Nature Language Processing: Sentiment Analysis on Mental Health Problem Classification

CHAN, Chun Hin (20853893)
LAW, Hui Nok (20860535)

Youtube link: https://youtu.be/Z-gUddy29Cs

Introduction

Introduction

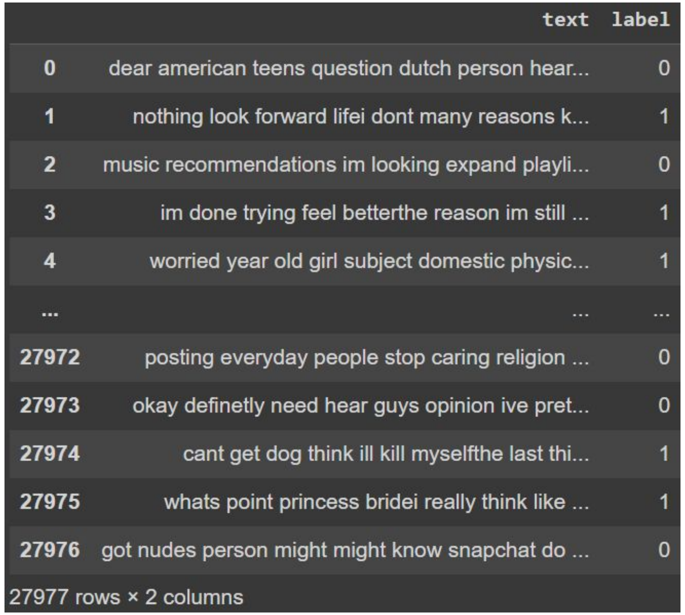
- **Source**

- Obtained from https://www.kaggle.com/datasets/reihanenamdari/mental-health-corpus
- Mental Health Corpus

Introduction

- **Structure of the Dataset**
  - Number of Data Samples: 27977
  - Number of Columns: 2

  - First Column:
    - Column Name: text
    - Data Type: string
    - People's comments
  - Second Column:
    - Column Name: label
    - Data Type: string
    - Binary: 0 or 1
    - Label 0: comment is not poisonous; commenter does not have potential mental health illnesses/issues
    - Label 1: comment is poisonous; commenter has potential mental health illnesses/issues

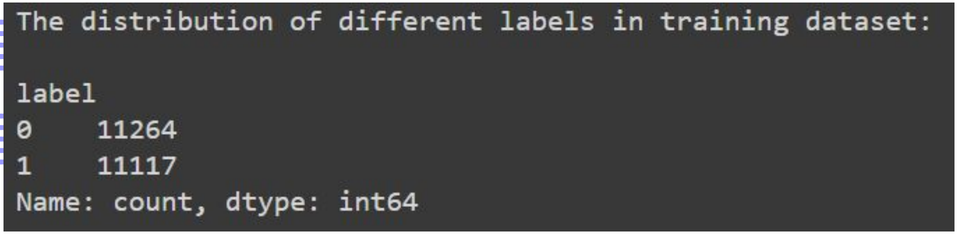


	text	label
0	dear american teens question dutch person hear...	0
1	nothing look forward lifei dont many reasons k...	1
2	music recommendations im looking expand playli...	0
3	im done trying feel betterthe reason im still ...	1
4	worried year old girl subject domestic physic...	1
...	...	...
27972	posting everyday people stop caring religion ...	0
27973	okay definetly need hear guys opinion ive pret...	0
27974	cant get dog think ill kill myselfthe last thi...	1
27975	whats point princess bridei really think like ...	1
27976	got nudes person might might know snapchat do ...	0

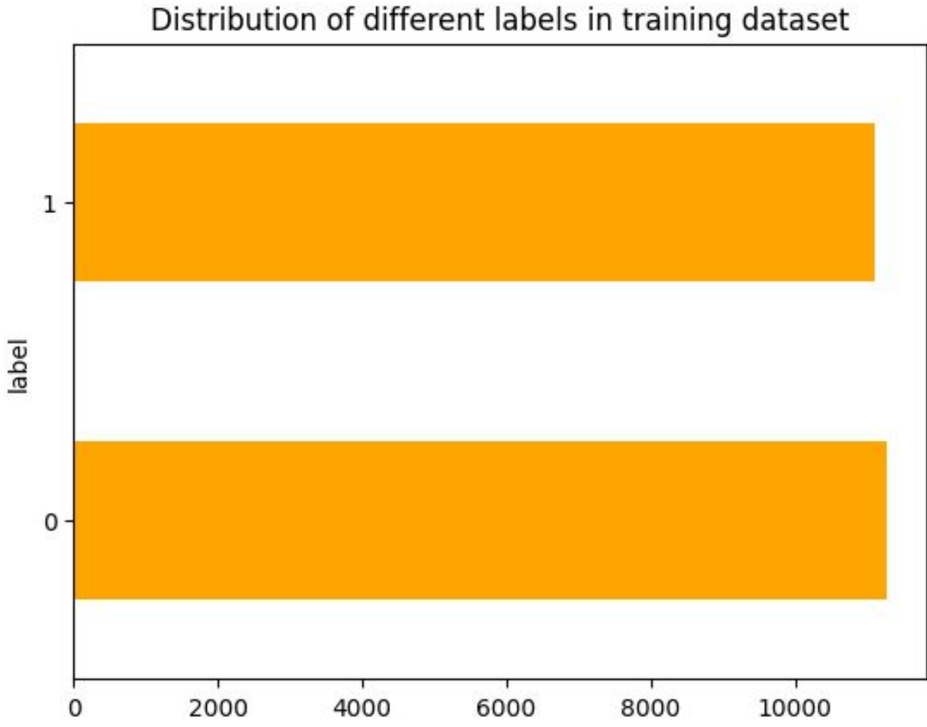
27977 rows × 2 columns

Introduction

- **Distribution of the Labels**
  - Training Dataset:
    - Number of Label 0: 11264
    - Number of Label 1: 11117



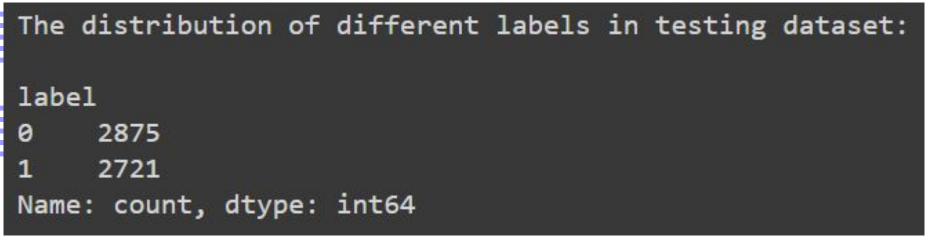
```
The distribution of different labels in training dataset:

label
0    11264
1    11117
Name: count, dtype: int64
```



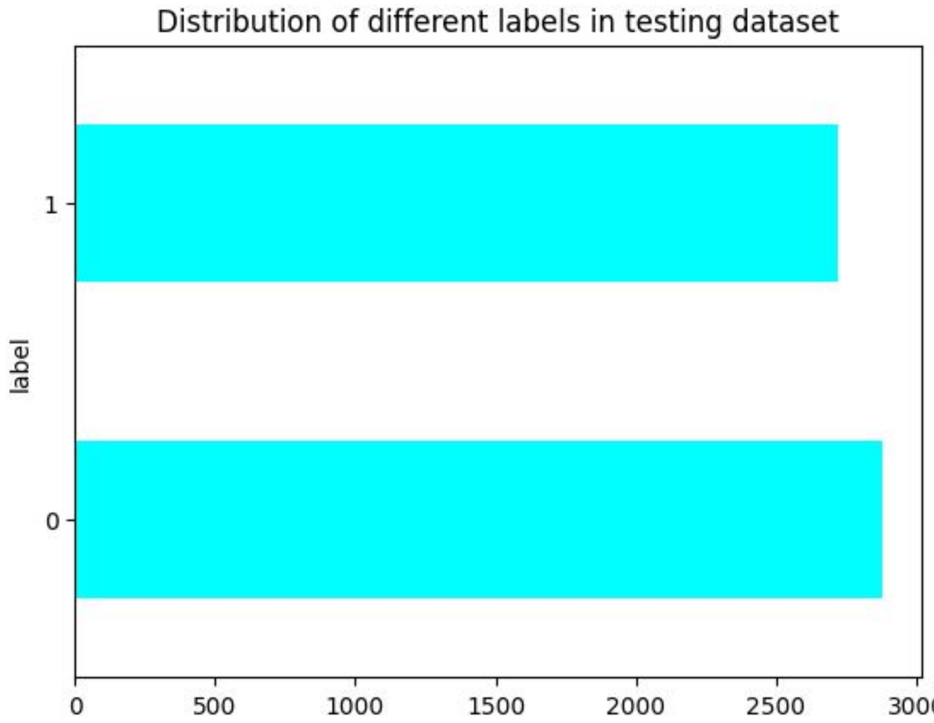
Introduction

- **Distribution of the Labels**

- Testing Dataset:
    - Number of Label 0: 2875
    - Number of Label 1: 2721



```
The distribution of different labels in testing dataset:

label
0    2875
1    2721
Name: count, dtype: int64
```



Introduction

- **Distribution of the Labels**

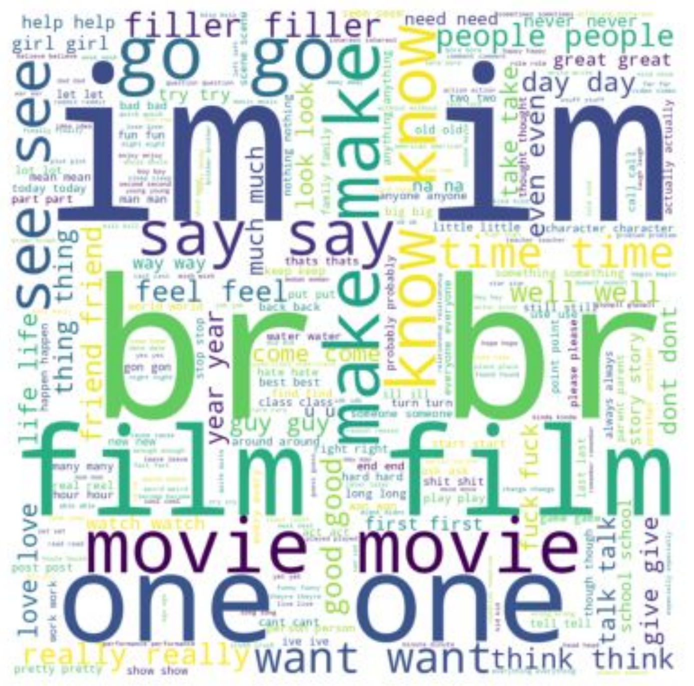
- Total:
    - Number of Label 0: 14139
    - Number of Label 1: 13838

