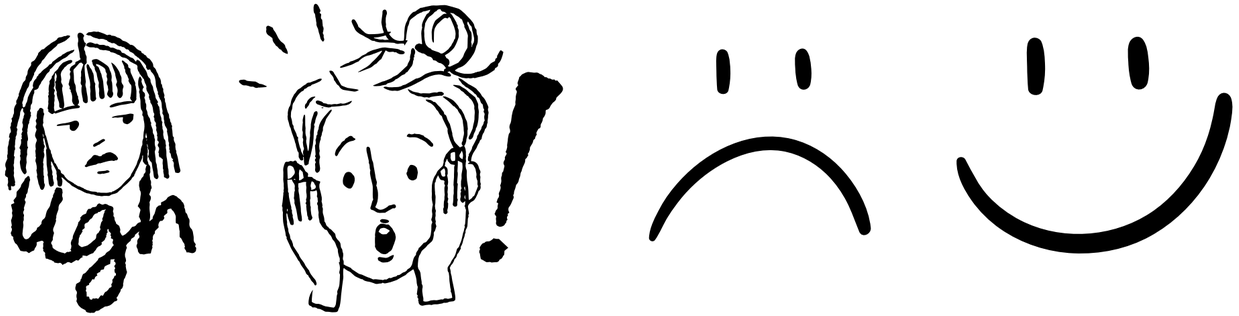
ADITYA PATKAR

Predicting Sentiment of Tweets Using the Sentiment140 Dataset

ADITYA PATKAR

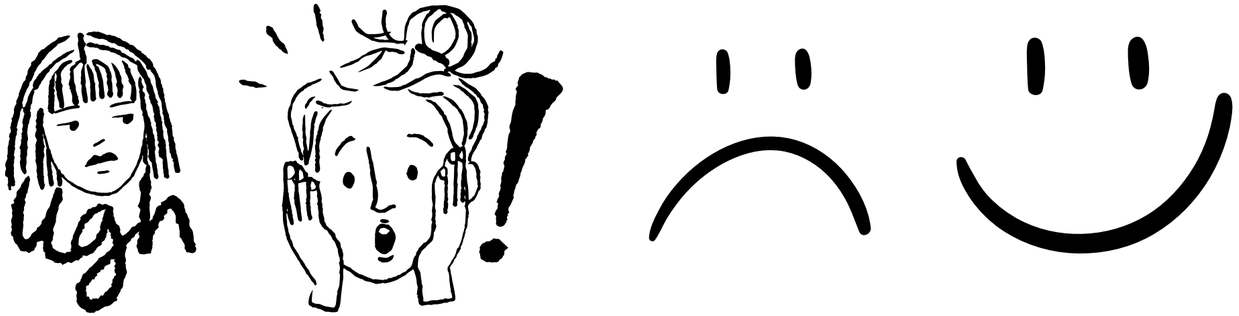


Introduction

- Exploring various architectures including **classical machine learning** and **deep learning** to **predict the sentiments** from tweets.
- **Binary classification** problem.
- The language and sentiment on twitter can be quite **polar and could become toxic** to many people. Thus using such a model would help us **better understand the nature of a tweet**.
- In this presentation we will:
  - Explore the **data**
  - Talk about **ways used to work with such a massive dataset**
  - Go over the **baseline** performance
  - Look at and **evaluate** different **model architectures** against the dataset
  - Talk about some **limitations**.

