Sentiment Analysis Online Movie Reviews

AI Applications

AI Applications Student | NLP Graded Assignment

Honor Pledge for Graded Assignments

"I affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

Signature:

Summary

- Data Pre-Processing
- Model Creation
- Prediction Results
- Difficulties Encountered & Learning Points

Data Pre-Processing

- Process Text
- Perform Tokenization

```
from nltk.tokenize import sent_tokenize, word_tokenize
```

- Remove Punctuation

```
stop = nltk.corpus.stopwords.words('english')
```

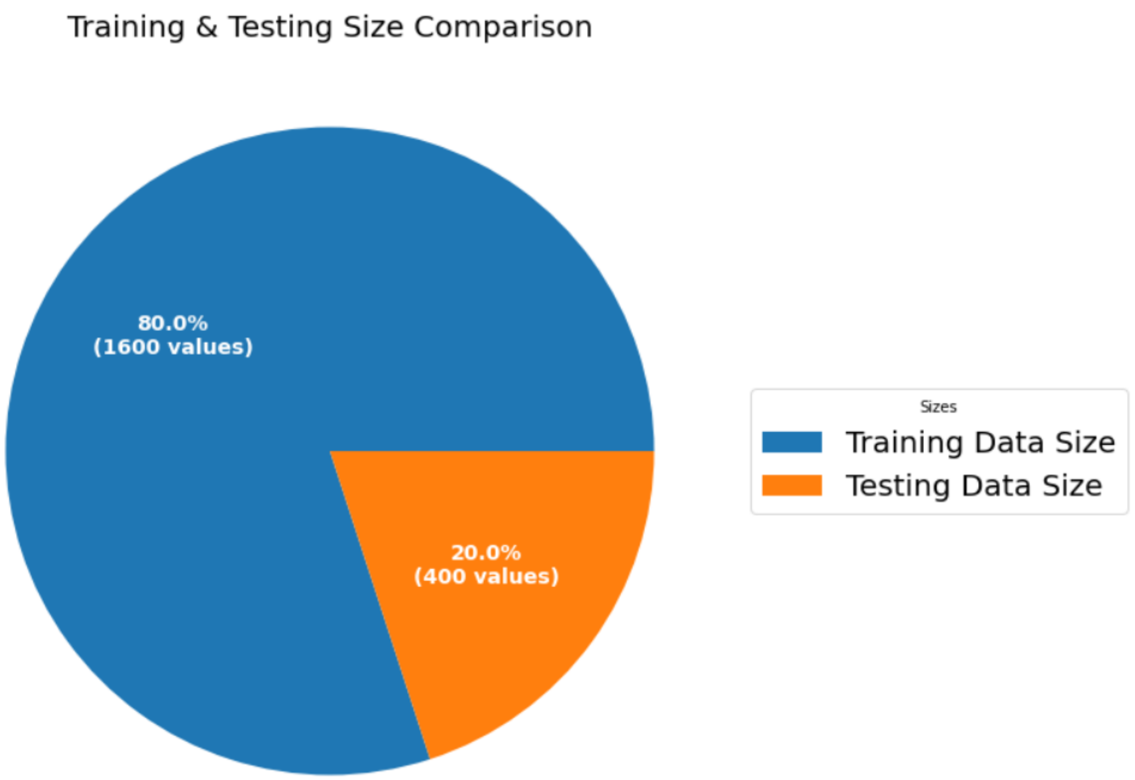
- Lemmatize

```
from nltk.stem import WordNetLemmatizer
```

Data Pre-Processing

- Train Test Split
- 80% Training
- 20% Testing

```
from sklearn.model_selection import train_test_split
```



Data Pre-Processing

- Vectorization
- TF-IDF

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

- TfidVectorizer
- Build TF-IDF function

Term Frequency: TF of a term or word is the number of times the term appears in a document compared to the total number of words in the document.

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

Inverse Document Frequency: IDF of a term reflects the proportion of documents in the corpus that contain the term. Words unique to a small percentage of documents (e.g., technical jargon terms) receive higher importance values than words common across all documents (e.g., a, the, and).

