

Sentiment Analysis of Social Media's Messages

Deep Learning Final Project

Nikita Breslavsky - 332363498

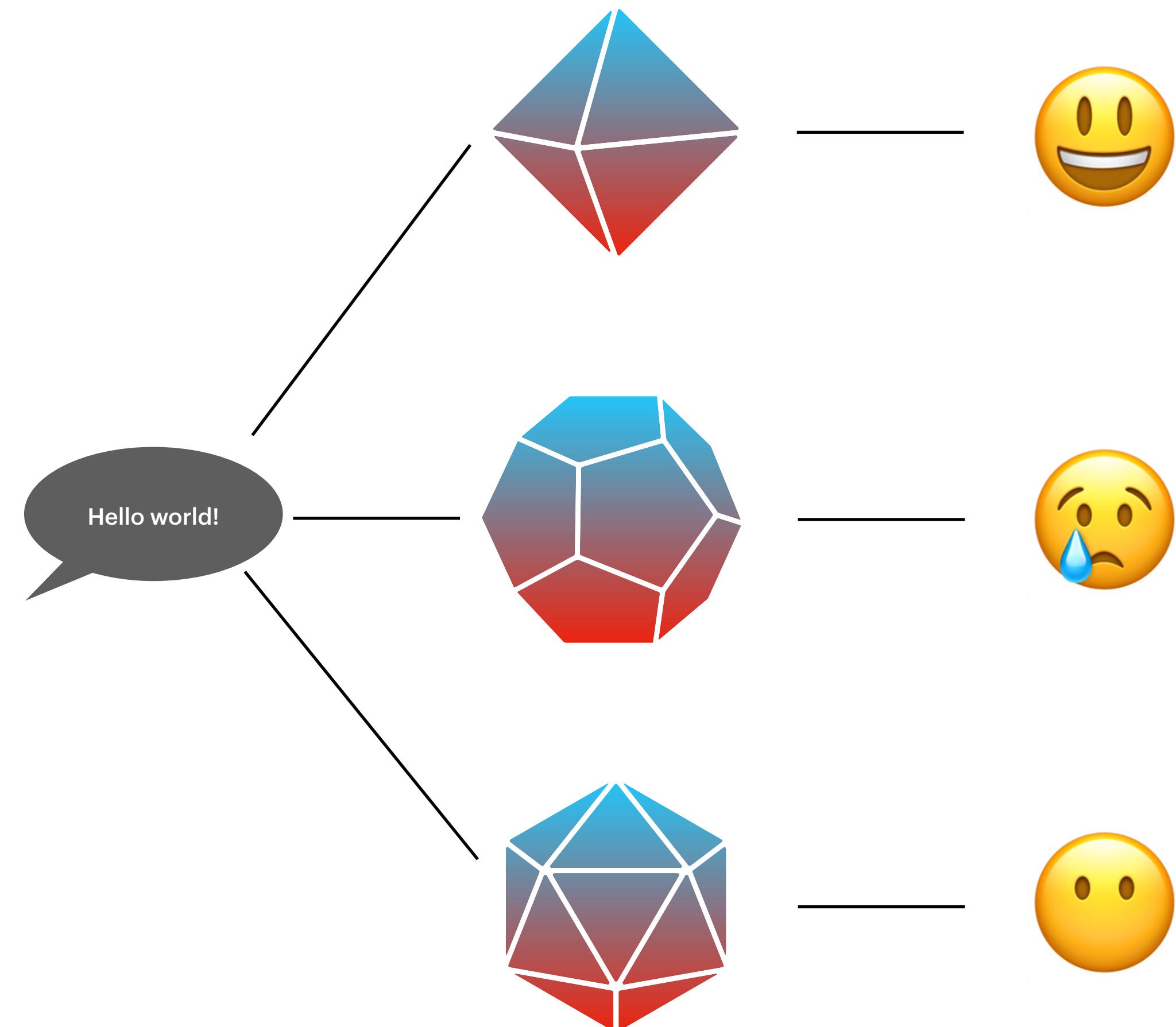
Barak Finkel - 206491714

Gilad Fisher - 318471109

Introduction

Project's goals

- Sentiment classification model
- Comparative analysis approaches



How can we use it?

