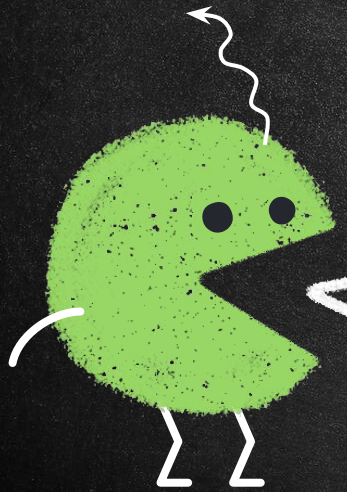
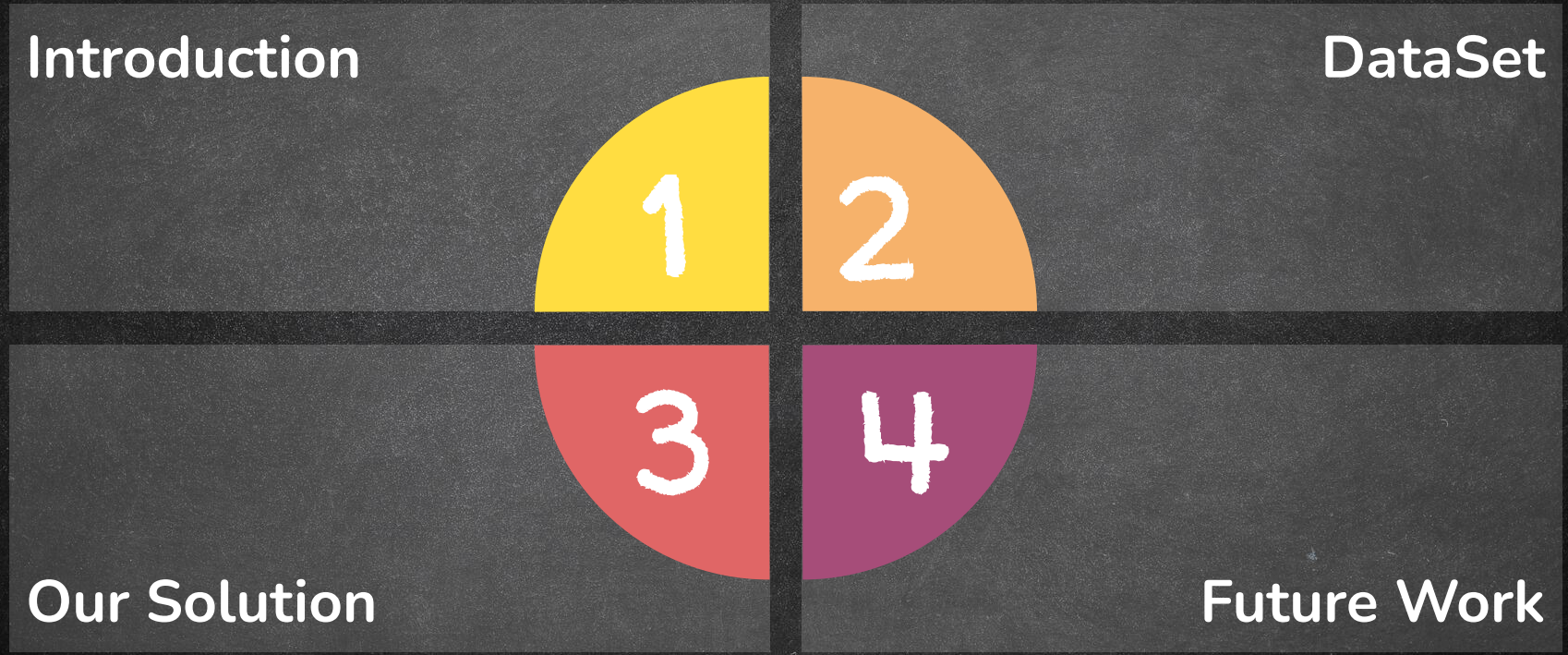
Sentiment Analysis