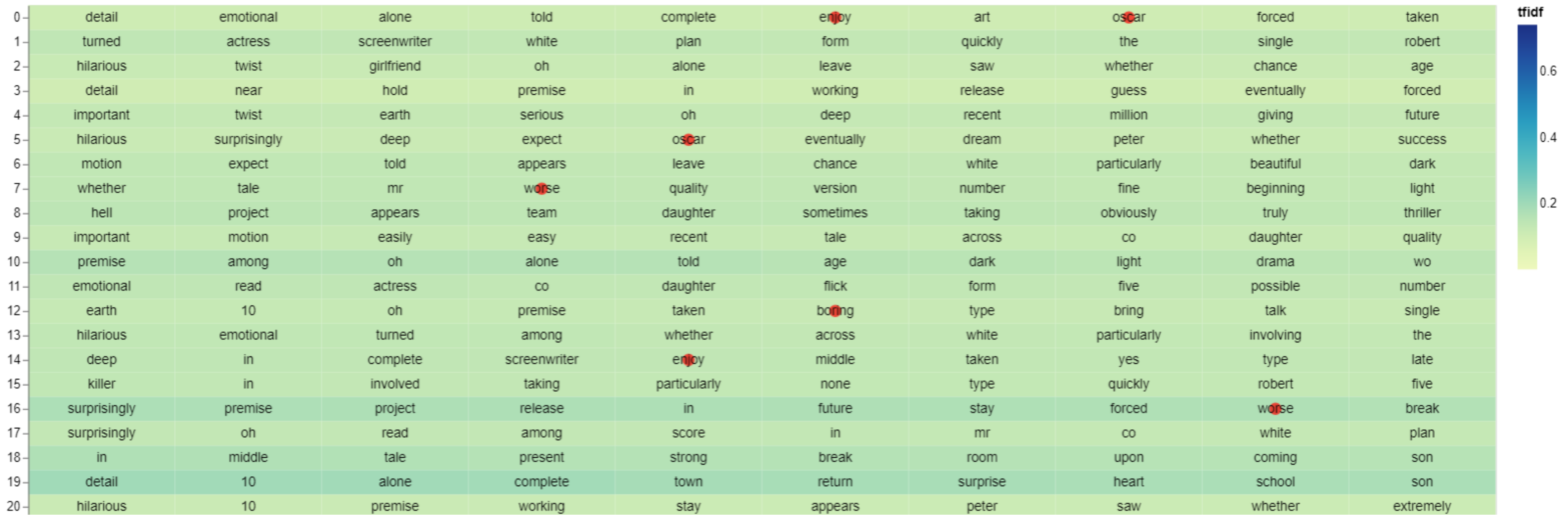
$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

The TF-IDF of a term is calculated by multiplying TF and IDF scores.