- Enhancing business insights
- Policy making and social research
- Content moderation and cyberbullying detection



Related Work

YouTube course

By Kylie Ying

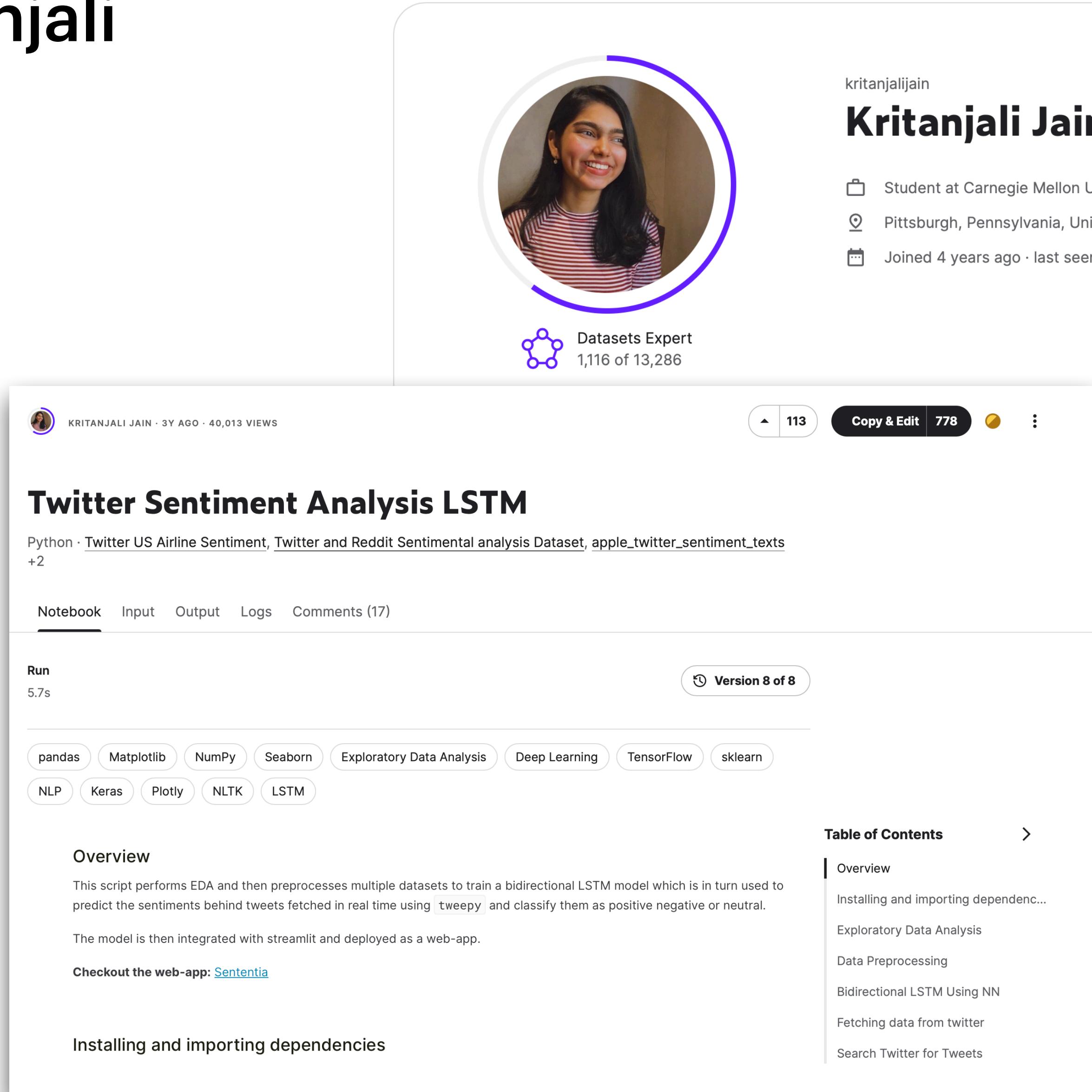
- Practical knowledge
- Data preprocessing
- Model building