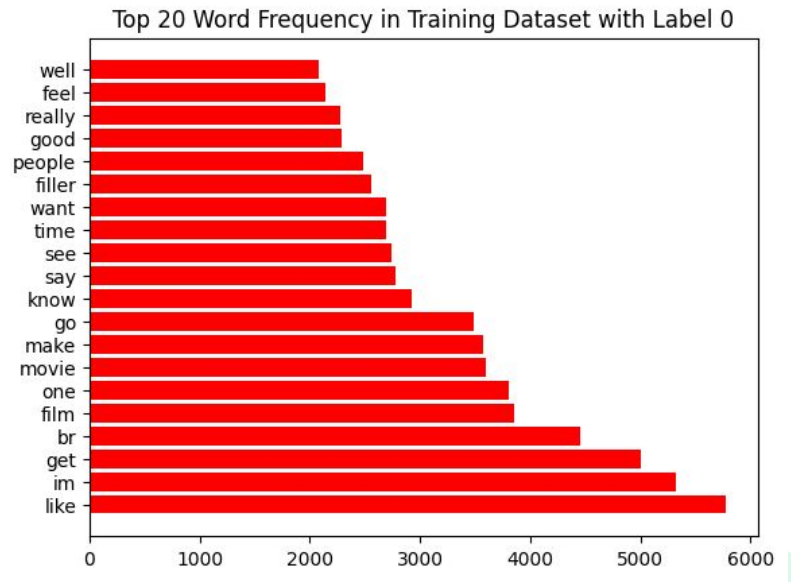
- Evenly Distributed

Introduction

- **Most Frequently Occurred Words in Each Label**

- Training Dataset (with Label 0)





Introduction

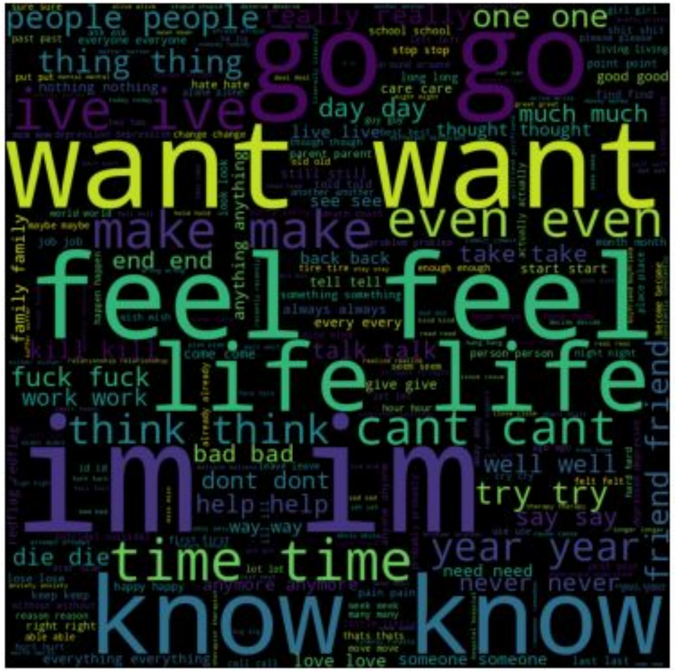
- **Most Frequently Occurred Words in Each Label**
- Training Dataset (with Label 0)

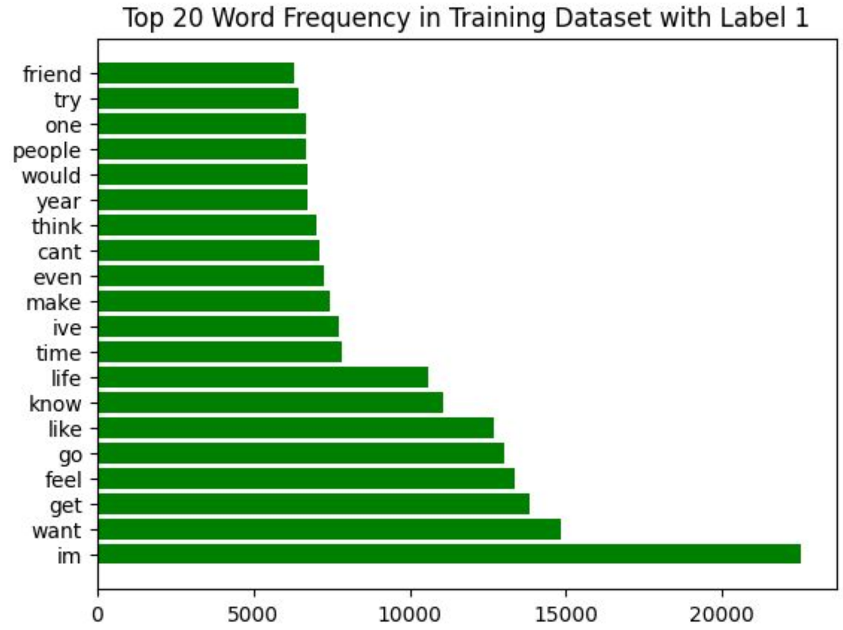
like	5777	one	3800	say	2783	people	2487
im	5326	movie	3592	see	2740	good	2286
get	5004	make	3576	time	2697	really	2275
br	4453	go	3485	want	2691	feel	2138
film	3851	know	2921	filter	2553	well	2085

Introduction

- **Most Frequently Occurred Words in Each Label**

- Training Dataset (with Label 1)





Introduction

- **Most Frequently Occurred Words in Each Label**

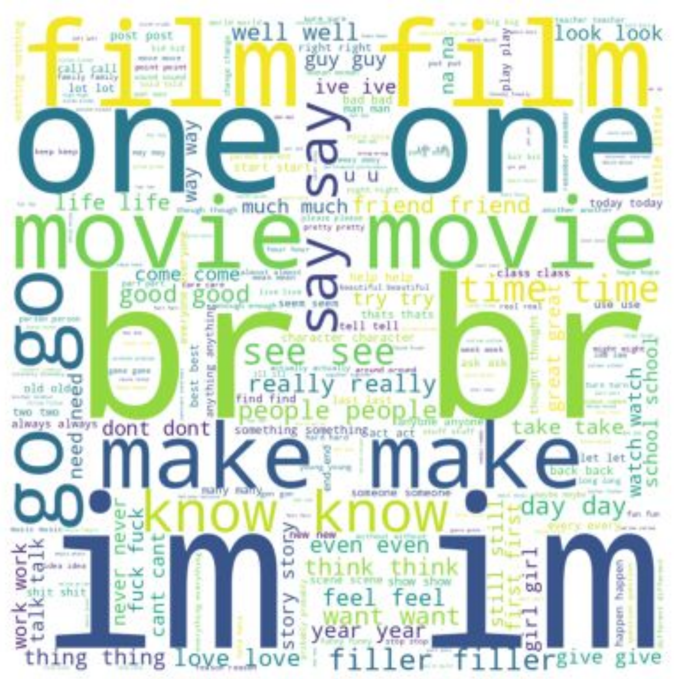
- Training Dataset (with Label 1)

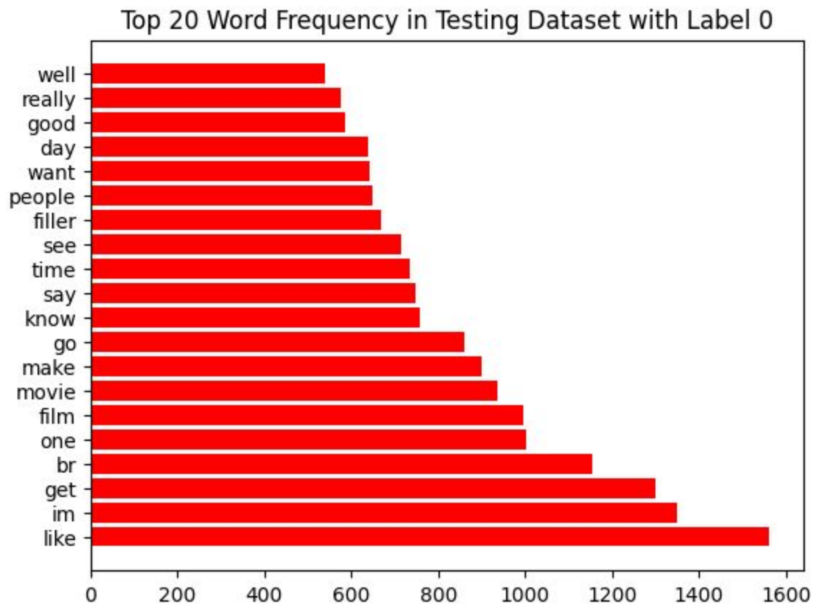
im	22520	like	12712	make	7426	would	6726
want	14818	know	11091	even	7249	people	6694
get	13856	life	10611	cant	7095	one	6688
feel	13362	time	7813	think	6999	try	6448
go	13031	ive	7744	year	6740	friend	6286

Introduction

- **Most Frequently Occurred Words in Each Label**

- Testing Dataset (with Label 0)





Introduction

- **Most Frequently Occurred Words in Each Label**

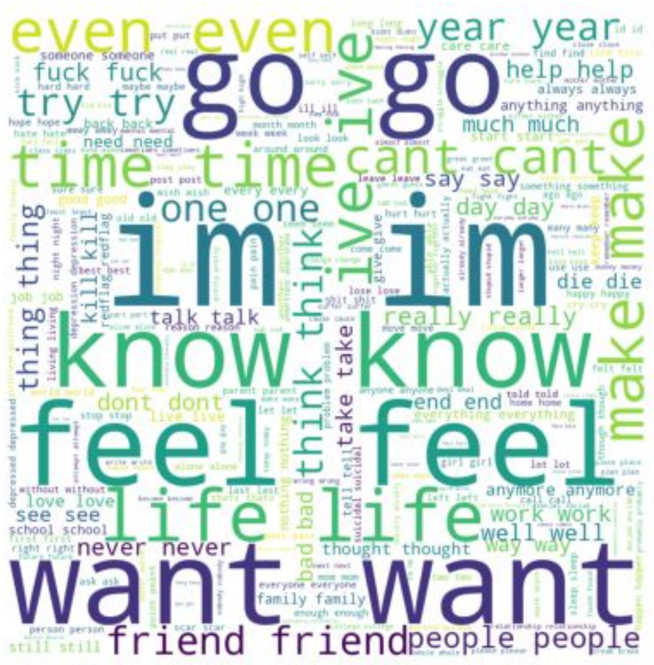
- Testing Dataset (with Label 0)

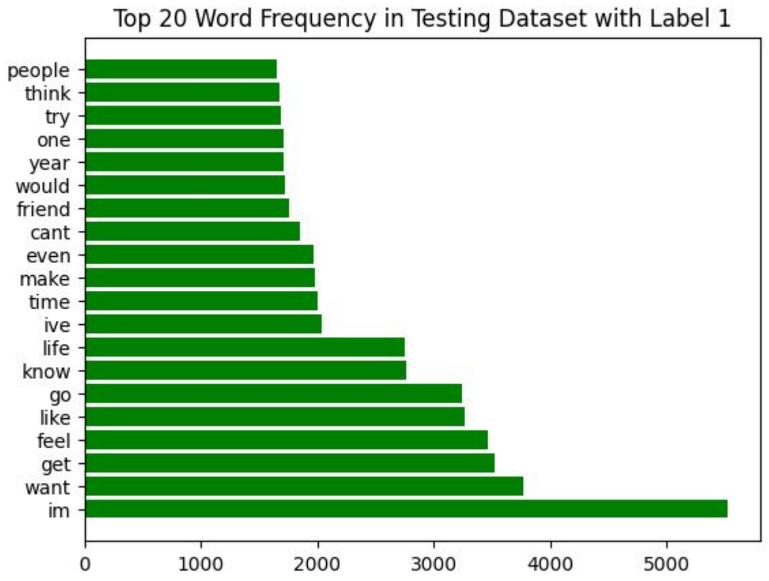
like	1562	film	997	say	748	want	642
im	1350	movie	935	time	733	day	639
get	1299	make	899	see	714	good	585
br	1155	go	859	filler	668	really	577
one	1001	know	758	people	647	well	539

Introduction

- **Most Frequently Occurred Words in Each Label**

- Testing Dataset (with Label 1)





Introduction

- **Most Frequently Occurred Words in Each Label**

- Testing Dataset (with Label 1)

im	5524	go	3244	make	1982	year	1714
want	3762	know	2767	even	1965	one	1704
get	3520	life	2747	cant	1846	try	1680
feel	3464	ive	2035	friend	1752	think	1678
like	3265	time	2006	would	1719	people	1656

Data Preprocessing

Data Preprocessing Technique

1) Lowercase

- Convert the words in String or List to lowercase
- Reducing the dimensionality when building corpus
- Stopwords removal is based on lowercase

Example:

Original Text: We are doing COMP 4211 Project!

