dropout = 0.5 , dropout = 0.5)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=keras.optimizers.Adam(lr = 0.01), loss='binary_crossentropy', metrics=['ac
c'])
```

Trainable embedding matrix
LSTM : long-short-term memory
Activation layer

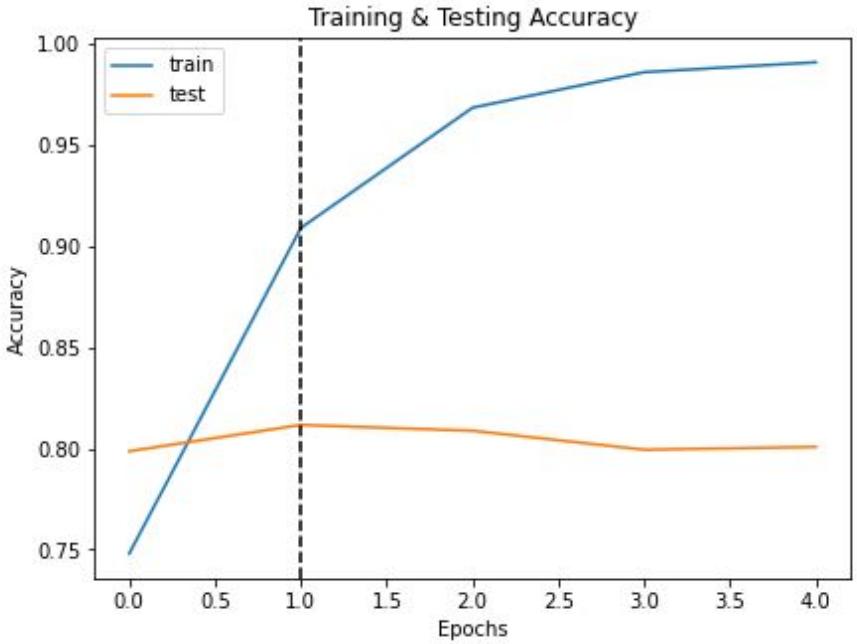
Glove

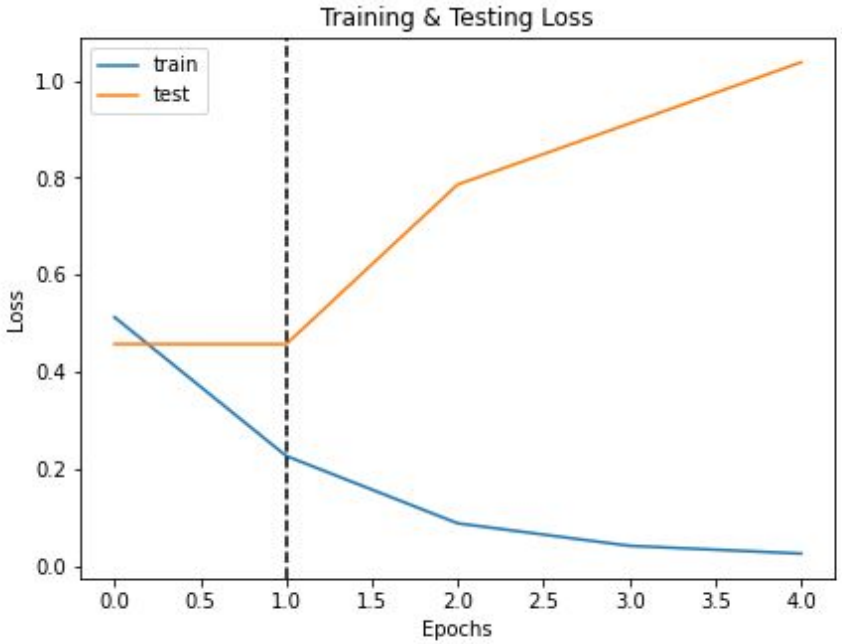
Tokenizing and padding before train/test split