Related Notebook Research

By Jain Kritanjali

- Noise reduction
- Text vectorizing
- LSTM

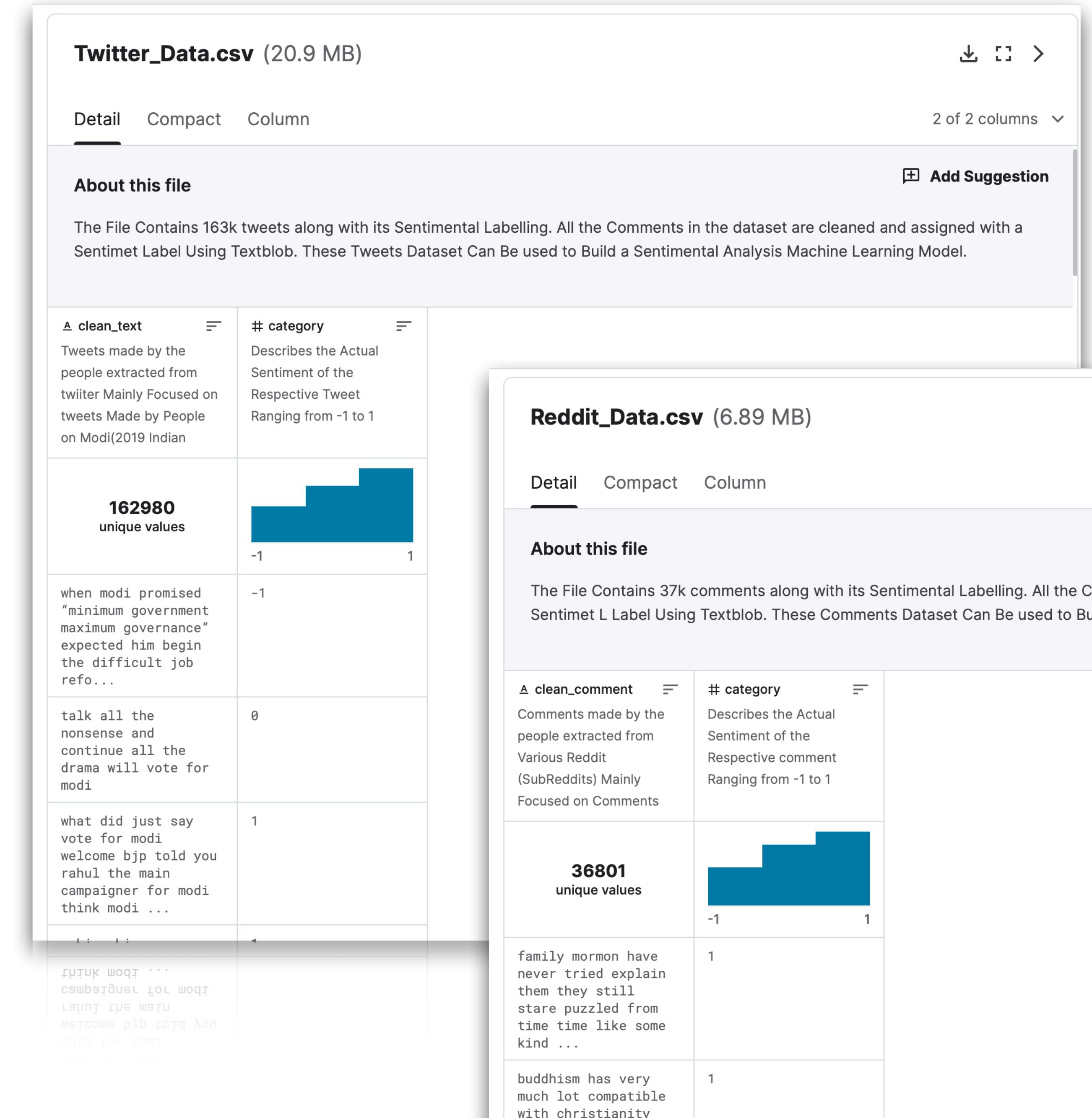


The screenshot shows a Jupyter Notebook profile page for a user named Kritanjali Jain. At the top right, there is a circular profile picture of a young woman with long dark hair, smiling. Below the profile picture, the username "kritanjali Jain" is displayed, followed by the name "Kritanjali Jain" in a bold, black font. To the right of the name, there are three small icons: a briefcase (Student at Carnegie Mellon University), a location pin (Pittsburgh, Pennsylvania, United States), and a calendar (Joined 4 years ago · last seen in the past day). Below these details, there is a purple icon of a neural network and the text "Datasets Expert 1,116 of 13,286". The main content area of the page displays a notebook titled "Twitter Sentiment Analysis LSTM". The notebook was created by Kritanjali Jain 3 years ago and has 40,013 views. It has 113 cells and 778 commits. The "Notebook" tab is selected. Below the title, the code cells are visible, showing imports like pandas, NumPy, Seaborn, Exploratory Data Analysis, Deep Learning, TensorFlow, sklearn, NLP, Keras, Plotly, NLTK, and LSTM. The first cell is titled "Run" and has a duration of 5.7s. A "Version 8 of 8" badge is present. On the right side of the notebook view, there is a "Table of Contents" sidebar with links to various sections of the notebook, including Overview, Installing and importing dependencies, Exploratory Data Analysis, Data Preprocessing, Bidirectional LSTM Using NN, Fetching data from twitter, and Search Twitter for Tweets.

Final Model Overview

Dataset

- Combined dataset of Twitter and Reddit posts
- With 200,000 posts



Preprocessing

- Noise reduction
- Data splitting
- No vectorization

```
1 pattern = re.compile(r"[^a-zA-Z0-9]")
2 def post_to_words(post):
3     """ Convert tweet text into a sequence of words """
4     # convert to lower case
5     text = post.lower()
6     # remove non letters
7     text = re.sub(pattern, " ", text)
8     # tokenize
9     words = text.split()
10    # remove stopwords
11    words = [w for w in words if w not in stopwords.words("english")]
12    # apply stemming
13    words = [PorterStemmer().stem(w) for w in words]
14    # return list
15    return words
16
17 print("\nOriginal tweet ->", df['clean_text'][0])
18 print("\nProcessed tweet ->", post_to_words(df['clean_text'][0]))
```

