אוניברסיטת
אריאל

Sentiment Analysis of Social Media's Messages

Deep Learning Final Project

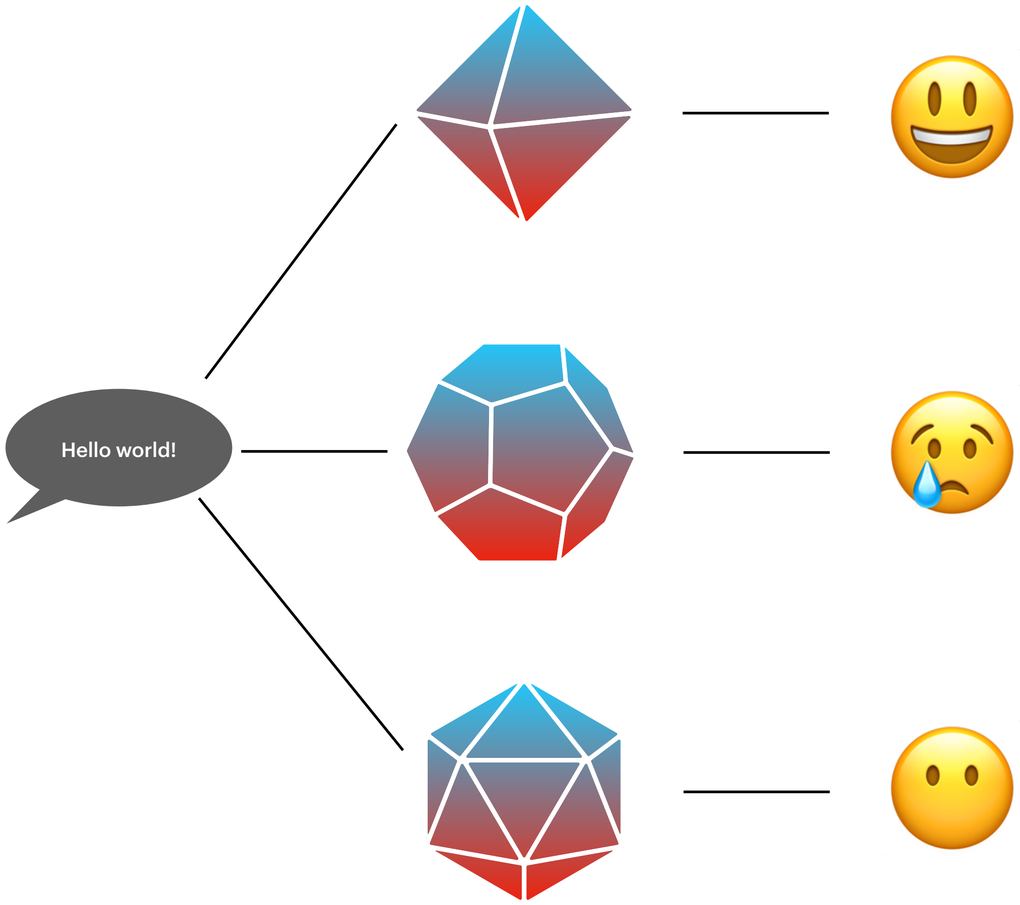
Nikita Breslavsky - 332363498
Barak Finkel - 206491714
Gilad Fisher - 318471109