```
tokenizer = text.Tokenizer(num_words=35000)
tokenizer.fit_on_texts(words)
tokenized_train = tokenizer.texts_to_sequences(words)
x = pad_sequences(tokenized_train, maxlen = 20)
```

Glove

Results



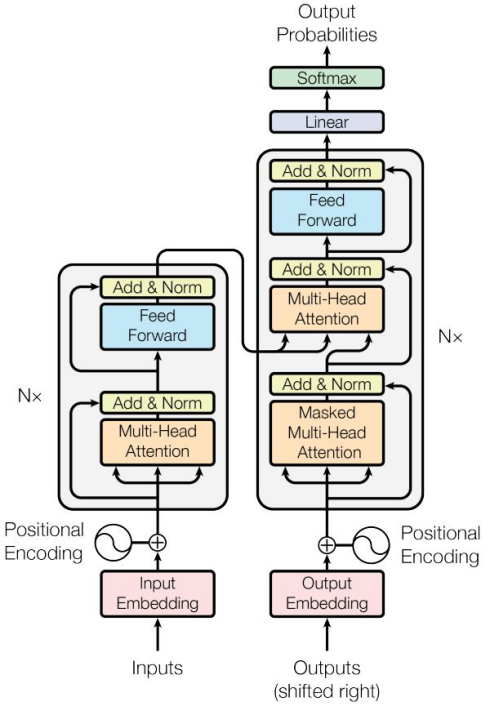


Glove

	precision	recall	f1-score	support
Not Sarcastic	0.79	0.87	0.83	4483
Sarcastic	0.82	0.71	0.76	3530
accuracy			0.80	8013
macro avg	0.80	0.79	0.79	8013
weighted avg	0.80	0.80	0.80	8013

Transformers

- Attention mechanism
- Positional encoding
- Pre-training
- Scalability



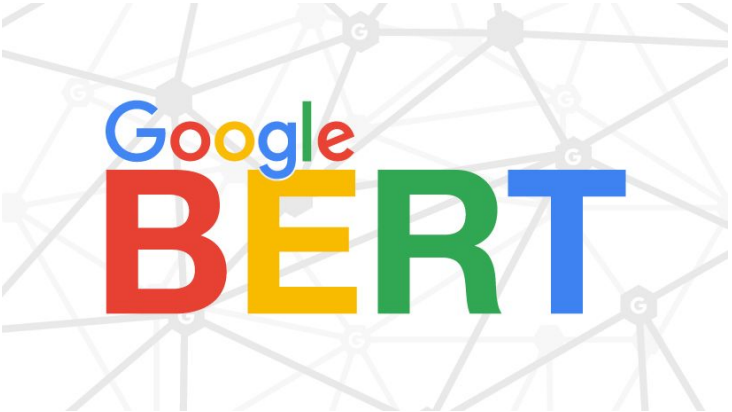
The transformer architecture

BERT

- Transformer based model

- Pre-trained

- Able to learn general representations



The BERT Tokenizer

```
# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

# Print the original sentence.
print(' Original: ', sentences[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))

```

```
oading BERT tokenizer...
ownloading (…)solve/main/vocab.txt: 100% 232k/232k [00:00<00:00, 348kB/s]
ownloading (…)okenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.15kB/s]
ownloading (…)lve/main/config.json: 100% 570/570 [00:00<00:00, 17.6kB/s]
 Original:  former versace store clerk sues secret black code minority shoppers
okenized:  ['former', 'versa', '##ce', 'store', 'clerk', 'sue', '##s', 'secret', 'black', 'code', 'minority', 'shop', '##pers']
oken IDs:  [2280, 18601, 3401, 3573, 7805, 9790, 2015, 3595, 2304, 3642, 7162, 4497, 7347]
```

The model

```
# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 2, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)
```

Training and validation

```
batch_size = 32

optimizer = AdamW(model.parameters(),
                  lr = 2e-5,
                  eps = 1e-8
                )
epochs = 4
```

epoch	Training Loss	Valid. Loss	Valid. Accur.	Valid. precision	Valid. recall	Valid. F1	Training Time	Validation Time
1	0.44	0.34	0.85	0.88	0.76	0.81	0:03:39	0:00:18
2	0.25	0.33	0.87	0.87	0.82	0.84	0:03:38	0:00:18
3	0.14	0.40	0.87	0.87	0.82	0.84	0:03:38	0:00:18
4	0.09	0.51	0.88	0.86	0.85	0.85	0:03:38	0:00:18

Graph of the loss

False positives and negatives

In the last batch

```
False Positives: [21 27]
False Negatives: [12 24]
False Positives Text:  21    trump assures nation that decision for syrian airstrikes came after carefully considering all his passing whims
27                                                                                                    ex-con back behind bar
Name: headline, dtype: object
False Negatives Text:  12     north korea praises trump and urges us voters to reject 'dull hillary'
24    ted cruz hits the panic button: 'we could lose both houses of congress'
```

Conclusion

- Good results for a task that seemed difficult

- Using a pre-trained model led to better performance