Executed at 2024.03.01 13:10:37 in 9ms

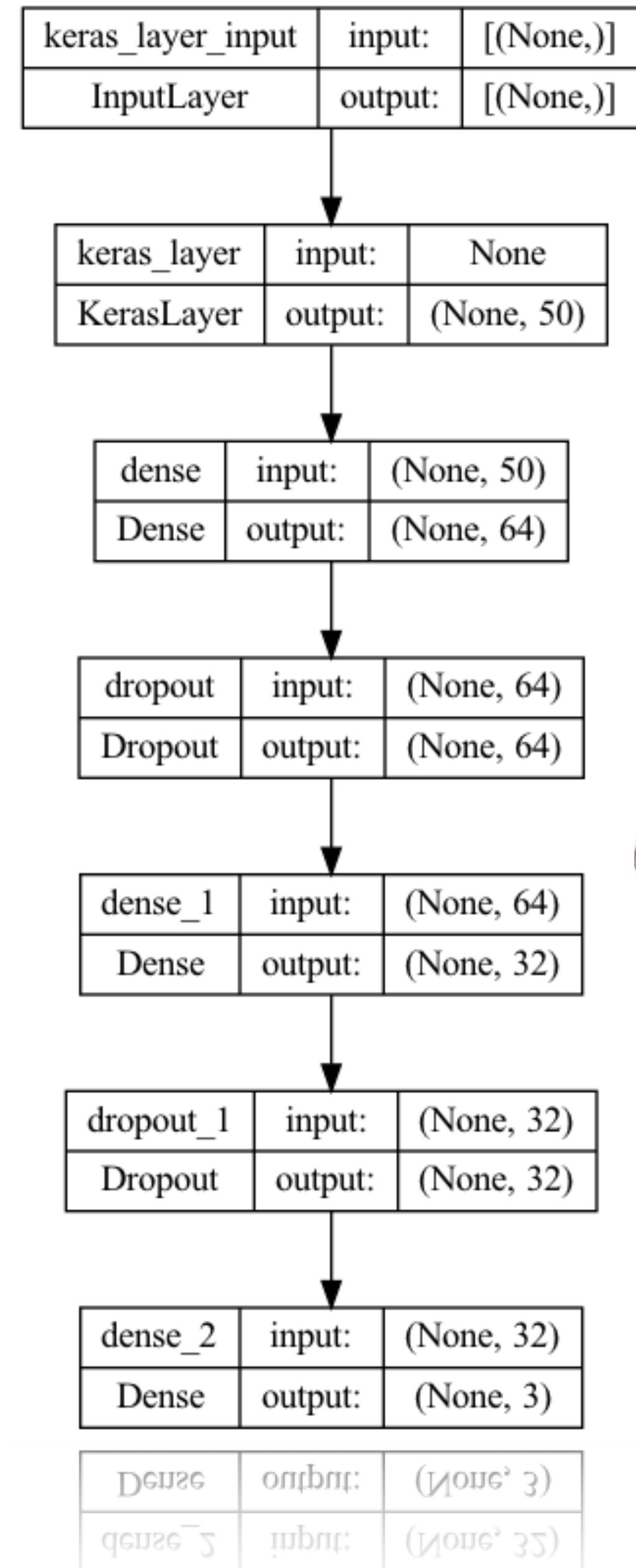
Original tweet -> when modi promised "minimum government maximum governance"
expected him begin the difficult job reforming the state why does take years get justice state
should and not business and should exit psus and temples

Processed tweet -> ['modi', 'promis', 'minimum', 'govern', 'maximum', 'govern', 'expect',
'begin', 'difficult', 'job', 'reform', 'state', 'take', 'year', 'get', 'justic', 'state', 'busi', 'exit', 'psu',
'templ']

[{'text': 'when modi promised "minimum government maximum governance"\nexpected him begin the difficult job reforming the state why does take years get justice state\nshould and not business and should exit psus and temples', 'tokens': ['modi', 'promis', 'minimum', 'govern', 'maximum', 'govern', 'expect', 'begin', 'difficult', 'job', 'reform', 'state', 'take', 'year', 'get', 'justic', 'state', 'busi', 'exit', 'psu', 'templ']}]