$$TF\text{-}IDF = TF * IDF$$

Data Pre-Processing

- Visualization using Altair
- Heatmap



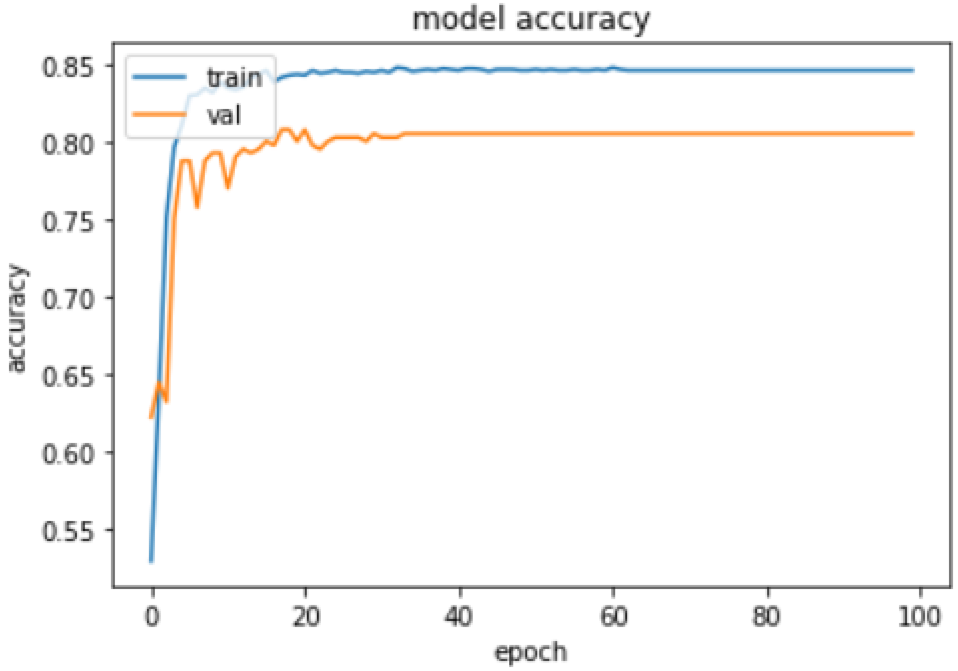
Data Pre-Processing

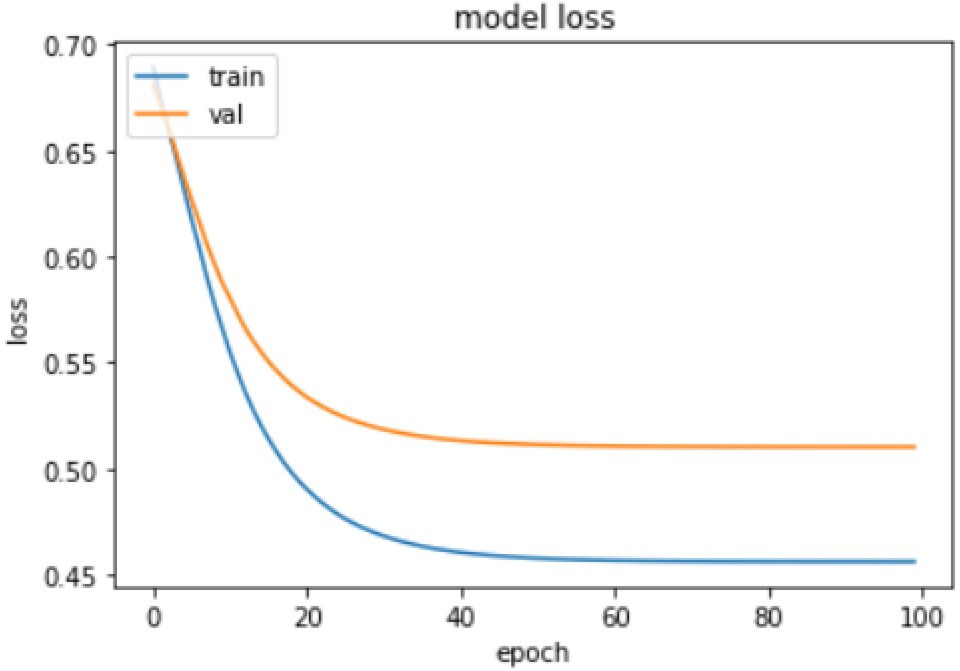
- TF_IDF Concentration Clusters
- Scatter Plots
- Majority of terms appeared in the range of 0.1 to 0.2, rarely above 0.3. Total range is 0 to 1.
- This shows that there are a lot of repetitive terms that appeared in multiple documents.

Model Creation

- Neural Network
- 2 Dense Layers
- Optimizers = Adam Optimizer
- Loss = binary_crossentropy
- Metrics = accuracy
- Callbacks = [

EarlyStopping,
ModelCheckpoint,
LearningRateScheduler
]