2024

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

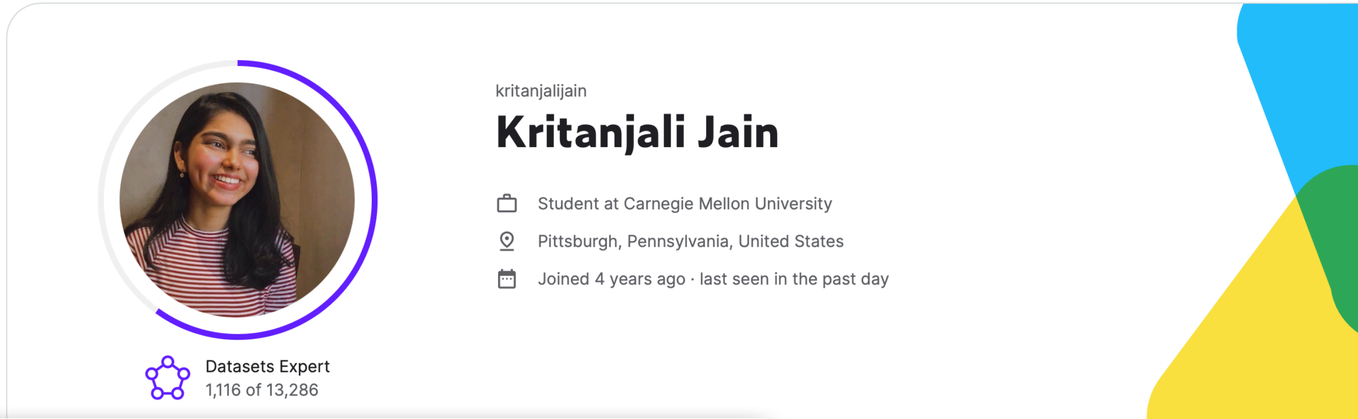
By Kylie Ying

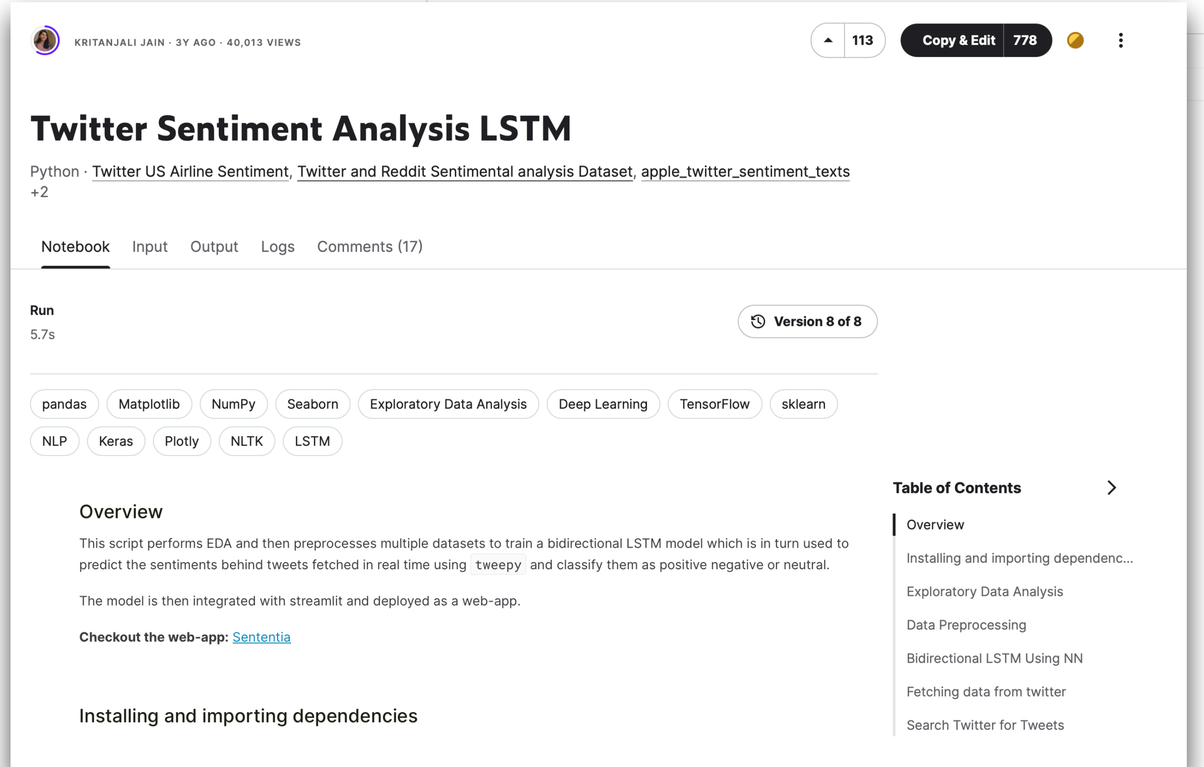
- Practical knowledge
- Data preprocessing
- Model building