After Processing: we are doing comp 4211 project!

Data Preprocessing Technique

2) Tokenization

- Separate all words, punctuations, and numbers into token
- Stemming, Removing Punctuations, and other preprocessing require a single token to process

Example:

Original Text in String: We are doing COMP 4211 Project!

After Processing in List of Token: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]

Data Preprocessing Technique

3) Removing Punctuations & Numbers

- Punctuations do not have much value in the analysis
- Number do not play a significant role in mental health classification
- Reducing the dimensionality of the token when building corpus

Example:

Original Text in a List of Token: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]

After Processing: [“We”, “are”, “doing”, “COMP”, “Project”]

Data Preprocessing Technique

4) Stemming or Lemmatization

Stemming

- Porter Stemming Algorithm

- Removing the prefix and suffix (in simple understanding, since this is not the main focus)

- Some meaningless word will appear such as “This” will become “Thi”

Lemmatization (We use this approach)

- Based on the part of speech of each word in the sentence, convert them back to the original form.

Example:
Original Text: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]
After Processing: [“We”, “be”, “do”, “COMP”, “4211”, “Project”, “!”]

Data Preprocessing Technique

5) Filter Stopwords

- Removing the stopwords (e.g. “is”, “the”) in the sentences since they do not carry meaningful information.

Example:

Original Text: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]

After Processing: [“We”, “doing”, “COMP”, “Project”]

Data Preprocessing Technique

6) N-Gram

- Capture the context and semantic information by considering a sequence of ‘n’ words together

Example:

Original Text: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]

N = 1: [“We”, “are”, “doing”, “COMP”, “4211”, “Project”, “!”]

N = 2: [“We are”, “are doing”, “ doing COMP”, “COMP 4211”, “4211 Project”, “Project !”]

Data Preprocessing Technique

7) Building Corpus

- Build the words corpus for the one-hot and index encoding

- <pad> = 0 is for the encoding with same length

- <unk> = 1 if for the words not find in the corpus

Example:

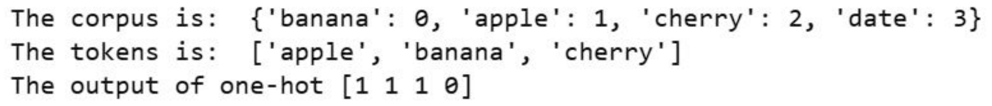
```
The token list is:  ['apple', 'banana', 'cherry', 'apple', 'banana', 'apple', 'orange']
The output corpus with minimum frequancy 2 is:  {'<pad>': 0, '<unk>': 1, 'apple': 2, 'banana': 3}  with length =  4
```

Data Preprocessing Technique

8) One-Hot Encoding

- Build the vector based on the corpus length

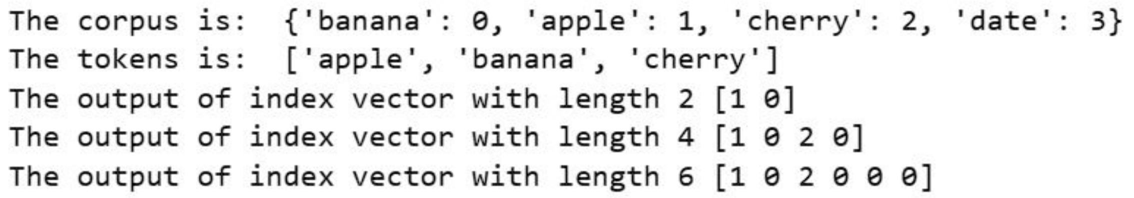
- If the words in the sentence are in the corpus, then that position will mark as 1.



```
The corpus is:  {'banana': 0, 'apple': 1, 'cherry': 2, 'date': 3}
The tokens is:  ['apple', 'banana', 'cherry']
The output of one-hot [1 1 1 0]
```

9) Index Encoding

- The first n words we consider in the sentence and for the vector with its corpus index



```
The corpus is:  {'banana': 0, 'apple': 1, 'cherry': 2, 'date': 3}
The tokens is:  ['apple', 'banana', 'cherry']
The output of index vector with length 2 [1 0]
The output of index vector with length 4 [1 0 2 0]
The output of index vector with length 6 [1 0 2 0 0 0]
```

Machine Learning Method

Machine Learning Method

1) Naive Bayes Classifier

- "Naive" assumes each piece of evidence is independent and equally contributes to the classes

$$
\begin{aligned}
P(B_i|E) &= \frac{P(B_i)P((e_1, e_2, e_3, \ldots, e_d)|B_i)}{\sum_{j=1}^{n} P((e_1, e_2, e_3, \ldots, e_d)|B_j)P(B_j)} \\
&= \frac{P(B_i)P(e_1|B_i)P(e_2|B_i)P(e_3|B_i)\ldots P(e_d|B_i)}{\sum_{j=1}^{n} P(e_1|B_j)P(e_2|B_j)P(e_3|B_j)\ldots P(e_d|B_j)P(B_j)}
\end{aligned}
$$

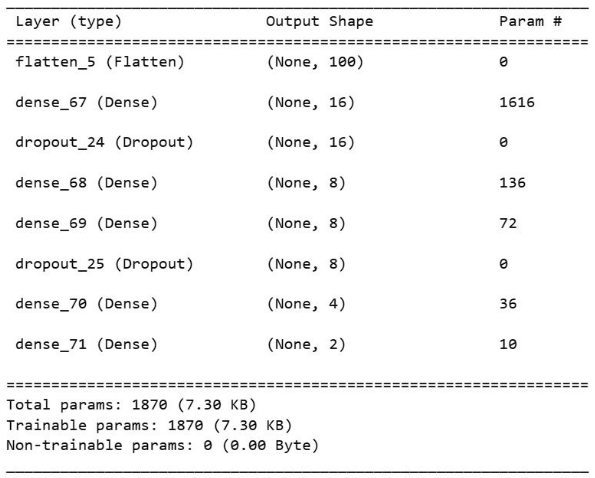
- Just needs the numerator since the denominator is the same for each Believe
- We take the log to prevent the floating point underflow
- Take the argmax to get which class belong to

$$
B_{NB} = argmax_{B_i}\left(logP(B_i) + \sum_{n=1}^{d} logP(e_n|B_i) \right)
$$

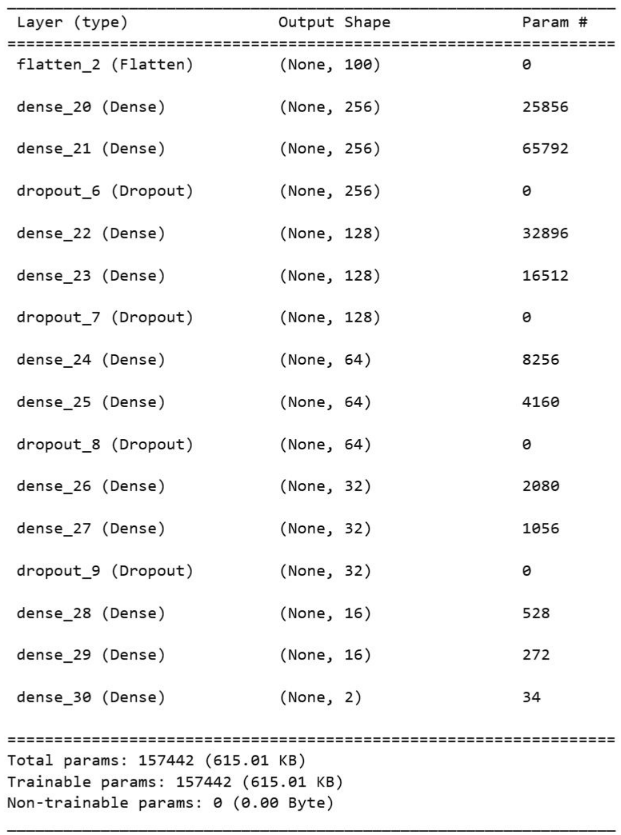
Machine Learning Method

2) One-Hot Model by MLP

- We have vector length = 100 and 1000 for the experiment
- Using the One-Hot preprocessing function to encode the sentence
- We have two models (one is simpler and one is more complex)
- Optimizer is using “adam”, Loss is using “binary_crossentropy”



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_5 (Flatten)         (None, 100)               0

 dense_67 (Dense)            (None, 16)                1616

 dropout_24 (Dropout)        (None, 16)                0

 dense_68 (Dense)            (None, 8)                 136

 dense_69 (Dense)            (None, 8)                 72

 dropout_25 (Dropout)        (None, 8)                 0

 dense_70 (Dense)            (None, 4)                 36

 dense_71 (Dense)            (None, 2)                 10

=================================================================
Total params: 1870 (7.30 KB)
Trainable params: 1870 (7.30 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

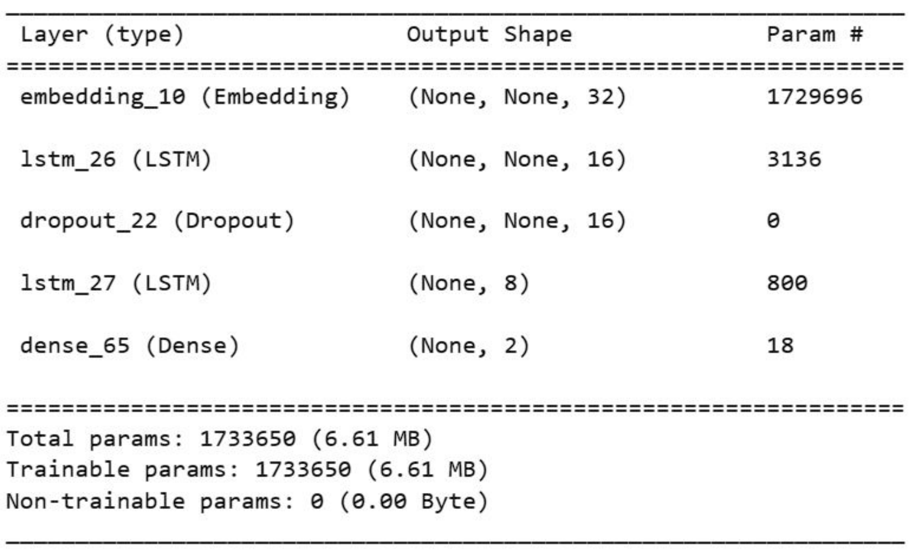


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 100)               0

 dense_20 (Dense)            (None, 256)               25856

 dense_21 (Dense)            (None, 256)               65792

 dropout_6 (Dropout)         (None, 256)               0

 dense_22 (Dense)            (None, 128)               32896

 dense_23 (Dense)            (None, 128)               16512

 dropout_7 (Dropout)         (None, 128)               0

 dense_24 (Dense)            (None, 64)                8256

 dense_25 (Dense)            (None, 64)                4160

 dropout_8 (Dropout)         (None, 64)                0

 dense_26 (Dense)            (None, 32)                2080

 dense_27 (Dense)            (None, 32)                1056

 dropout_9 (Dropout)         (None, 32)                0

 dense_28 (Dense)            (None, 16)                528

 dense_29 (Dense)            (None, 16)                272

 dense_30 (Dense)            (None, 2)                 34

=================================================================
Total params: 157442 (615.01 KB)
Trainable params: 157442 (615.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Machine Learning Method