On short movie reviews

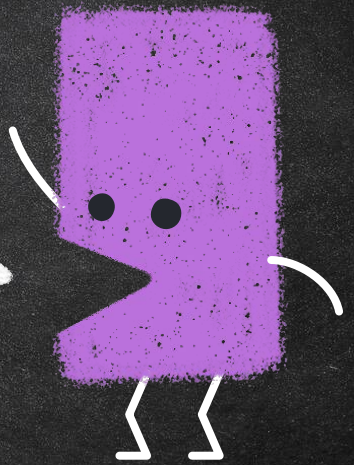


Agenda

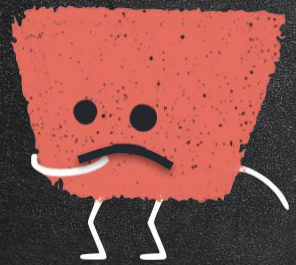


Part 1

The Problem



Sentiment Analysis



Balabalabalabalab
alabalabalabalaba
labalabalabalabal
abalabalabala...
×&%¥#@.....

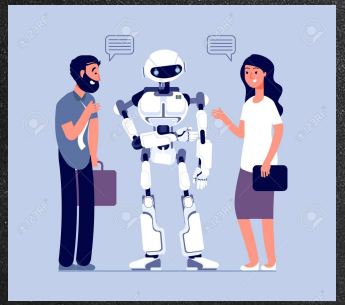
Binary
Classification

Importance ! ! !

1.