Related Notebook Research

By Jain Kritanjali

- Noise reduction
- Text vectorizing
- LSTM

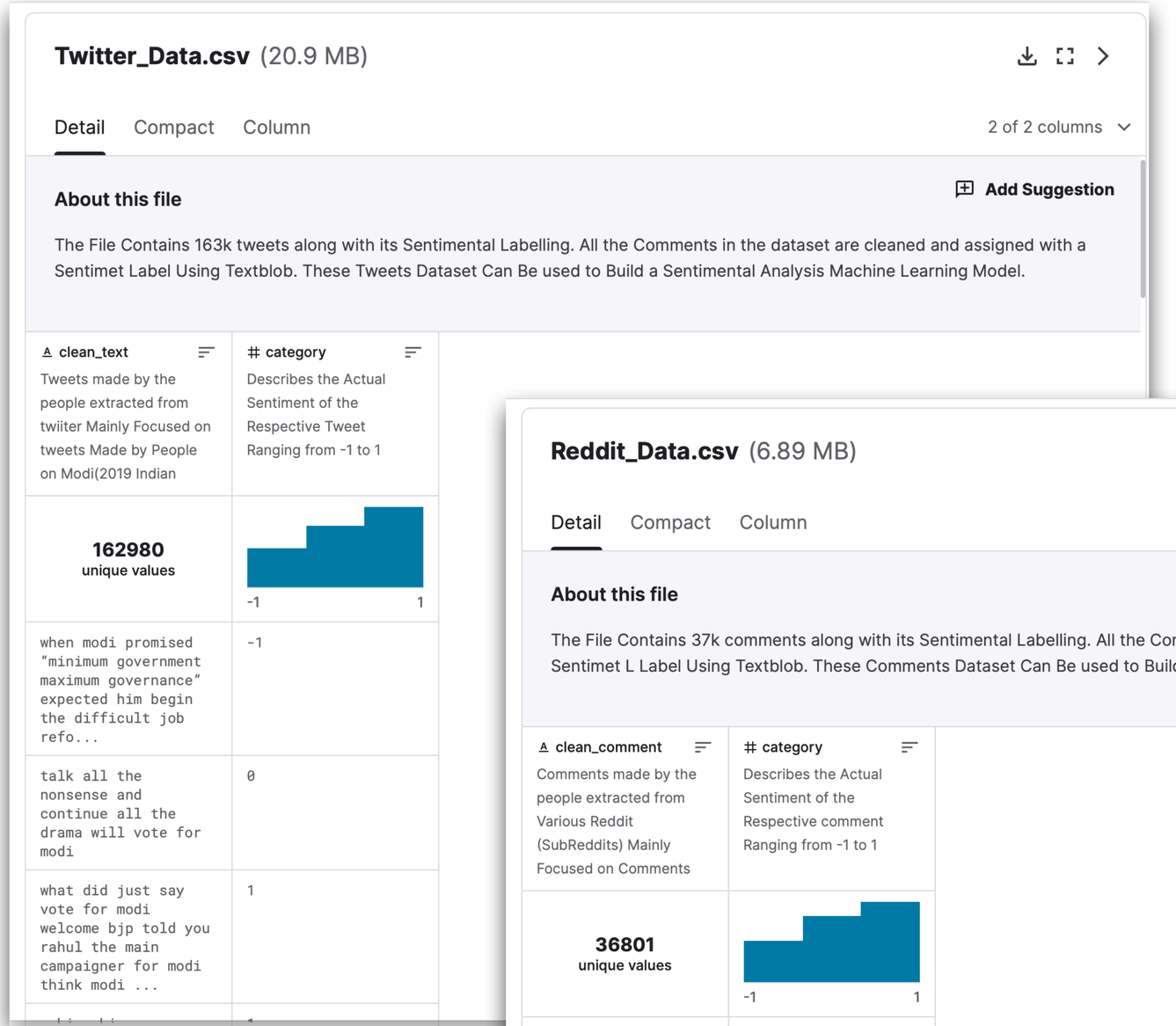




Final Model Overview

Dataset

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



Preprocessing

- Noise reduction
- Data splitting
- No vectorization

```
pattern  = re.compile(r"[^a-zA-Z0-9]")
def post_to_words(post):
    ''' Convert tweet text into a sequence of words '''
    # convert to lower case
    text = post.lower()
    # remove non letters
    text = re.sub(pattern, " ", text)