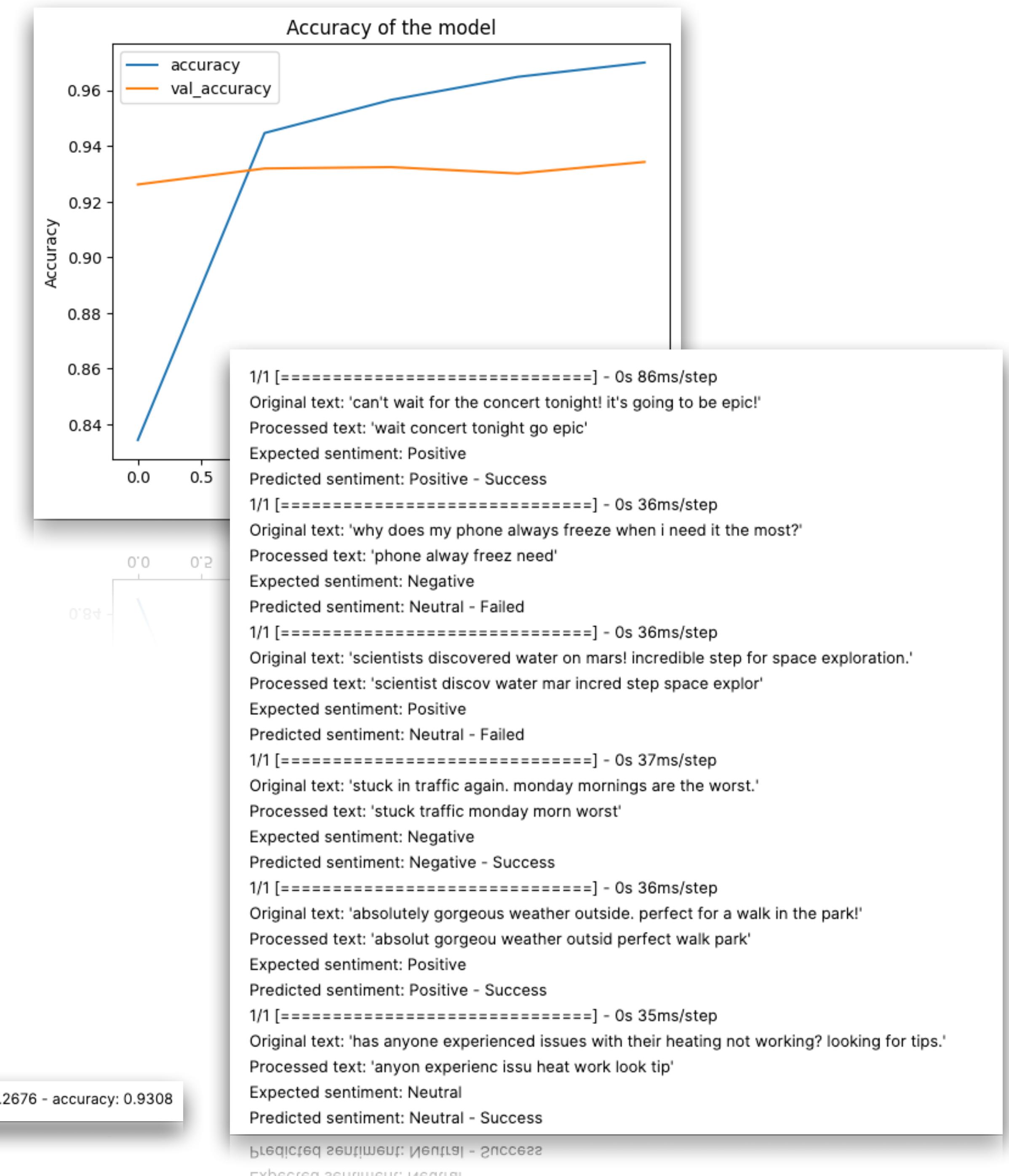
Model

- Google's embedding layer
- Dense layers
- Dropout layers
- Softmax layer
- Adam optimizer
- Categorical Crossentropy



Simulation Results

- 2 minutes fit
- Less than a half second test data evaluation
- 93% accuracy on test data
- Manual testing



Other attempts

Initial

- Data exploration
 - Simple model (nnlm)
 - 80-90% accuracy without preprocessing
 - Future improvements