Machines' interaction with humans

E.g: AI robot Siri



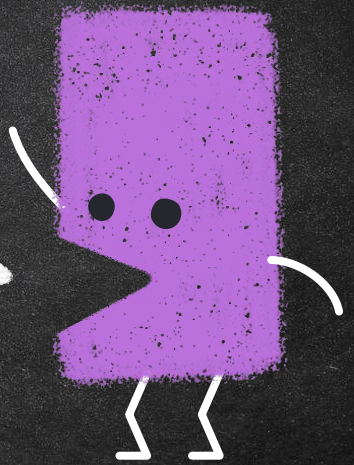
2

Efficiently decide the emotion in natural language

E.g. our dataset

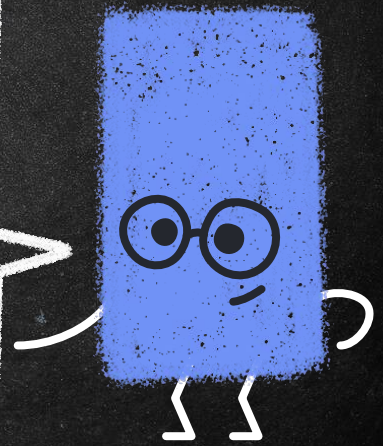
Part 2

The Dataset



Kaggle

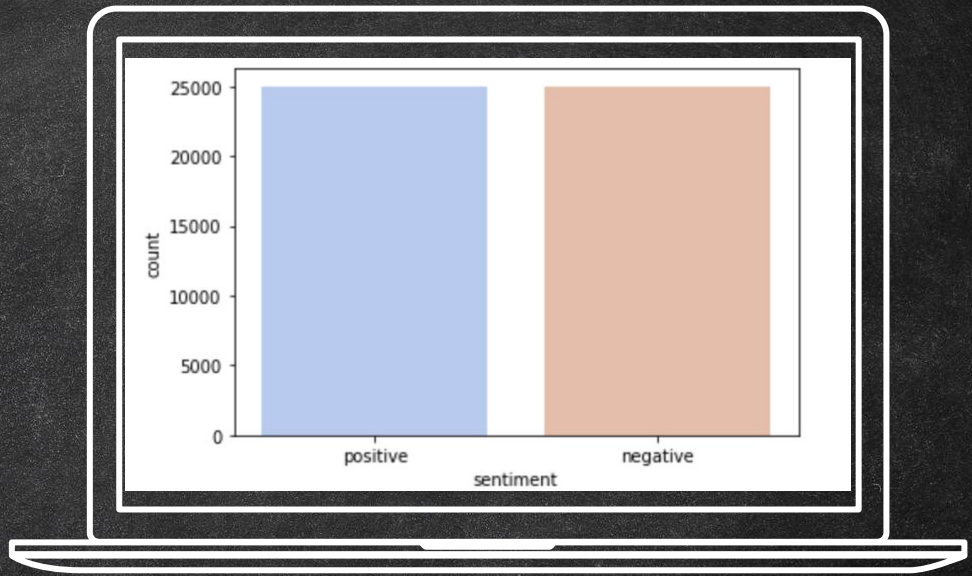

Dataset
IMDB Dataset of 50K Movie Reviews
Large Movie Review Dataset



Dataset

Distribution

Positive:negative =1:1



Dataset

Around 231 Words in each comment

Average sentence count 26

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

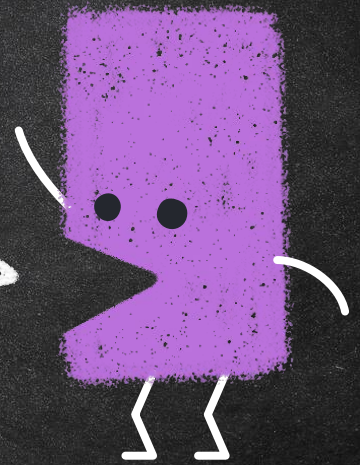
For data preprocessing, we do the basic text preprocessing in the very beginning.
(e.g. Remove stopwords and punctuation)

It turns out that the preprocessing of texts affects the performance largely(will explain later)

“

Part 3

Our Trial



In this part...

1. First few trials
2. Final solution
3. Optimize the model
    a. Change Text preprocessing
    b. Hyperparameters

Sentiment
Analysis
Model

NLP

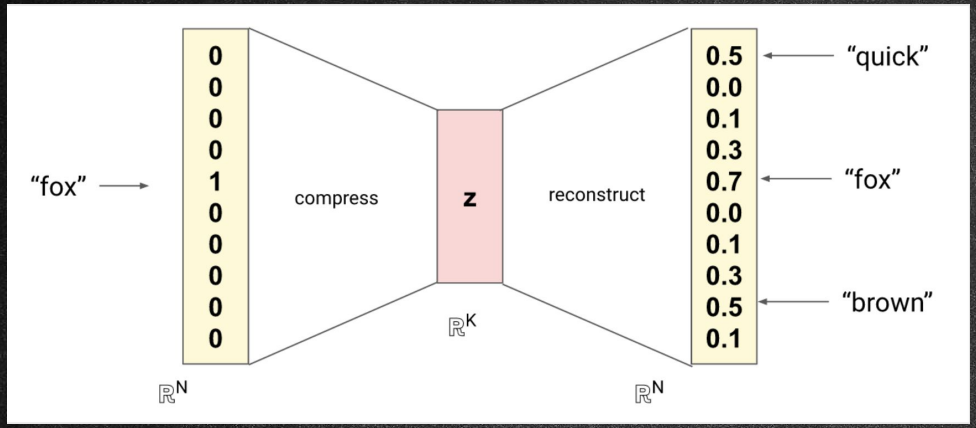
=

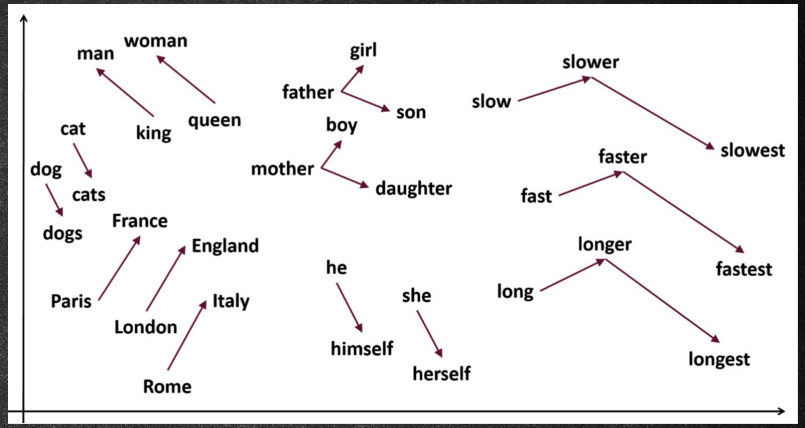
Word Embedding

+

Classifier

First Trial for Word Embedding: Word2Vec





- Convert each word to a **vector** (direction matters)
- If everything goes on correctly:

King - Queen = Man - Woman

Word2Vec + SVM (Gensim)

```
model = Word2Vec(review[:40000], vector_size=500, window=10, min_count=10)
```