3) Index Model by RNN

- We build the corpus for all words
- Get the Index Vector by the function with first 100 words in the sentence
- Optimizer is using “adam”, Loss is using “binary_crossentropy”

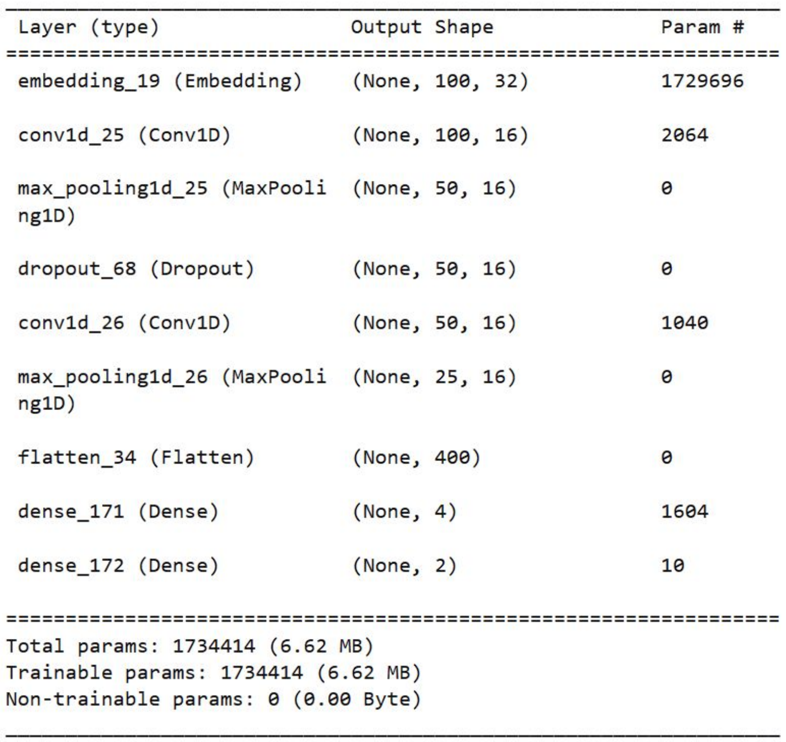


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, None, 32)          1729696

 lstm_26 (LSTM)              (None, None, 16)          3136

 dropout_22 (Dropout)        (None, None, 16)          0

 lstm_27 (LSTM)              (None, 8)                 800

 dense_65 (Dense)            (None, 2)                 18

=================================================================
Total params: 1733650 (6.61 MB)
Trainable params: 1733650 (6.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Machine Learning Method

4) **CNN Model**

- We build the corpus for all words
- Get the Index Vector by the function with first 100 words in the sentence
- Optimizer is using “adam”, Loss is using “binary_crossentropy”

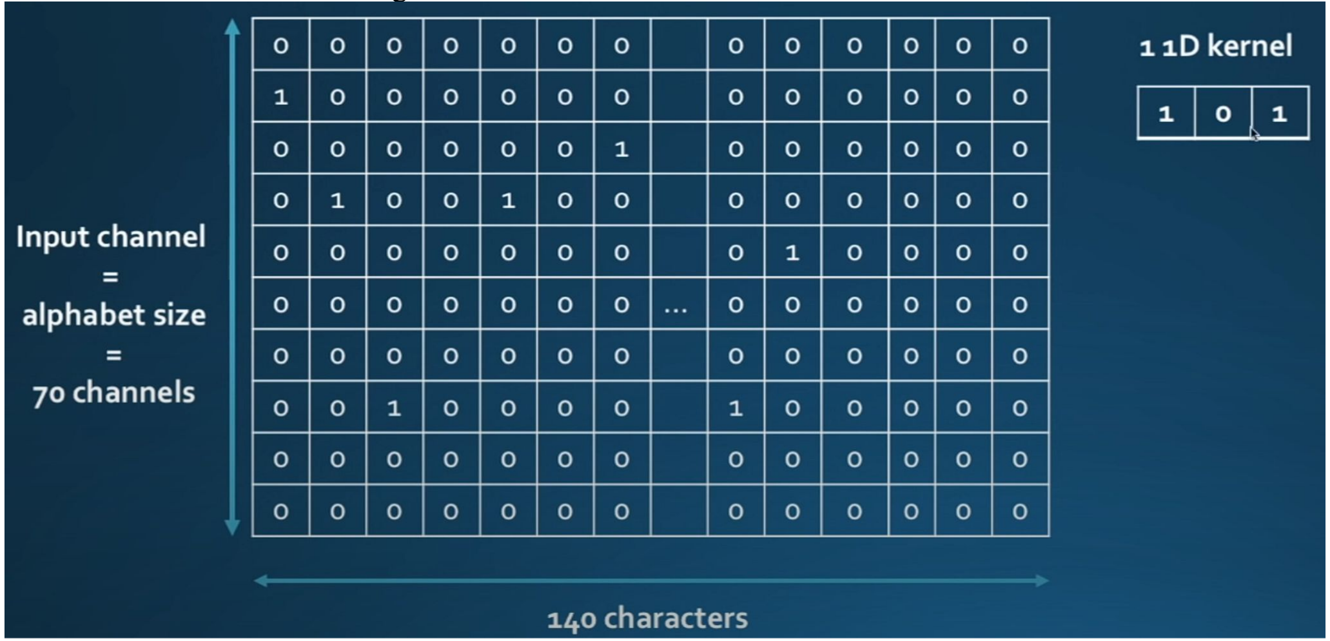


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_19 (Embedding)    (None, 100, 32)           1729696

 conv1d_25 (Conv1D)          (None, 100, 16)           2064

 max_pooling1d_25 (MaxPooli  (None, 50, 16)            0
 ng1D)

 dropout_68 (Dropout)        (None, 50, 16)            0

 conv1d_26 (Conv1D)          (None, 50, 16)            1040

 max_pooling1d_26 (MaxPooli  (None, 25, 16)            0
 ng1D)

 flatten_34 (Flatten)        (None, 400)               0

 dense_171 (Dense)           (None, 4)                 1604

 dense_172 (Dense)           (None, 2)                 10

=================================================================
Total params: 1734414 (6.62 MB)
Trainable params: 1734414 (6.62 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Machine Learning Method

4) CNN Model

- Each Column is the encoding of a word



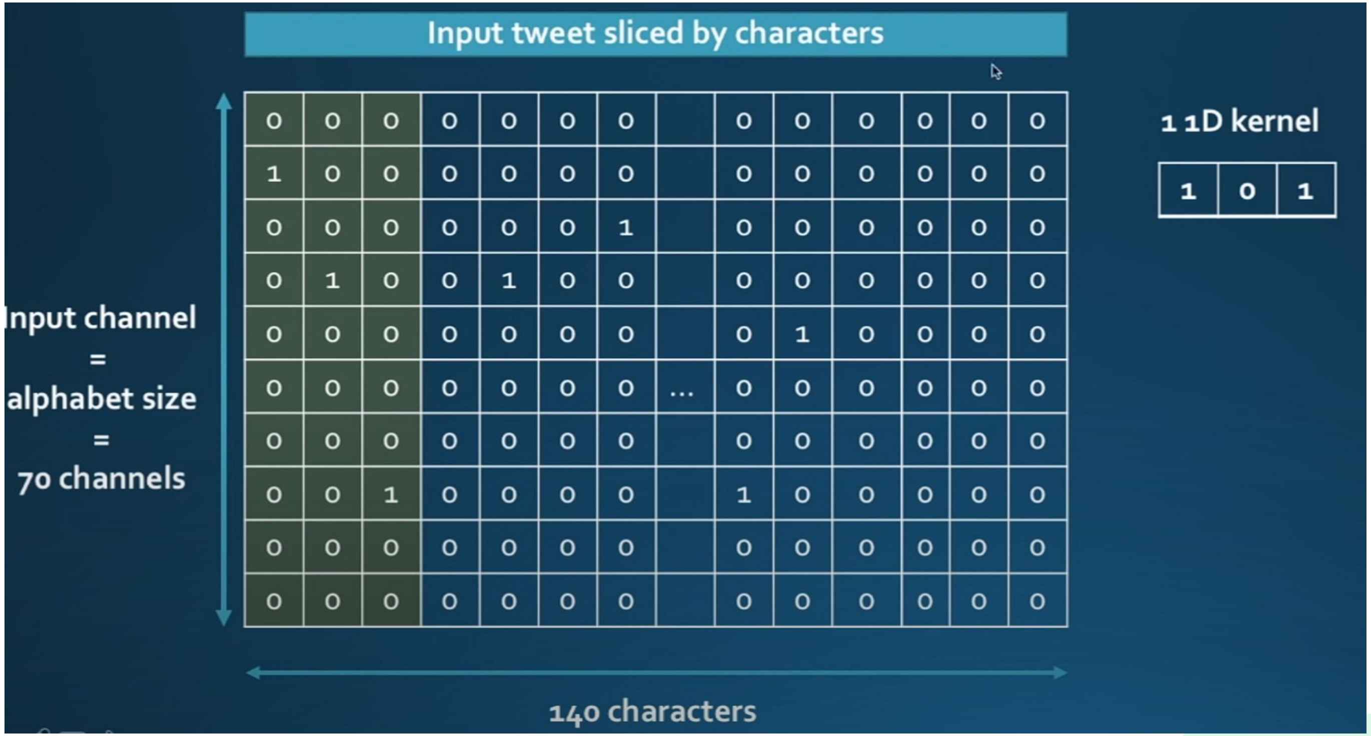
0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	1		0	0	0	0	0	0
0	1	0	0	1	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	1	0	0	0	0
0	0	0	0	0	0	0	...	0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0	0	0		1	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0

1	0	1

(https://www.youtube.com/watch?v=CNY8VjJt-iQ&ab_channel=AhmedBesbes)