Model

```
In 13 1 embedding = "https://www.kaggle.com/models/google/nnlm/frameworks/TensorFlow2/variations/en-dim50/versions/1" # Load the  
      embedding model  
2 hub_layer = hub.KerasLayer(embedding, input_shape=[], dtype=tf.string, trainable=True) # Create a keras layer with the embedding model
```

```
In 14 1 model = tf.keras.Sequential() # Create a sequential model
2 model.add(hub_layer) # Add the pre-trained layer
3
4 # Assuming the output of hub_layer is a flat vector. This is just a co
5 # In practice, CNNs need sequence or matrix inputs.
6 #model.add(tf.keras.layers.Reshape((50, 1))) # Reshape for CNN,
7 #model.add(tf.keras.layers.Conv1D(filters=64, kernel_size=3, activi
8 #model.add(tf.keras.layers.GlobalMaxPooling1D()) # Pooling layer
9
10
11 model.add(tf.keras.layers.Dense(16, activation='relu',kernel_regula
    neurons
12 model.add(tf.keras.layers.Dense(3, activation='softmax'))
```

Executed at 2024.03.01 17:31:44 in 224ms

```
In 15 1 model.compile(  
2     optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001),  
3     loss = tf.keras.losses.SparseCategoricalCrossentropy(), # loss function  
4     metrics = ['accuracy'])  
5 )
```

Executed at 2024.03.01 17:31:44 in 6ms

twitter_training.csv (10.33 MB)

Detail Compact Column

About this file

Training data

# 2401	Tweet ID	Borderlands	Positive sentiment	im getting on bor...
1	13.2k	Microsoft TomClancysRainbo... Other (69881)	Negative 3% Positive 3% Other (94%)	30% 28% 42%
2401		Borderlands	Positive	I am coming to the borders and I will kill you all,
2401		Borderlands	Positive	im getting on borderlands and i will kill you all,
2401		Borderlands	Positive	im coming on borderlands and i will murder you all,
2401		Borderlands	Positive	im getting on borderlands 2 and i will murder you me all,
2401		Borderlands	Positive	im getting into borderlands and i can murder you all,
2402		Borderlands	Positive	So I spent a few hours making something for fun. . . If you don't know I am a HUGE @Borderlands fan ...

Logistic Regression

- Noise reduction
- TF-IDF vectorizing
- 85% accuracy
- Very fast!!!
- 17sec to fit
- 35ms to evaluate test data

```
9 1 y = df['category'] # Target
2 X = [''.join(post_to_words(text)) for text in df['clean_text']]
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
Executed at 2024.02.29 17:56:51 in 3m 15s 751ms
```

```
10 1 vectorizer = TfidfVectorizer()
2 X_train_tfidf = vectorizer.fit_transform(X_train)
3 X_test_tfidf = vectorizer.transform(X_test)
Executed at 2024.02.29 17:57:00 in 1s 478ms
```

```
11 1 model = LogisticRegression(max_iter=1000) # Increasing max_iter for convergence
2 model.fit(X_train_tfidf, y_train)
Executed at 2024.02.29 17:57:19 in 17s 835ms
```

```
12 1 LogisticRegression
LogisticRegression(max_iter=1000)
```

```
13 1 # 3. Prediction and Evaluation
2 y_pred = model.predict(X_test_tfidf)
3 accuracy = accuracy_score(y_test, y_pred)
4 print(f'Accuracy: {accuracy * 100:.2f}%')
5
6 # Detailed performance report
7 print(classification_report(y_test, y_pred))
Executed at 2024.02.29 17:57:22 in 35ms
```

```
14 Accuracy: 85.21%
precision    recall   f1-score   support
```

0	0.82	0.73	0.77	4288
1	0.83	0.92	0.87	6847
2	0.89	0.86	0.87	8877

accuracy	0.85	20012		
macro avg	0.85	0.84	0.84	20012
weighted avg	0.85	0.85	0.85	20012

precision	0.82	0.82	0.82	20005
recall	0.82	0.82	0.82	20005
f1-score	0.82	0.82	0.82	20005
support	20005	20005	20005	20005