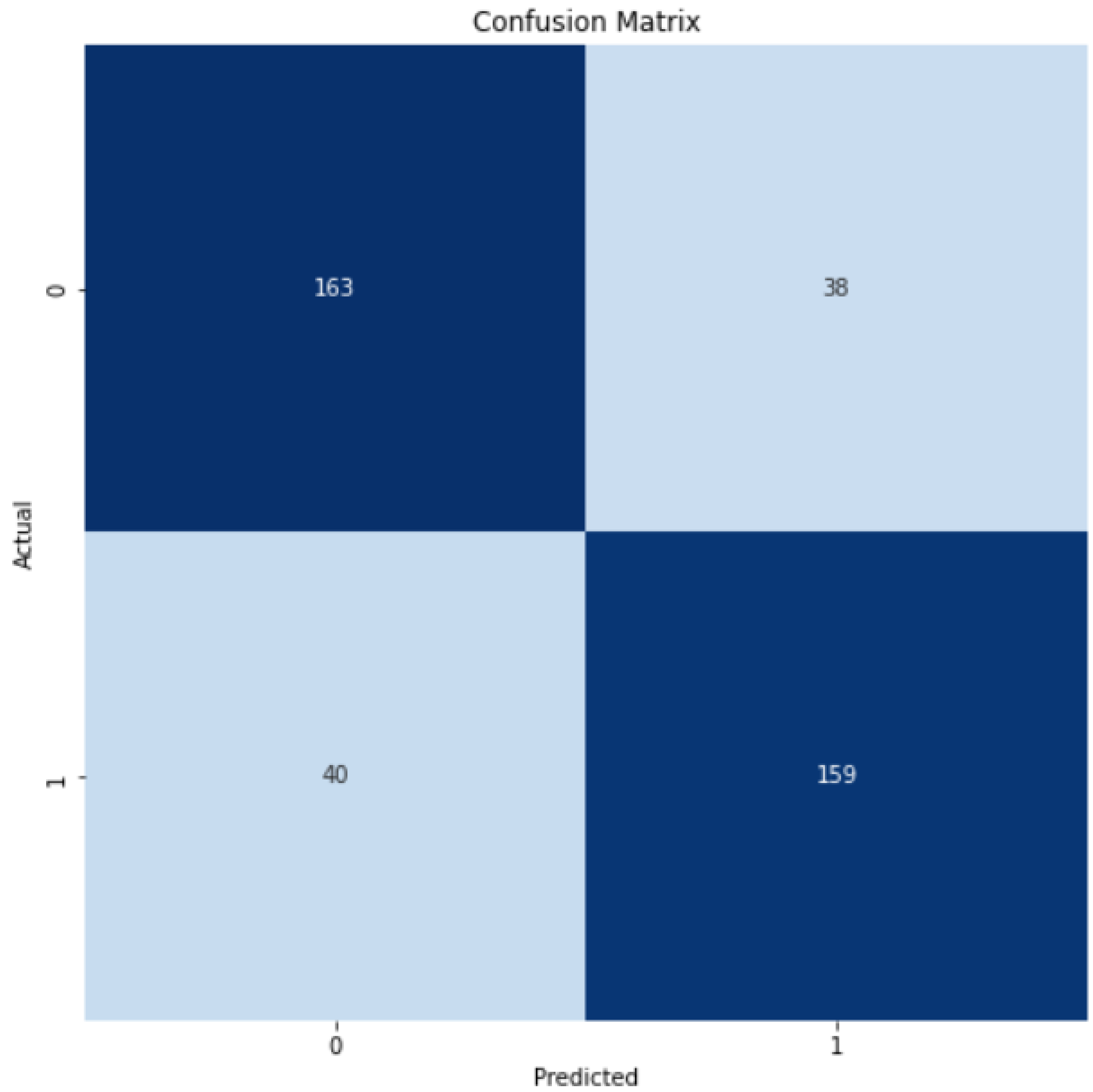
Prediction Results

- Evaluation acc = 80%
- Precision label 0 = 0.80
- Precision label 1 = 0.81

Classification Report:

---

	precision	recall	f1-score	support
0	0.80	0.81	0.81	201
1	0.81	0.80	0.80	199
accuracy			0.81	400
macro avg	0.81	0.80	0.80	400
weighted avg	0.81	0.81	0.80	400



Prediction Results

- <u>**Test for Unseen Text**</u>

Positive text

"This is going to go down as one of 2022's most entertaining motion pictures"

```
100%|██████████| 100/100 [00:01<00:00, 97.30it/s]
Predicted Sentiment: POSITIVE
```