Machine Learning Method

4) **CNN Model**

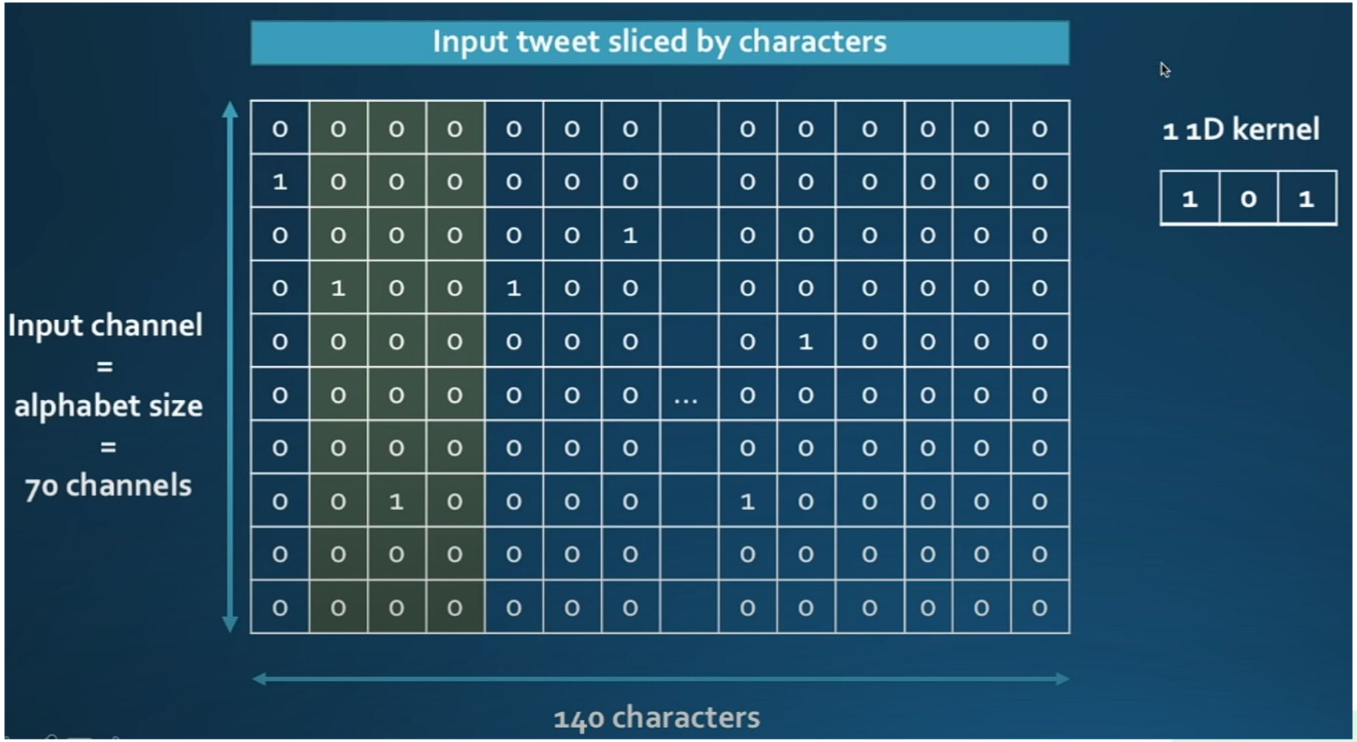

Input tweet sliced by characters
Input channel = alphabet size = 70 channels
140 characters
1 1D kernel

0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	1		0	0	0	0	0	0
0	1	0	0	1	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	1	0	0	0	0
0	0	0	0	0	0	0	...	0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0	0	0		1	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0

1	0	1

Machine Learning Method

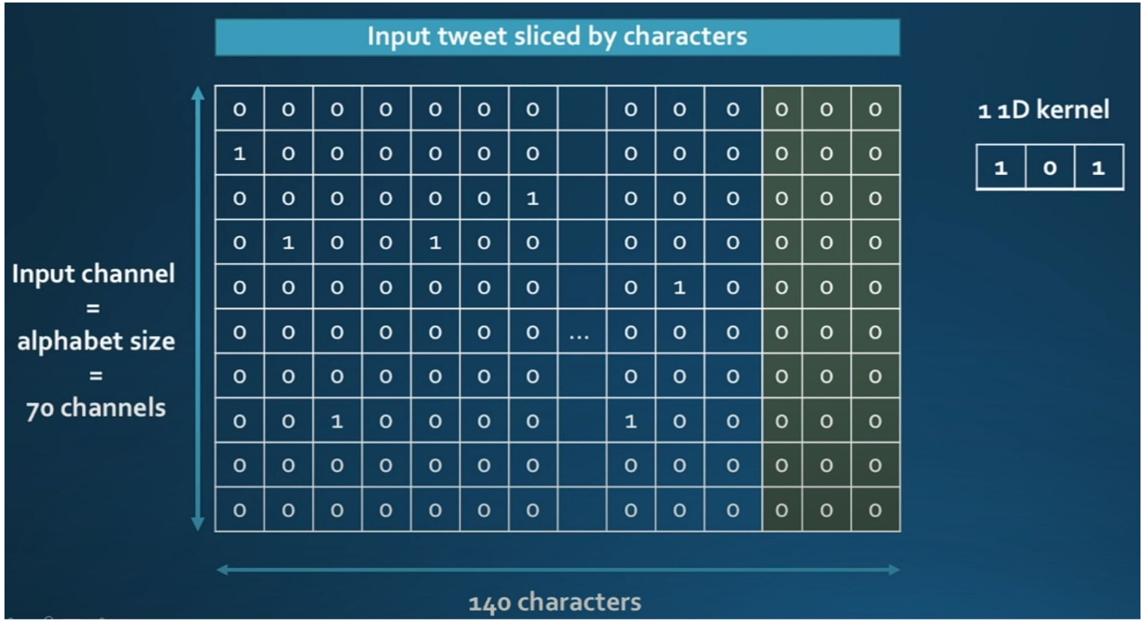
4) CNN Model


Input tweet sliced by characters
Input channel = alphabet size = 70 channels
140 characters
1 1D kernel
1 0 1

0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	1		0	0	0	0	0	0
0	1	0	0	1	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	1	0	0	0	0
0	0	0	0	0	0	0	...	0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0	0	0		1	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0

Machine Learning Method

4) **CNN Model**


Input tweet sliced by characters
Input channel = alphabet size = 70 channels
140 characters
1 1D kernel

0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	1		0	0	0	0	0	0
0	1	0	0	1	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	1	0	0	0	0
0	0	0	0	0	0	0	...	0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0	0	0		1	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0

1	0	1

Batch: 1 image
Input Channel: 70
Output Channel: 1
Output Shape: (1,1,138)

Machine Learning Method

5) DistilBERT Model

- "Distil" stands for distillation which means this model is a simple version of the original BERT model

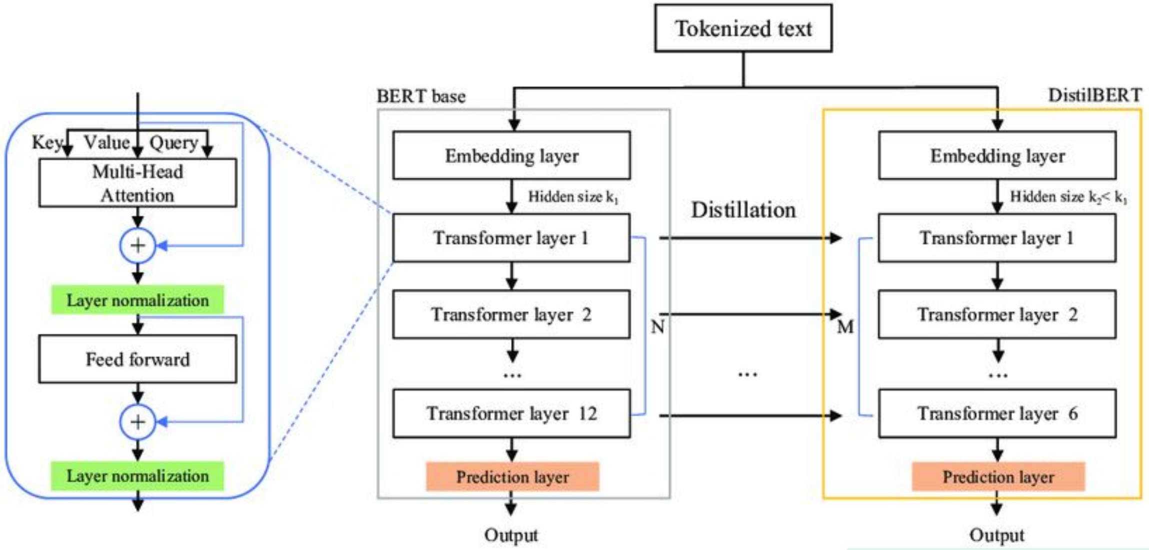
Architecture \ Model	DistilBERT	BERT
Embedding vector for word	Yes	Yes
Positional Encoding	Yes	Yes
# of Encoder (Base)	6	12
Encoder details	Self-attention + add & normalize + FFNN	Self-attention + add & normalize + FFNN

- After that then perform the downstream task

Machine Learning Method

5) DistilBERT Model

Initialization:
The first encoder's weight of the DistilBERT Model will pick either layer 1 or 2 weights from the BERT model
The second encoder's weight of the DistilBERT Model will pick either layer 3 or 4 weights from the BERT model
.
.
.
Until the last encoder



Machine Learning Method

5) DistilBERT Model

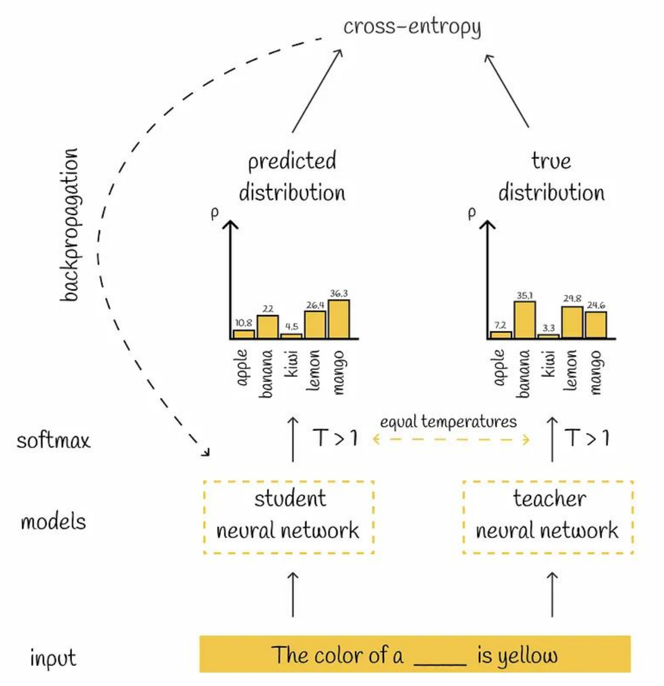
There are 3 loss function:

1) Distillation loss (cross-entropy loss)

$$L_{ce} = \sum_i t_i * \log(s_i)$$

where ti is the predicted output from BERT and si is the predicted output from DistilBERT

The function of this loss is to help the DistilBERT model learn from the BERT model, which is the student and teacher relationship.