    # tokenize
    words = text.split()
    # remove stopwords
    words = [w for w in words if w not in stopwords.words("english")]
    # apply stemming
    words = [PorterStemmer().stem(w) for w in words]
    # return list
    return words

print("\nOriginal tweet ->", df['clean_text'][0])
print("\nProcessed tweet ->", post_to_words(df['clean_text'][0]))
```

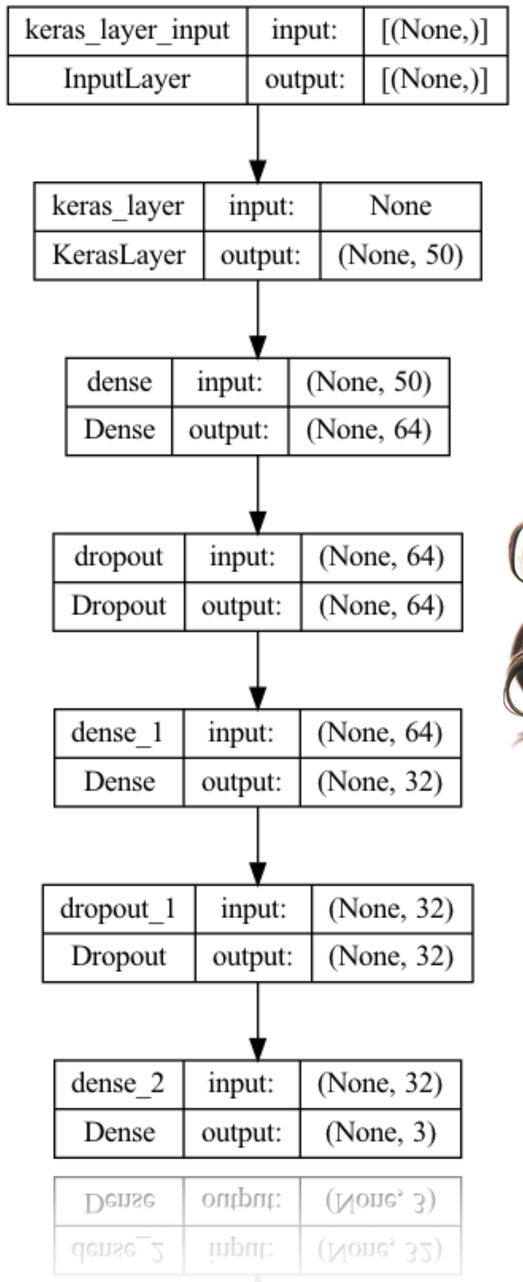
Executed at 2024.03.01 13:10:37 in 9ms

```
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']
```