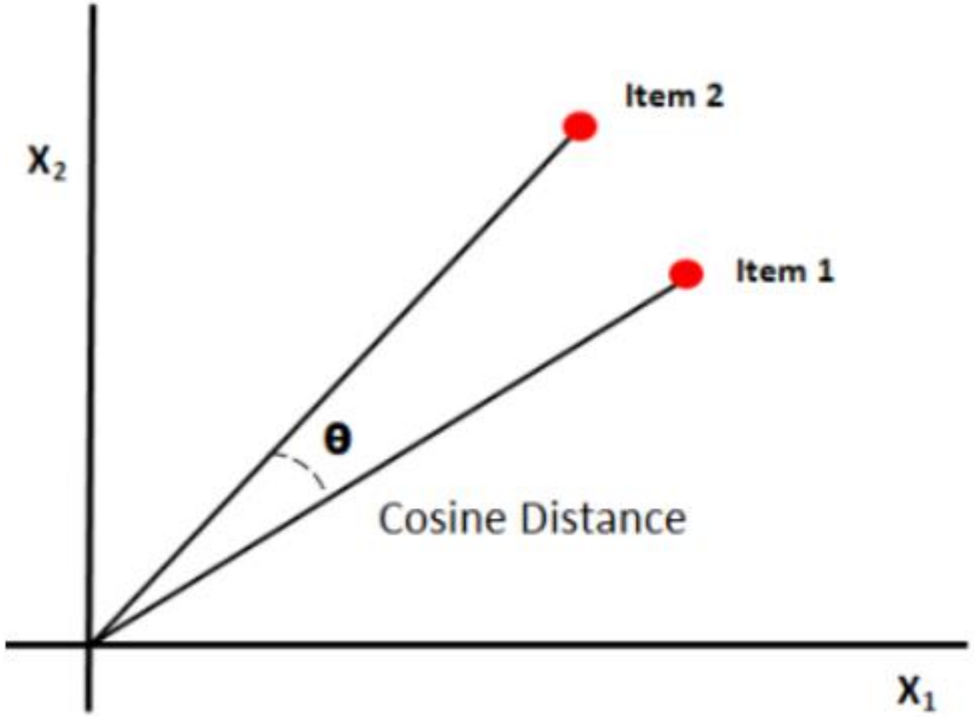
Negative text

"Just when you think you've seen the worst movie ever made, along comes this pile of toxic waste."

```
100%|██████████| 100/100 [00:01<00:00, 97.20it/s]
Predicted Sentiment: NEGATIVE
```

Prediction Results

- **Apply Cosine Similarity to Find Similar Text**



- Compute cosine similarity for each word in unseen texts against each word in training dataset.
- Sort values from descending order and display the relevant word for the top cosine similarity values.

Prediction Results

- **Apply Cosine Similarity to Find Similar Text**

```
from sklearn.metrics.pairwise import cosine_similarity
```

Similar words for positive text:

	Similar words	Cosine Similarity
0	several	0.26121273222675995
1	nice	0.17099611612487206
2	beginning	0.16436803502996014
3	mean	0.16434677982006513
4	show	0.16399003084068903
5	taken	0.16192629754923127

Similar words for negative text:

	Similar words	Cosine Similarity
0	come	0.2535885476337849
1	turn	0.22630730810113545
2	screenwriter	0.2120569879127968
3	think	0.2087952832181084
4	violence	0.19622243100038428
5	involving	0.18709544095036856

Difficulties Encountered & Learning Points

Dr Chang | PhD, Computational Biology, Bioinformatics

"Friend of mine, Dr Chang, gave advices on how to build the **tfidfvectorizer.** He corrected my mistake specifically on how I used the **fit_transform** method. I had to be careful when building the vectorizer. Should only ***fit_transform*** on the ***X_train*** then use transform on ***X_test***. The reason for not using fit_transform on ***X_test*** is because ***fit_transform*** chooses the best words you provide. So even though you may have equal amount of vocabs in both sets, using ***fit_transform*** may result in the mis-alignment in the arrays(because the vocabs are different). It would render your validation set useless because your model is validating against nonsense. He also used an analogy of describing the TF-IDF function like a mother function, it gave birth to the vectorizer, then you can use it subsequently."

End of Presentation