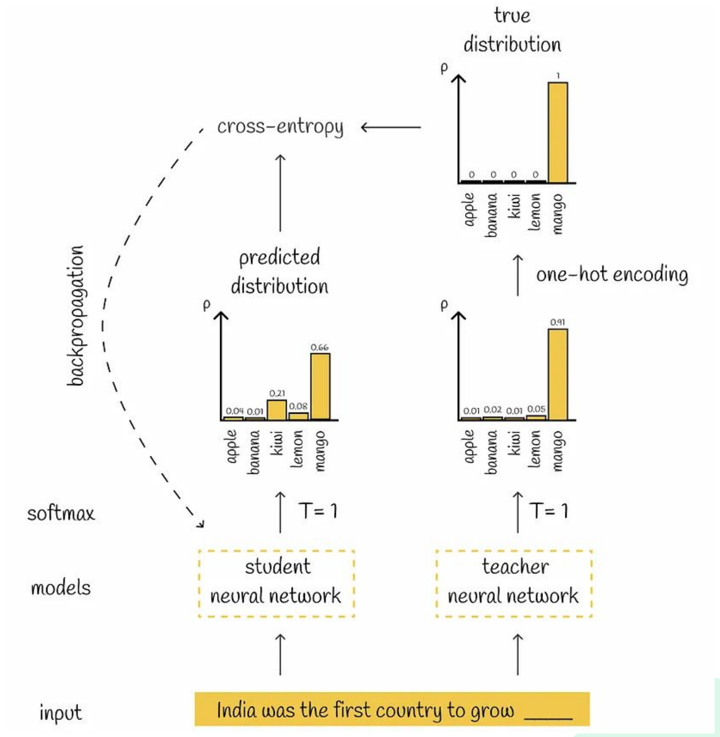
Machine Learning Method

5) DistilBERT Model

There are 3 loss function:

2) Masked Language Modeling (the same one as BERT)

- Also is a cross entropy loss
- Randomly masking 15% of the input and predicting the masked words
- Learn the bidirectional representation of the sentence



Machine Learning Method

5) DistilBERT Model

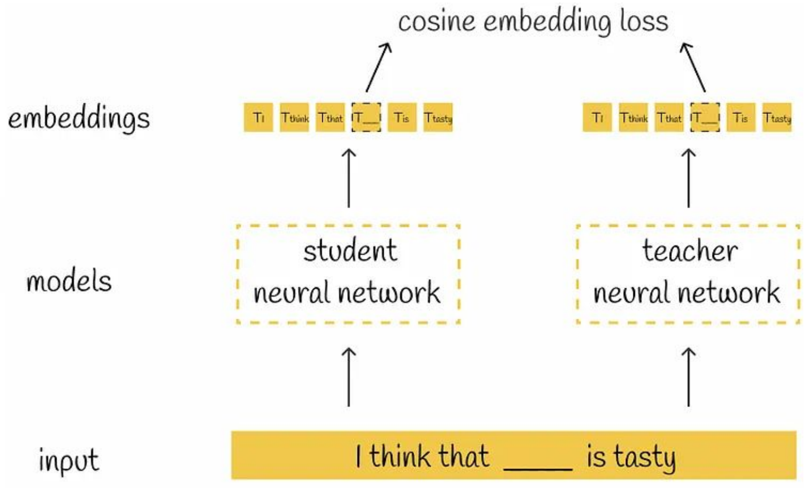
There are 3 loss function:

1) Cosine Embedding Loss

$$L_{cos}(x, y) = 1 - \cos(x, y) = 1 - \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

- The x and y are the predicted output after the BERT model's encoders and the DistilBERT model's encoders respectively.

- Help the DistilBERT model construct the embedding and mimic the behaviour of the BERT model as closely as possible


cosine embedding loss
embeddings
TI Tthink Tthat T__ Tis Ttasty
TI Tthink Tthat T__ Tis Ttasty
models
student neural network
teacher neural network
input
I think that ____ is tasty

Machine Learning Method

6) XLNet Model

- Propose by Google in 2019

- During Model Pretraining:
  - Utilizes best things from (a) Autoencoding (AE) & (b) Autoregressive (AR) Language Modeling
  - Avoid the weaknesses of both Autoencoding & Autoregressive Language Modeling

- Adopted 4 New Techniques to Boost the Performance:
  - (i) Permutation Language Modeling
  - (ii) Two-stream Self-attention
  - (iii) Relative Positional Encoding Scheme (from Transformer-XL)
  - (iv) Segment Recurrence Mechanism (from Transformer-XL)

Machine Learning Method

6) **XLNet Model**

(a) Autoencoding (AE)

- Pros: Perform Denoising
  - First model to use AE: BERT
  - Predict the randomly masked token using the bidirectional context

Machine Learning Method

6) XLNet Model

(a) Autoencoding (AE)

- Cons: Independence Assumption

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$

  - All masked tokens are separately predicted
  - Joining conditional probability is assumed to be independent to each other

  - (Solved by Autoregressive (AR) Language Modeling due to lack of this assumption)

Machine Learning Method

6) **XLNet Model**

(a) Autoencoding (AE)

- Cons: Input Noise Problem
    - BERT contains [MASK]
    - Never appears in finetuning task
    - Result in pretrain-finetune discrepancy

    - (Solved by Autoregressive (AR) Language Modeling due to lack of dependence on any masked input words)

Machine Learning Method

6) XLNet Model

(b) Autoregressive (AR) Language Modeling

- Pros: Maximise the Likelihood using the Forward Autoregressive Factorization

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$

  - Example: Word Sequence [x_1, x_2, …, x_t]
  - Uses [x_1] to predict x_2
  - Uses [x_1, x_2] to predict x_3
  - Uses [x_1, x_2, …, x_{t-1}] to predict x_t

Machine Learning Method

6) XLNet Model

(b) Autoregressive (AR) Language Modeling

- Cons: <u>Only uses the context on the Left Side of the Token for Prediction</u>
  - Cannot uses the context on the right side of the token for prediction
  - (Solved by Autoencoding (AE) due to the bidirectional context dependency)

Machine Learning Method

6) XLNet Model

(i) Permutation Language Modeling

- Aim: Collect bidirectional context information
- Prevent: independence assumption & pretrain-finetune discrepancy problems

- Borrowed from orderless NADE
- Key idea: Permutation
- Sequence **x** with length of T
- T! Different orderings to perform autoregressive factorization
- Bidirectional context capturing is possible

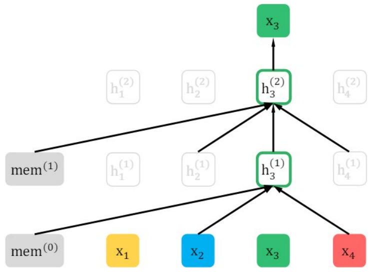
$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]$$

Machine Learning Method

6) **XLNet Model**

(i) Permutation Language Modeling

- How to capture the bidirectional context without any masked tokens?
  - Given: a text sequence **x**
  - sample a factorization order **z** at one time
  - decompose the likelihood $p_\theta(\mathbf{x})$ according to factorization orders
  - x_t would see every possible element $x_i \neq x_t$ in the sequence



Factorization order: 2 → 4 → 3 → 1

Machine Learning Method

6) XLNet Model

(i) Permutation Language Modeling

- How to capture the bidirectional context without any masked tokens?
  - Initial sequence: [x_1, x_2, x_3, x_4]
  - New sequence after one round of permutation: [x_2, x_4, x_3, x_1]

  - Prediction target: x_3
  - Can attend to the context on the left x_2
  - Can attend to the context on the right x_4
  - x_3 can attend to [x_2, x_4, x_3]

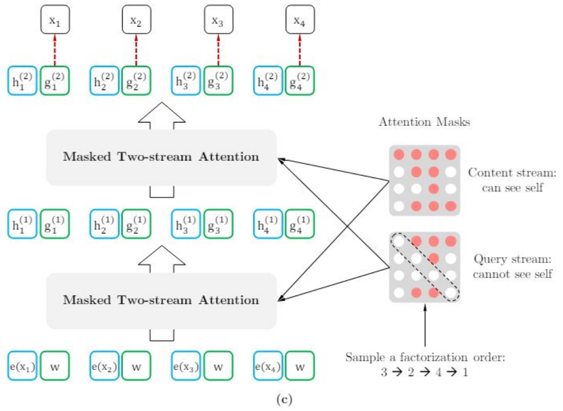
  - 2 points to note:
  - mem$^{(0)}$ is the architecture of Transformer-XL
  - Permutation would not change the actual attention matrix

Machine Learning Method

6) XLNet Model

(ii) Two-stream Self-attention

- Aim: Predict the location of a word token & bidirectional context relationship
- 2 streams:
  - Content stream
  - Query stream



x_1 x_2 x_3 x_4
$h_1^{(2)}$ $g_1^{(2)}$ $h_2^{(2)}$ $g_2^{(2)}$ $h_3^{(2)}$ $g_3^{(2)}$ $h_4^{(2)}$ $g_4^{(2)}$
Masked Two-stream Attention
$h_1^{(1)}$ $g_1^{(1)}$ $h_2^{(1)}$ $g_2^{(1)}$ $h_3^{(1)}$ $g_3^{(1)}$ $h_4^{(1)}$ $g_4^{(1)}$
Masked Two-stream Attention
$e(x_1)$ w $e(x_2)$ w $e(x_3)$ w $e(x_4)$ w
Attention Masks
Content stream: can see self
Query stream: cannot see self
Sample a factorization order: 3 → 2 → 4 → 1
(c)

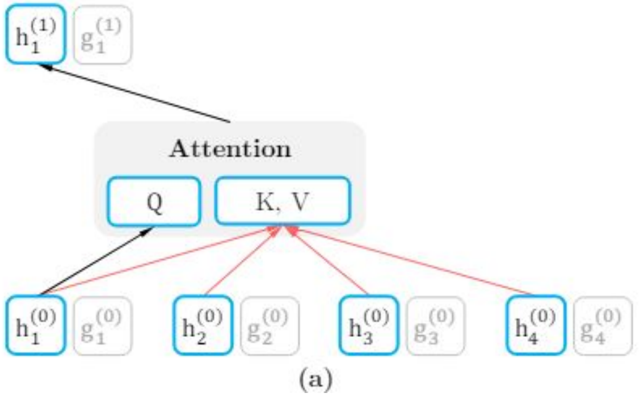
Machine Learning Method

6) XLNet Model

(ii) Two-stream Self-attention

- Content stream:
  - Aim: learn the bidirectional context
  - Same as the standard self-attention
  - Content representation h: similar to a standard hidden state (including context and the context information)

  - Example:
  - h_1 is Q
  - h_1 to h_4 as both K and V
  - h_1 would perform attention with x_1 to x_4 & gives attention weight
  - multiply with V to get the representation of h_1 in the second layer



$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad \text{(content stream: use both } z_t \text{ and } x_{z_t}\text{)}$$