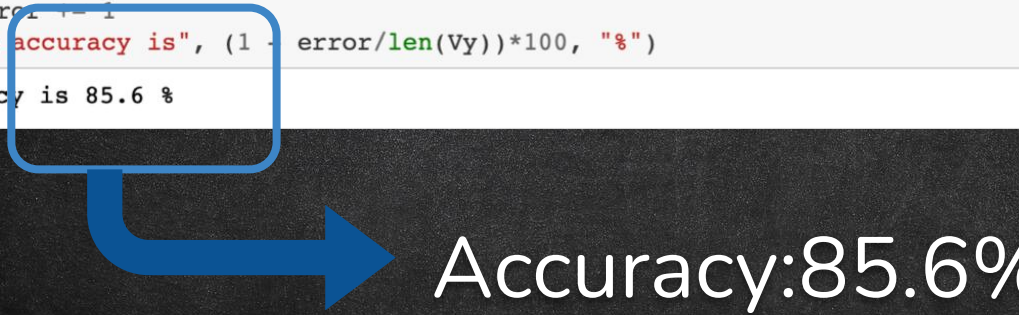
```
clf = svm.SVC()
clf.fit(X, Y)
```

```
SVC()
```

```
VX = val_X.apply(generate_vec).values
Vy = val_y.values
```

```
error = 0
for i in range(len(Vy)):
    if clf.predict([VX[i]]) != Vy[i]:
        error += 1
print("The accuracy is", (1 - error/len(Vy))*100, "%")
```

```
The accuracy is 85.6 %
```



Accuracy:85.6%

However, surprisingly

One-Hot Encoding + NaiveBayes

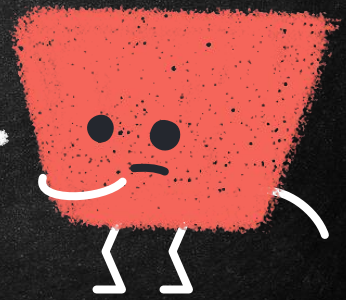
(The exactly the same method that we used in HW3, to identify spam emails)

Gives us a SIMILAR RESULT*

*Accuracy: 84%

Results

Word Embedding	Classifier	Accuracy
W2V	SVM	85.6% (gensim) 84.8% (TF pretrained)
	Random Forest	84.2 %
	FCNN	84.6%
OneHot Encoding	Naive Bayes	84.5%



New Choice: TF-IDF

Abbreviation of :

Term Frequency - Inverse Document Frequency

TFIDF (relative frequency)

$$TF_{w,d_i} = \frac{count(w)}{\sum_{t \in d_i} count(t)}$$

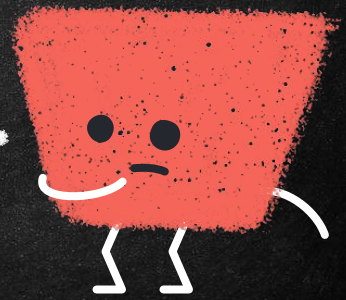
$$IDF_{w,D} = log \frac{|D|}{1 + \sum_{i=1}^{|D|} I(w, d_i)}$$

$$TFIDF_{w,d_i} = TF_{w,d_i} * IDF_{w,D}$$

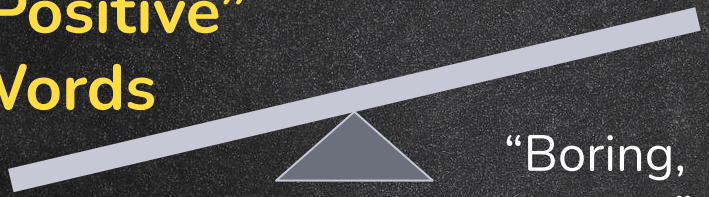
- If a term is frequently occur in one review, but not in the other reviews, its TFIDF value will be high.
- The logic:
  Higher TFIDF => Informative
- In some way, it can be viewed as a enhanced OneHot encoding

Results

Word Embedding	Classifier	Accuracy
TFIDF	**FCNN**	**Around 88.5%**
	Random Forest	83.2%

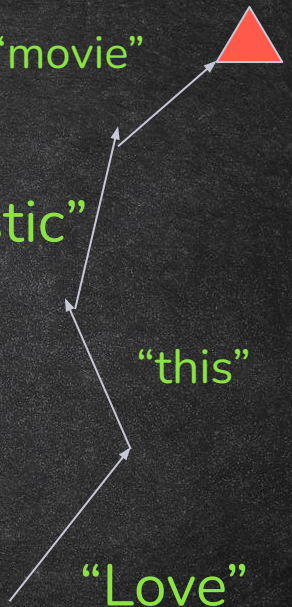


TFIDF



“Love, awesome, fantastic…”