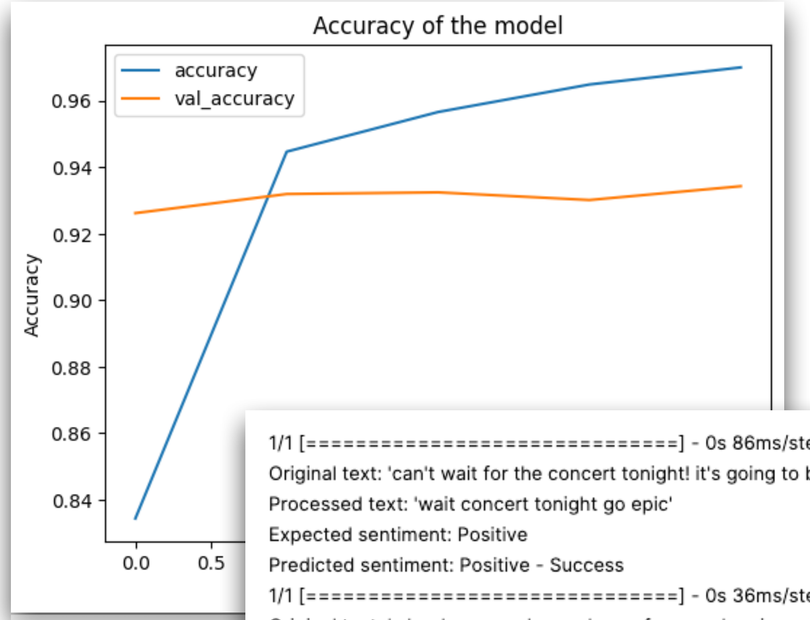
Model

- Google’s embedding layer
- Dense layers
- Dropout layers
- Softmax layer
- Adam optimizer
- Categorical Cressentropy