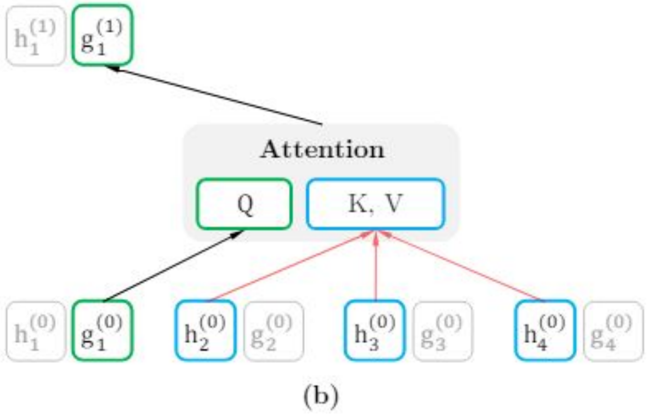
Machine Learning Method

6) XLNet Model

(ii) Two-stream Self-attention

- Query stream:
  - Aim: Replace the [MASK] token in BERT
  - Query representation g: contains context information and position (not including content)

  - Example:
  - g_1 is Q
  - h_2 to h_4 is K and V
  - Apply the equation below
  - g_1 attend to h_2 to h_4



$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = g_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad \text{(query stream: use } z_t \text{ but cannot see } x_{z_t}\text{)}$$

Machine Learning Method

6) XLNet Model

(iii) Relative Positional Encoding Scheme (from Transformer-XL)

- Aim: Tackle the weakness of absolute positional encoding
- Consider relationship between a word position relative to another word position

- Randomly initializing the word embeddings in each position
- Create the pair-wise embedding matrix of size [T, 2*T - 1]
- Row index: the word we concerned about
- Column index: the relative positional distance from the word we are concerned about to other words (including both before and after the concerned word)

Machine Learning Method

6) XLNet Model

(iii) Relative Positional Encoding Scheme (from Transformer-XL)

- Example:
- Simple sentence: "***I like cats and dogs***"
- Get 2-dimensional vector encoding for each word after encoding:

I	[0.3, 0.7]
like	[0.2, 0.9]
cats	[0.4, 0.5]
and	[0.1, 0.3]
dogs	[0.8, 0.4]

Machine Learning Method

6) XLNet Model

(iii) Relative Positional Encoding Scheme (from Transformer-XL)

- 2D relative positional distance:

Relative position of “like” relative to “I”	[0.1, 0.2]	Relative position of “and” relative to “like”	[0.3, 0.2]
Relative position of “cats” relative to “I”	[0.2, 0.4]	Relative position of “dogs” relative to “like”	[0.1, 0.1]
Relative position of “and” relative to “I”	[0.3, 0.7]	Relative position of “and” relative to “cats”	[0.3, 0.4]
Relative position of “dogs” relative to “I”	[0.4, 0.9]	Relative position of “dogs” relative to “cats”	[0.3, 0.6]
Relative position of “cats” relative to “like”	[0.1, 0.1]	Relative position of “dogs” relative to “and”	[0.3, 0.5]

- Closer relationship: “like” and “cats”, “like” and “dogs”
- distance vector is smaller than other pairwise vector distances

Machine Learning Method

6) XLNet Model

(iv) Segment Recurrence Mechanism (from Transformer-XL)

- Aim: each sentence segment from the past can be used with a new sentence segment
- Avoid the same fixed-length context fragmentation embodiment

- Equation for updating the attention with memory for the next segment **x**:

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \left[\tilde{\mathbf{h}}^{(m-1)}, \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}\right]; \theta)$$

Machine Learning Method

6) XLNet Model

(iv) Segment Recurrence Mechanism (from Transformer-XL)

- Example:
- Two segments extracted from a long sequence **s**

$$\tilde{\mathbf{x}} = \mathbf{s}_{1:T} \text{ and } \mathbf{x} = \mathbf{s}_{T+1:2T}$$

- $\tilde{\mathbf{z}}$: permutation of [1 … T]
- z: permutation of [T + 1 … 2T]

- Process the first segment
- Cache the obtained content representations $\tilde{\mathbf{h}}^{(m)}$ for each layer m
- Update the attention with memory for the next segment **x** using the equation below:

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \left[\tilde{\mathbf{h}}^{(m-1)}, \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}\right]; \theta)$$

Machine Learning Method

6) XLNet Model

(iv) Segment Recurrence Mechanism (from Transformer-XL)

- Positional encodings depends on actual positions in original sequence of words only
- Attention updating equation is independent of $\tilde{\mathbf{z}}$ after the representation of $\tilde{\mathbf{h}}^{(m)}$ is obtained
- Cache & reuse the memory (without knowing the permutation order of the previous segment)
- Expected: model can use the memory over all permutation orders of the last segment of sequence
- Expected: query stream can be computed identically.

Experiment and Result

Experiment and Result

1) Naive Bayes Classifier

- According to the research “Heart disease prediction using Naive Bayes algorithm and Laplace Smoothing technique” draws the conclusion that if a number of testing data do not appear in the training, after applying the Laplace Correction will have a significant increase in the accuracy.

Hypothesis: We can use the Naive Bayes’ accuracy to estimate is the testing data is similar to training data

Independent variable: the Laplace Correction value (ie. alpha = 0 means don’t perform and alpha = 1 means adding 1 count to each word to obtain no zero probability which is Laplace Correction)

Dependent Variable: The accuracy of both Naive Bayes classifier which has or hasn’t Laplace Correction.

Fixed Condition: The training data and testing splitting in both cases are the same and CountVectorizer() performs the counting of each word for preparing the probability in the Naive Bayes Classifier.

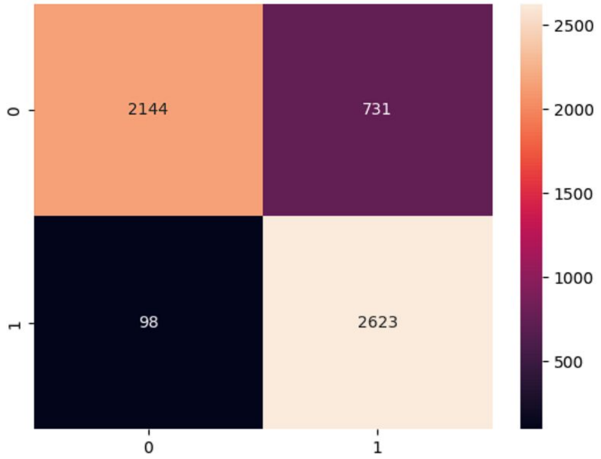
Process: We utilize MultinomialNB() with one set alpha to 0 and another one set alpha to 1 to perform the testing. Then we fit the same training data into both of them and use two models to predict on same testing data.

Experiment and Result

1) Naive Bayes Classifier

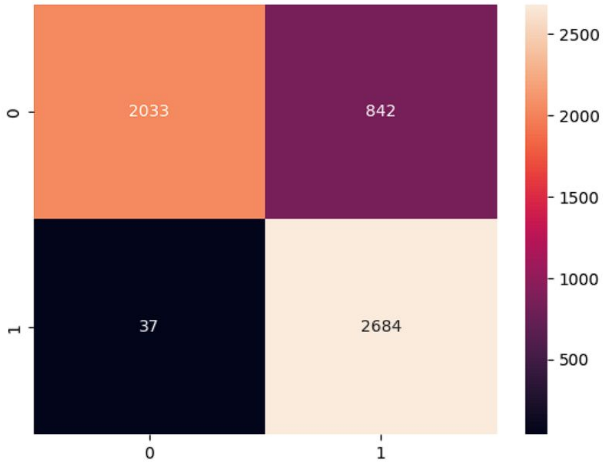
Alpha = 0:

```
Naive Bayes Accuracy without smoothing:  0.8518584703359543
Precision: 0.782051282051282
Recall: 0.9639838294744579
F1 Score: 0.8635390946502058
```



Alpha = 1:

```
Naive Bayes Accuracy with smoothing:  0.8429235167977126
Precision: 0.7612024957458877
Recall: 0.9864020580668872
F1 Score: 0.859292460380983
```



Experiment and Result

1) Naive Bayes Classifier

- Accuracy without Laplace Correction is 0.8518584703359543
- Accuracy with Laplace Correction is 0.8429235167977126
- Time usage for both trials is 1.5413191318511963 seconds

From the data distribution part:
The probability of the most frequent word appearing in the testing data which is also in the most frequent of the training data is 39/40 = 97.5%

** Utilizing the Naive Bayes with or without Laplace Correction, if the accuracy is similar which implies the data in both training and testing is also similar, if the accuracy has a significant improvement after applying Laplace Correction which implies there are more variations in the training and testing data.

Experiment and Result

2) One-Hot Model by MLP

Prior knowledge: Shorter length of the one-hot encoding will reduce the information and the dimensionality which may lower the accuracy but have a less usage of the RAM.

Question: How the model will be affected or limited by a shorter encoding.

Independent Variable: Length of the encoding or model architecture

Dependent Variable: The accuracy, precision, recall and F1 score for the three models

Process: In the first trial, we fixed the length of the one-hot encoding to be 100. Then we have one less complex MLP and one more complex MLP model for training and comparison. Afterwards, we fixed the MLP model to be the one with less complexity and performed the comparison with one using an encoding length is 100 and another one with an encoding length is 1000.

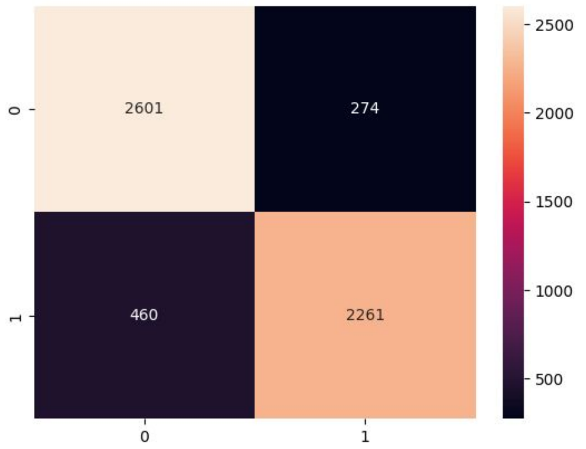
Experiment and Result

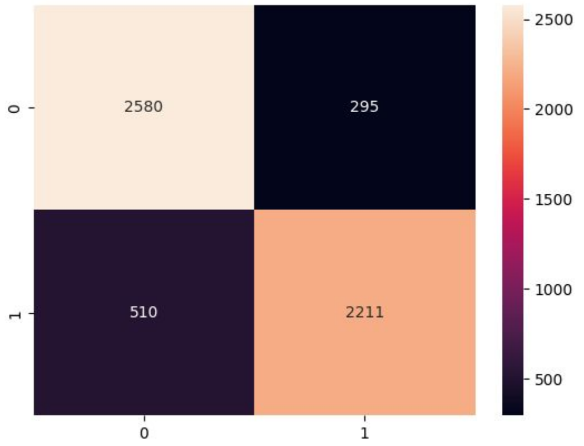
2) One-Hot Model by MLP

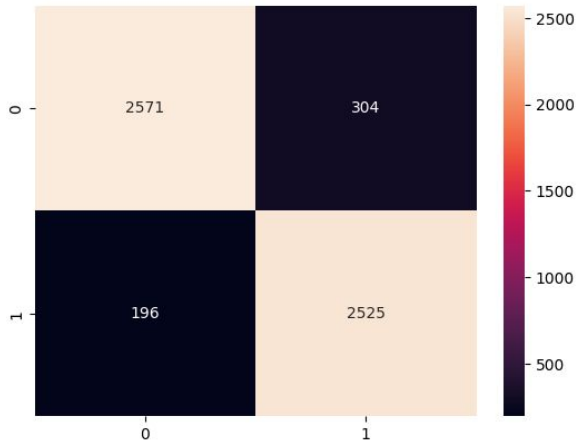
Length 100 with simple model performance: accuracy = 86.88%, the precision = 89.19%, the recall = 83.09%, the F1 score = 86.04%