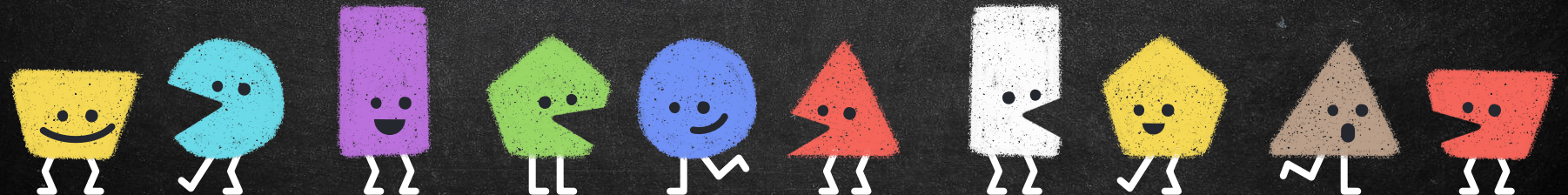
Word2Vec



TFIDF + FCNN

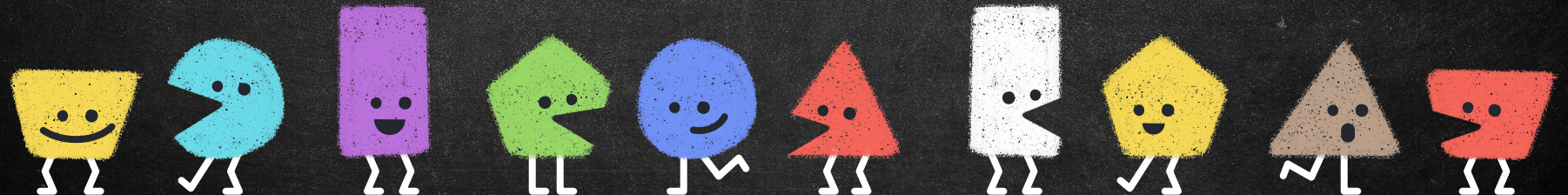
Activation function for output layer: sigmoid

Loss function: Binary Cross entropy loss

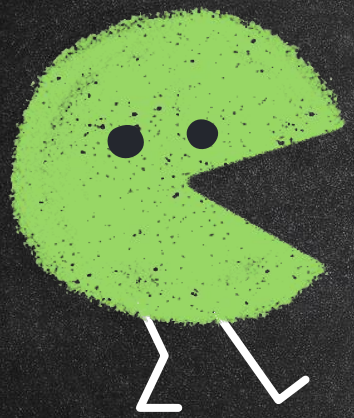


Can we do better?

Optimization of the model



An unexpected discovery!



One day when Eddy was training the model, he **forgot to do the text preprocessing.**

However, the performance of the model is **even better** than the previous model!

Text Preprocessing

1. Removing all numbers
2. Convert all letters to lowercase
3. Removing punctuation
4. Tokenize
5. Remove stopwords
6. Lemmatize

Text Preprocessing

1. Removing all numbers
2. Convert all letters to lowercase
3. Removing punctuation
4. Tokenize
5. Remove stopwords
6. Lemmatize

Improvements

1. Removing all numbers
2. Convert all letters to lowercase(weigh uppercase more)
3. Removing punctuation
4. Tokenize
5. Remove stopwords
6. Lemmatize

E.g. ABSOLUTELY GREAT
Is counted as 2 * absolutely great

Improvements

1. Removing all numbers
2. Convert all letters to lowercase(weigh uppercase more)
3. Removing punctuation(except ! ?)
4. Tokenize → split
5. Remove stopwords
6. Lemmatize

Improvements

1. Removing all numbers
2. Convert all letters to lowercase(weigh uppercase more)
3. Removing punctuation(except ! ?)
4. Tokenize
5. Remove stopwords
6. Lemmatize