LSTM

- No noise reduction
- Simple tf encoder and vectorization
- Non-optimized parameters
- Very slow!!!
- 20 min to fit
- Promising accuracy of 89%

```
# LSTM MODEL
encoder = tf.keras.layers.TextVectorization(max_tokens=2000)
encoder.adapt(train_data.map(lambda text, label: text))
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
model2 = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=32,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(3, activation='softmax')
])
model2.compile(
    optimizer =
tf.keras.optimizers.legacy.Adam(learning_rate=0.001), # gradient
descent algorithm
    loss = tf.keras.losses.SparseCategoricalCrossentropy(), # loss
function
    metrics = ['accuracy']
)

metrics = ['accuracy']
EPOCHS = 20
TBSZ = 32 * XTEST.TBSZ * 32 * TBSZ * 32 * TBSZ * 32 * TBSZ
```



LSTM improved

- Noise reduction
- Vocabulary based vectorizer
- Optimized model with CNN filters and RNN bidirectional LSTM layer
- Improved speed 6 minutes to fit
- 92% accuracy on test data

```
vocab_size = 5000
embedding_size = 32
epochs=15
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.9

model2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size,
    tf.keras.layers.Conv1D(filters=32,kernel_size= 3,
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(3, activation='softmax')
])
Executed at 2024.02.29 18:37:34 in 201ms
```

```
optimizer = tf.keras.optimizers.legacy.SGD(learning_rate=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
adamOptimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)
model2.compile(
    optimizer = optimizer, # gradient descent algorithm
    loss = tf.keras.losses.SparseCategoricalCrossentropy(), # loss function
    metrics = ['accuracy']
)
Executed at 2024.02.29 18:37:36 in 7ms
```

```
50 1 vocabulary_size = 5000
2 count_vector = CountVectorizer(max_features=vocabulary_size,
3                             preprocessor=lambda x: x,
4                             tokenizer=lambda x: x)
5 X_train = count_vector.fit_transform(X_train).toarray()
6 X_val = count_vector.transform(X_val).toarray()
7 X_test = count_vector.transform(X_test).toarray()
Executed at 2024.02.29 18:07:47 in 2s 444ms
> /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/feature_extraction/text.py:525:

251 1 from keras.preprocessing.text import Tokenizer
2 from keras.preprocessing.sequence import pad_sequences
3
4 max_words = 5000
5 max_len=100
6
7 def tokenize_pad_sequences(text):
8     """
9         This function tokenize the input text into sequences of integers and then
10        pad each sequence to the same length
11        """
12        # Text tokenization
13        tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
14        tokenizer.fit_on_texts(text)
15        # Transforms text to a sequence of integers
16        X = tokenizer.texts_to_sequences(text)
17        # Pad sequences to the same length
18        X = pad_sequences(X, padding='post', maxlen=max_len)
19        # return sequences
20        return X, tokenizer
21
22 print('Before Tokenization & Padding \n', df['clean_text'][0])
23 X, tokenizer = tokenize_pad_sequences(df['clean_text'])
24 print('After Tokenization & Padding \n', X[0])
Executed at 2024.02.29 18:07:51 in 4s 828ms
```

```
Before Tokenization & Padding
when modi promised "minimum government maximum governance" expected him begin the difficult job reforming the state
does take years get justice state should and not business and should exit psus and temples
After Tokenization & Padding
[ 41  2 394  67 1956 1026  45 2254  11132 217  1 169  32
 143 100  57  60 1049 169  49  3  9 517  3 49 2525  3
 2462  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  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  0 ]
```

Optimizers

Adam	SGD
Slower epoch	Faster epoch
Needs few epochs to evaluate	Need many epochs to evaluate
Less predictable	More predictable
Slightly more accurate	Less accurate

Conclusions

Models

	Initial	LogR	LSTM	Final
Speed (Fitting)	4min	17sec	6-20min	2min
Accuracy (Test data)	80%	85%	92%	93%

93%

Accuracy plateau

Future Improvements

- Broader dataset
- Combination of different models
- Better preprocessing