Simulation Results

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



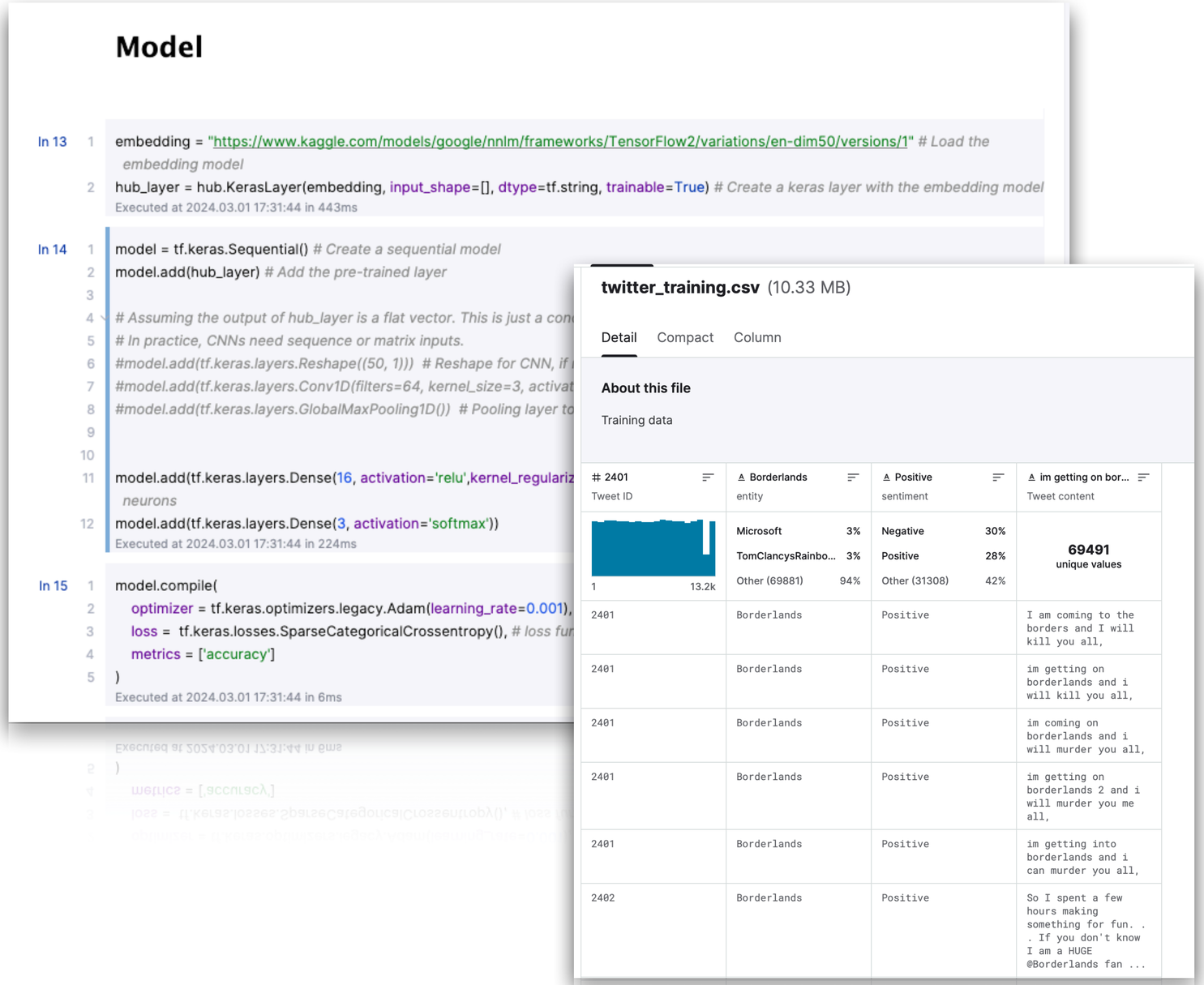

```
1/1 [==============================] - 0s 86ms/step
Original text: 'can't wait for the concert tonight! it's going to be epic!'
Processed text: 'wait concert tonight go epic'
Expected sentiment: Positive
Predicted sentiment: Positive - Success
1/1 [==============================] - 0s 36ms/step
Original text: 'why does my phone always freeze when i need it the most?'
Processed text: 'phone alway freez need'
Expected sentiment: Negative
Predicted sentiment: Neutral - Failed
1/1 [==============================] - 0s 36ms/step
Original text: 'scientists discovered water on mars! incredible step for space exploration.'
Processed text: 'scientist discov water mar incred step space explor'
Expected sentiment: Positive
Predicted sentiment: Neutral - Failed
1/1 [==============================] - 0s 37ms/step
Original text: 'stuck in traffic again. monday mornings are the worst.'
Processed text: 'stuck traffic monday morn worst'
Expected sentiment: Negative
Predicted sentiment: Negative - Success
1/1 [==============================] - 0s 36ms/step
Original text: 'absolutely gorgeous weather outside. perfect for a walk in the park!'
Processed text: 'absolut gorgeou weather outsid perfect walk park'
Expected sentiment: Positive
Predicted sentiment: Positive - Success
1/1 [==============================] - 0s 35ms/step
Original text: 'has anyone experienced issues with their heating not working? looking for tips.'
Processed text: 'anyon experienc issu heat work look tip'
Expected sentiment: Neutral
Predicted sentiment: Neutral - Success
```

```
20/20 [==============================] - 1s 25ms/step - loss: 0.2676 - accuracy: 0.9308
```

Other attempts

Initial

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



Logistic Regression

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

```
y = df['category'] # Target
X = [' '.join(post_to_words(text)) for text in df['clean_text']]
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

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

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

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
# 3. Prediction and Evaluation
y_pred = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Detailed performance report
print(classification_report(y_test, y_pred))
```
Executed at 2024.02.29 17:57:22 in 35ms

```
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
```

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']
)
```

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

```
vocabulary_size = 5000
count_vector = CountVectorizer(max_features=vocabulary_size,
                               preprocessor=lambda x: x,
                               tokenizer=lambda x: x)
X_train = count_vector.fit_transform(X_train).toarray()
X_val = count_vector.transform(X_val).toarray()
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:
```

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

max_words = 5000
max_len=100

def tokenize_pad_sequences(text):
    '''
    This function tokenize the input text into sequnences of intergers and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=max_len)
    # return sequences
    return X, tokenizer

print('Before Tokenization & Padding \n', df['clean_text'][0])
X, tokenizer = tokenize_pad_sequences(df['clean_text'])
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   1 1132 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   0   0   0   0
    0   0]
```

```
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(
    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

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