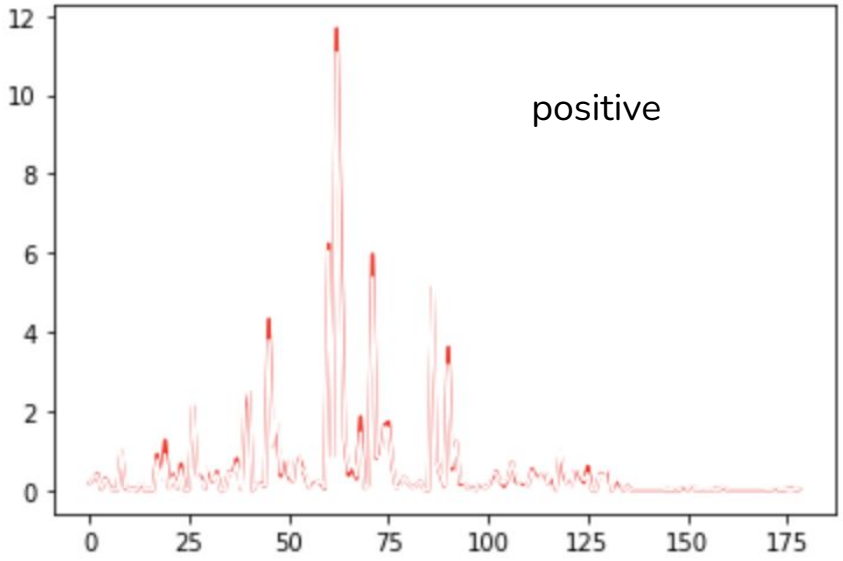
In [1]:
```
from nltk.corpus import stopwords
```

In [2]:
```
print(stopwords.words("english"))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "
it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', '
having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while
', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then'
, 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'oth
er', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will
', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren
't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'should
n', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

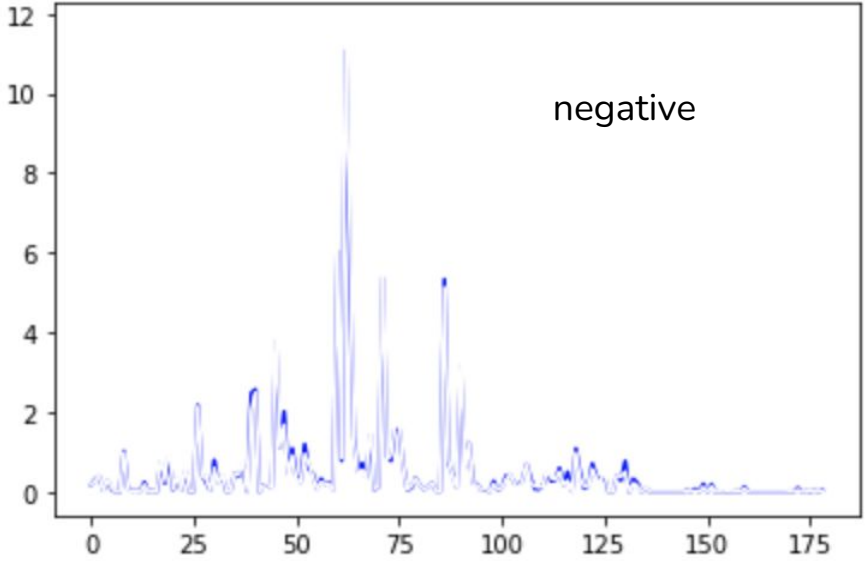
STOPWORDS ?

Average occurrence



index of stopwords

Average occurrence



index of stopwords

Improvements

1. Removing all numbers
2. Convert all letters to lowercase(weigh uppercase more)
3. Removing punctuation(except ! ?)
4. Tokenize
   ~~Remove stopwords~~
5. Lemmatize