Length 100 with complex model performance: accuracy = 85.61%, the precision = 88.23%, the recall = 81.26%, the F1 score = 84.60%

Length 1000 with simple model performance: accuracy = 91.07%, the precision = 89.25%, the recall = 92.80%, the F1 score = 90.99%







Experiment and Result

2) One-Hot Model by MLP

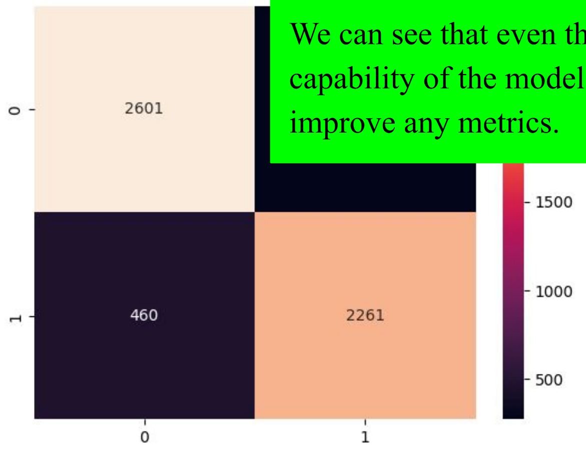
Length 100 with simple model performance: accuracy = 86.88%, the precision = 89.19%, the recall = 83.09%, the F1 score = 86.04%

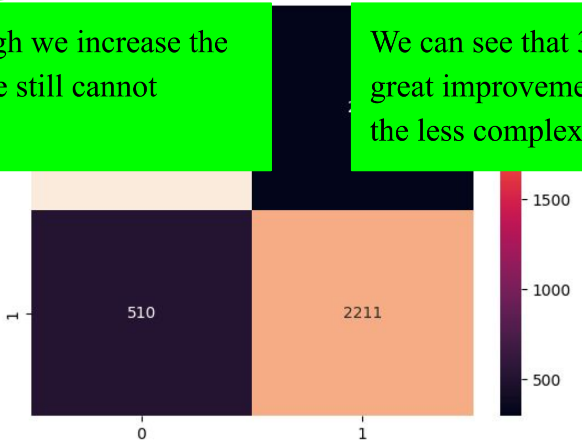
Length 100 with complex model performance: accuracy = 85.61%, the precision = 88.23%, the recall = 81.26%, the F1 score = 84.60%

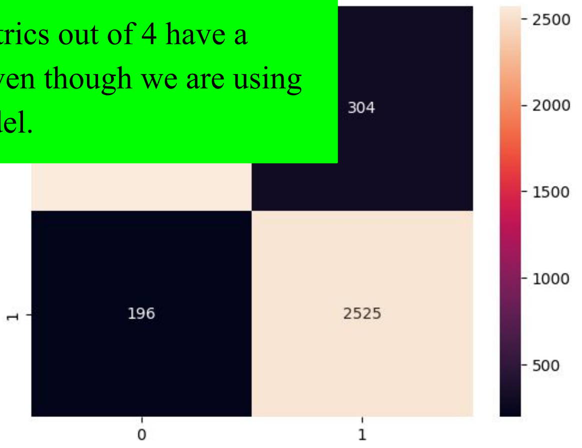
Length 1000 with simple model performance: accuracy = 91.07%, the precision = 89.25%, the recall = 92.80%, the F1 score = 90.99%

We can see that even though we increase the capability of the model, we still cannot improve any metrics.

We can see that 3 metrics out of 4 have a great improvement even though we are using the less complex model.







Conclusion: The length of the encoding is a critical factor in determining the performance of the MLP model.

Experiment and Result

3) **Model Comparison and Best Model for Mental Health Classification**

Question: There are so many models we have implemented. Which model works best for the task?

Independent Variable: 6 different models

Dependent Variable: The computational power (time and memory usage), model performance (accuracy, precision, recall and F1 score)

Process: We built all models as the last part stated with their performance. So we can directly jump to the result part.

Experiment and Result

3) Model Comparison and Best Model for Mental Health Classification

Naive Bayes Classifier:

a) Advantages

Fast, only takes 1.5413191318511963 seconds for both tasks

b) Disadvantages:

Assumption: Equal contribution and independent probability (not the case in real world)

c) Performance (with Laplace Correction)

The accuracy 84.29%, precision 76.12%, recall 98.64%, F1 score 85.93%

One-Hot MLP model

a) Advantages

Fast, 1 to 2 seconds for each epoch

b) Disadvantages:

Dense Layer treated every input as independent

One-hot encoding requires lots of RAM usage

c) Performance (Encoding Length = 100 with simple model)

The accuracy = 86.88%, the precision = 89.19%, the recall = 83.09%, the F1 score = 86.04%

Experiment and Result

3) Model Comparison and Best Model for Mental Health Classification

Index RNN model:

a) Advantages
It has embedding layer, each word can have better representation
LSTM layer can have a better semantic understanding
LSTM gate mechanism can prevent the vanishing gradient problem

b) Disadvantages:
Training time is longer, 27 - 32 seconds for each epoch
LSTM is like a “black box” and it is hard to understand the interpretation in it.

c) Performance (with Laplace Correction)
The accuracy = 88.46%, the precision = 90.04%, the recall = 85.74%, the F1 score = 87.84%

Experiment and Result

3) Model Comparison and Best Model for Mental Health Classification

CNN Model:

a) Advantages
It has embedding layer, each word can have better representation
Convolution layer can easily capture the pattern of the sentences
Training time is relatively faster, 10 seconds per each epoch

b) Disadvantages:
Cannot perform well in adversarial attacks and not general enough (only learn the pattern but not semantic meaning)

c) Performance (with Laplace Correction)
The accuracy = 91.53%, the precision = 92.87%, the recall = 89.45%, the F1 score = 91.13%

Experiment and Result

3) Model Comparison and Best Model for Mental Health Classification

DistilBERT:

a) Advantages
Contextual understanding -> self-attention mechanism
Bidirectional understanding -> Mask mechanism
Have pre-trained model

b) Disadvantages:
Training time is long, 4 minutes for each epoch

c) Performance (with Laplace Correction)
The accuracy = 92.12%, the precision = 91.16%, the recall = 92.80%, the F1 score = 91.97%

Experiment and Result

3) Model Comparison and Best Model for Mental Health Classification

XLNet Model:

a) Advantages
- Both autoencoding (AE) and autoregressive (AR) benefits -> permutation language modeling
- Better positional information -> relative positional encoding
- Two-stream self-attention mechanism (can hide the content of current position in Query) is better than self-attention mechanism in BERT
- Pre-trained model

b) Disadvantages:
Training time is long, 10 minutes for each epoch

c) Performance (with Laplace Correction)
The accuracy = 93.07%, the precision = 88.43%, the recall = 98.64%, the F1 score = 93.26%

Experiment and Result

3) **Model Comparison and Best Model for Mental Health Classification**

Which is the best model for mental health classification?

- Human language is ambiguous, so we should choose the model that can understanding the semantic meaning
❌Naive Bayes, One-Hot MLP, CNN model

- Accuracy for the three models is similar but the recall metric of the data is much more important than the others

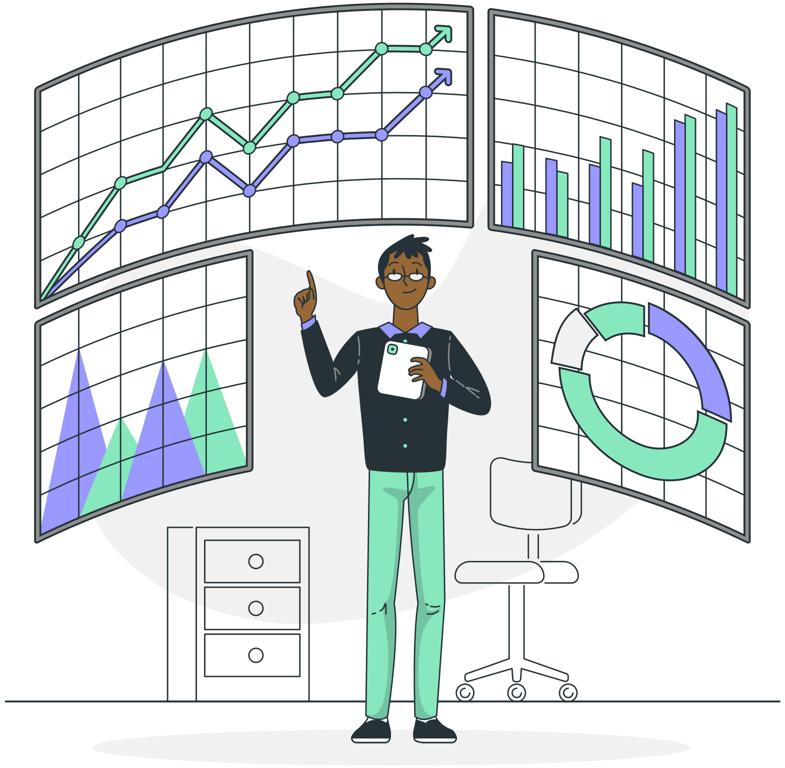
High recall means that there are less people who have mental health problems but are classified as having no problems

We want to catch as much as possible if the people is having problems in the purpose

SO, Recall of XLNet is 98.64%, distilBERT is 92.80% and the Index RNN model is 85.74%.

- XLNet is the best model that we examine to solve this mental health classification problem.

Thanks!



Material that accessed

Cherian, V., & Bindu, M. S. (2017). Heart disease prediction using Naive Bayes algorithm and Laplace Smoothing technique. *Int. J. Comput. Sci. Trends Technol*, *5*(2), 68-73.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv:1901.02860*

Efimov, V. (2023, October 9). Large language models: DistilBERT — smaller, faster, cheaper and lighter. *Medium*. https://towardsdatascience.com/distilbert-11c8810d29fc

Kianyew, N. (2024, January 7). What is Relative Positional Encoding. *Medium*. https://medium.com/@ngiengkianyew/what-is-relative-positional-encoding-7e2fbaa3b510

Pei, C. T. (2021, December 13). DistilBERT — 更小更快的BERT模型 - NLP-Trend-and-Review - Medium. *Medium*. https://medium.com/nlp-tsupei/distilbert-%E6%9B%B4%E5%B0%8F%E6%9B%B4%E5%BF%AB%E7%9A%84bert%E6%A8%A1%E5%9E%8B-eec345d17230

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Tsoi, Y. C., Xiao, H. R., Naive Bayes Classifier. https://course.cse.ust.hk/comp2211/notes/3-naive-bayes-full.pdf

Material that accessed

Xiao, M. (2020, May 5). Understanding Language using XLNet with autoregressive pre-training. *Medium*. https://medium.com/@zxiao2015/understanding-language-using-xlnet-with-autoregressive-pre-training-9c86e5bea443

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *Advances in neural information processing systems*, *32*.