```
-> rocks : rock
-> corpora : corpus
-> better : good
```

Other Problem in Wordlist

im
you
Ll

- Expand slangs and contractions

a
ahh
ahhhhhh
Ahhhhhhhh

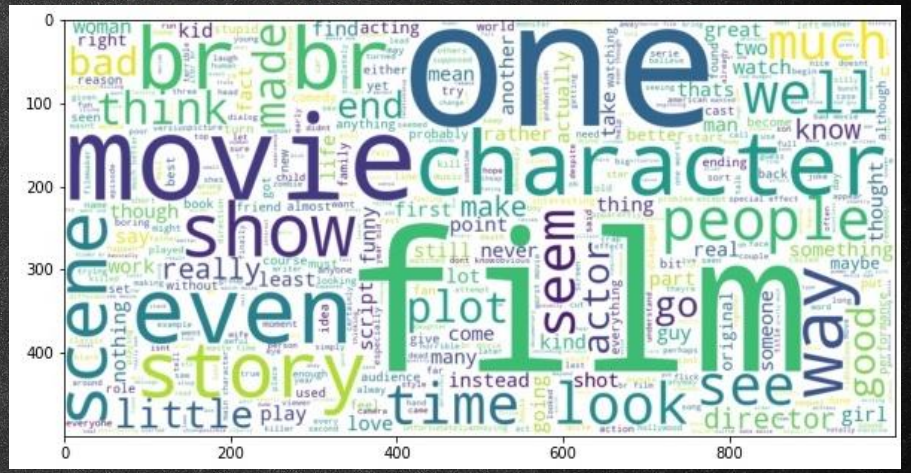
E.G. “Hahaha” = 3* “ha”

haha
hahahaha
hahaaaahahahaha

- Reweigh long interjection

WordCloud

positive



negative

Improvements

- Removing all numbers
- Convert all letters to lowercase(weigh uppercase more)
- Expand slangs and contractions
- Reweigh long interjection
- Removing punctuation(except ! ?)
- Tokenize
- ~~Remove stopwords~~
- Lemmatize
- Remove high occurrence words

Other hyper-parameters

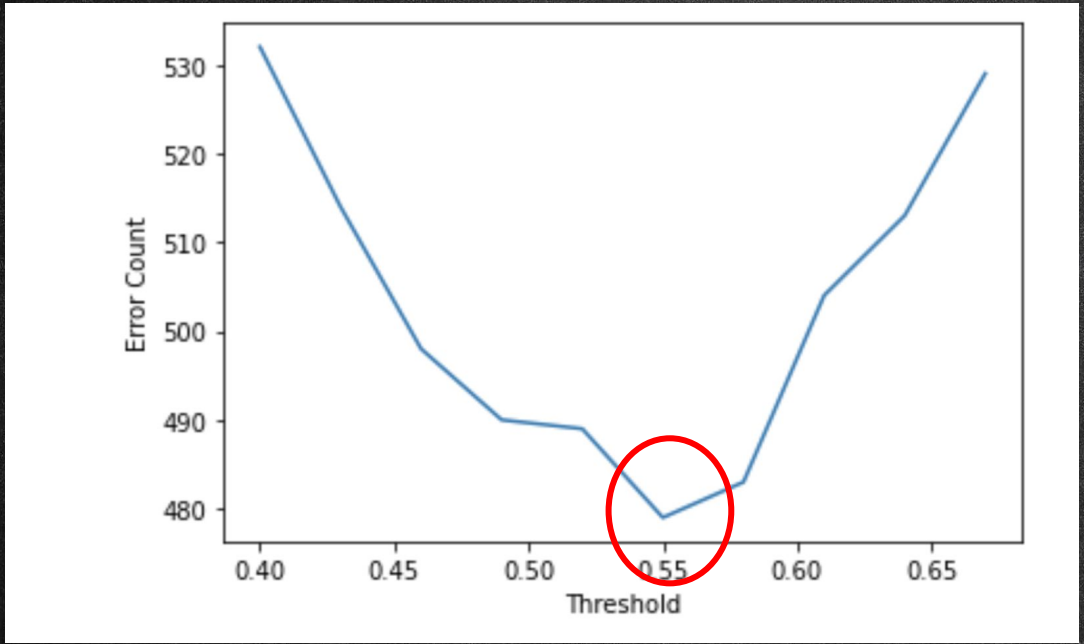
- The structure of the neural network and its hyperparameters (including dropout rate, regularization)
- The threshold for classification

After trying some structures, we finally get a low-cost FCNN

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, 31690)             0
_________________________________________________________________
dense (Dense)                (None, 32)                1014112
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,014,145
Trainable params: 1,014,145
Non-trainable params: 0
_________________________________________________________________
```

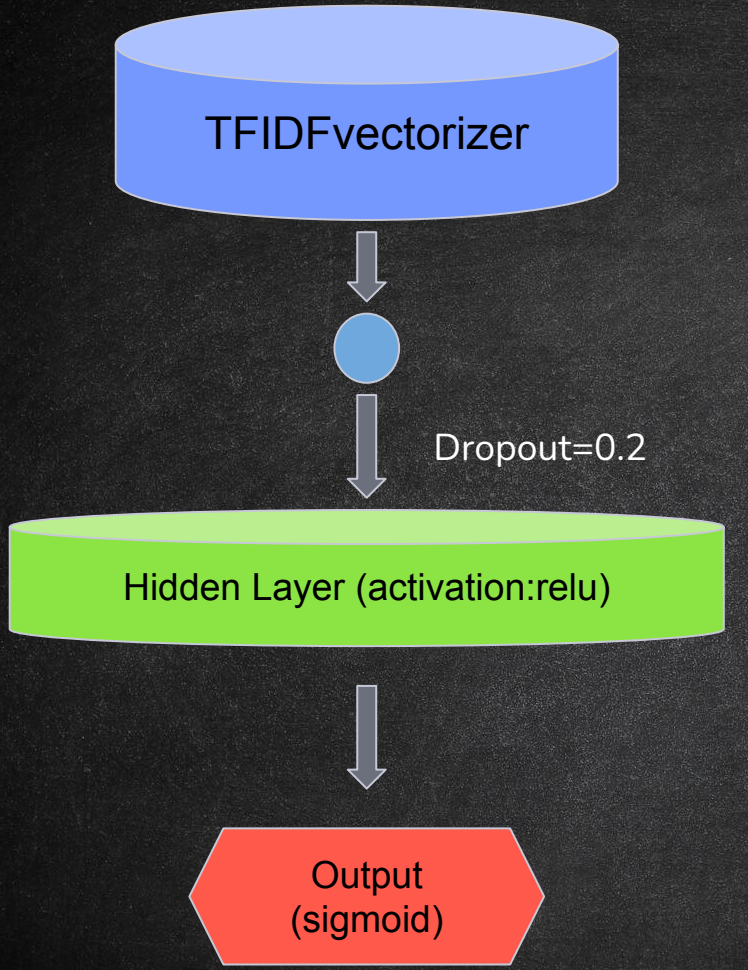
Dropout rate = 0.2
GD: ADAM
Batch Size = 32

Tune the threshold on validation set



Validation size:
5000

Optimal Threshold:
0.55



Max_Df=0.95, no stop words removing

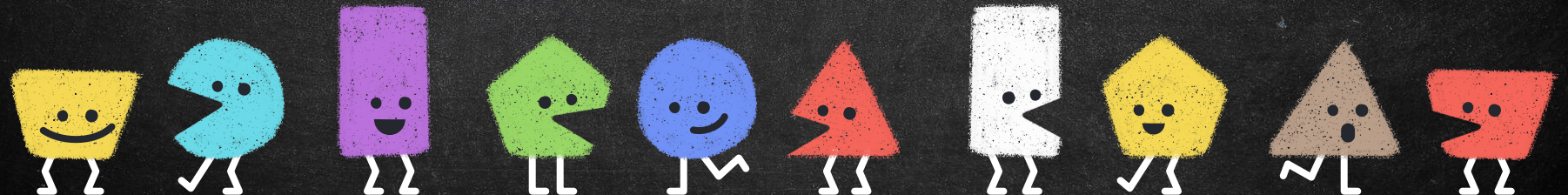
Vector dimension around 30000

32 nodes in the hidden layer

With threshold=0.55,

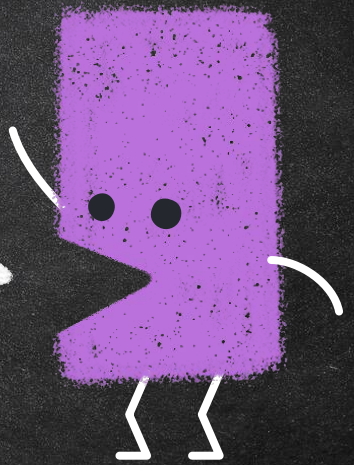
90.8%

Whoa! That's a big number, aren't you proud of us?



Part 4

Future Work



If we have infinite time...

- FCNN with Better performance
- TFIDF loses the information of context, we want to take that into consideration
- We want to make our Word Embedding process trainable

Thank you for your listening