ADITYA PATKAR

Data Exploration

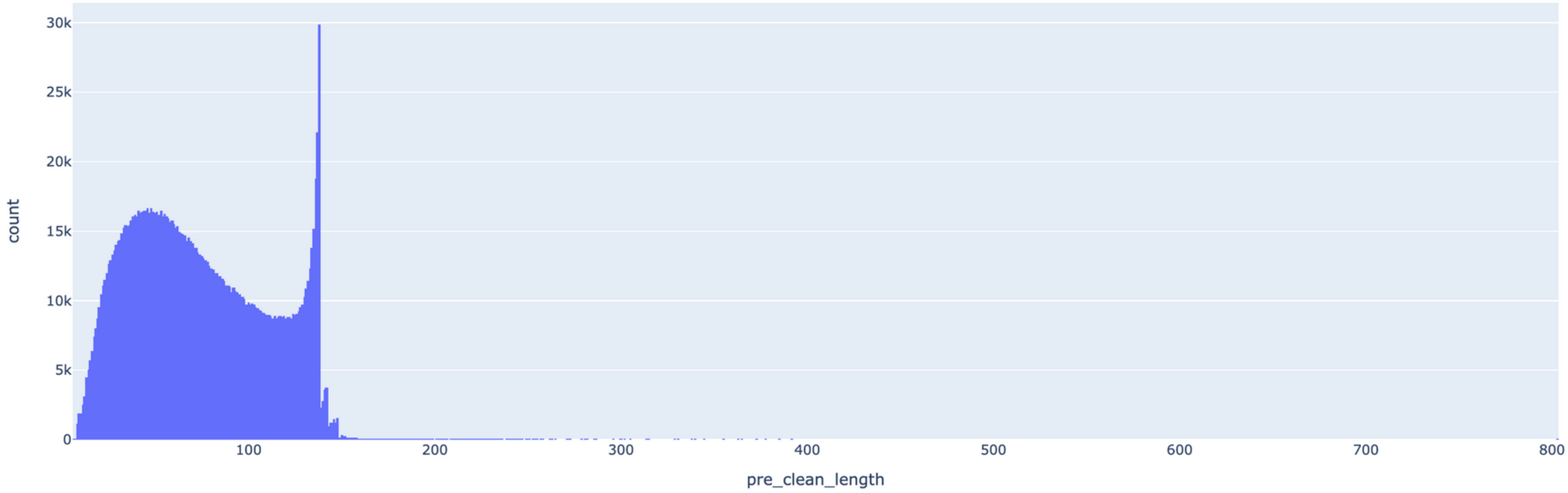


- 1.6 million rows!
- Columns: tweet_id, **target, tweet**, query_flag, username
- Data Split:
  - Positive (target=4): 800,000 rows
  - Negative (target=0): 800,000 rows
- Challenges:
  - The sheer size
  - Untidy data (inclusion of usernames, links etc)
- Source:
  - **Go, Alec and Bhayani, Richa and Huang, Lei from Stanford.**
- Example:
  - Tweet: getting some rest! tomorrow is event planning, job interviews, and running errands like getting stuff to make my moms bday cake!!
  - **Target = 4**

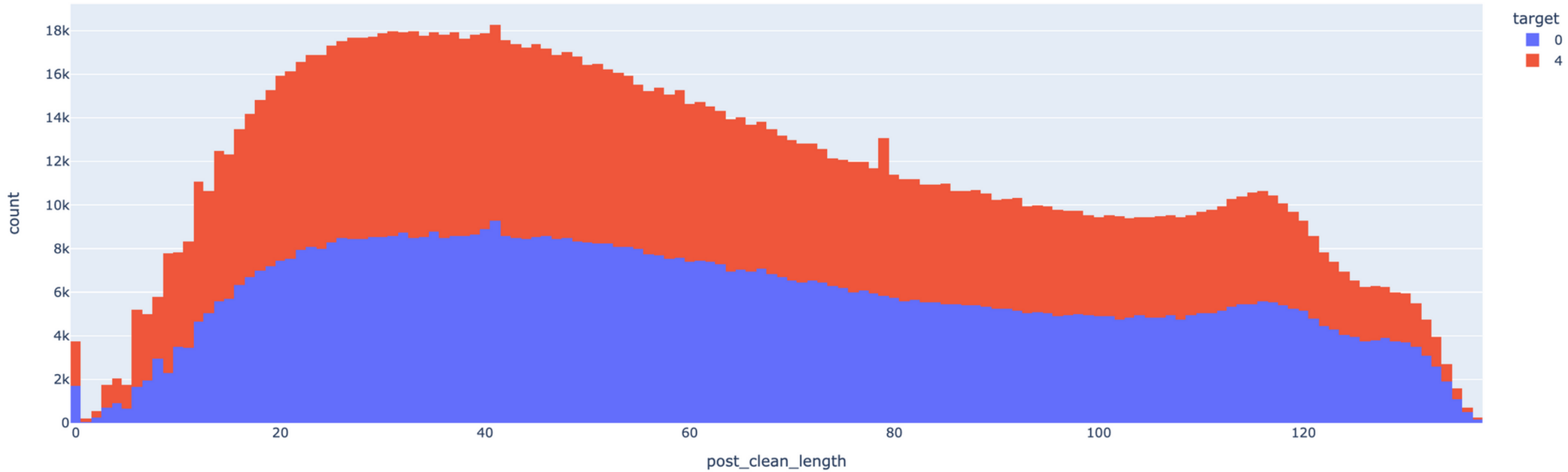
ADITYA PATKAR

Data Cleaning

- Calculate pre-cleaning length.
- **Convert HTML to text** using BeautifulSoup to get rid of words like &amp and some big html code blocks in some tweets.
- Remove **usernames and website links** using regex, this helped the performance of the model.
- Remove **non A to Z characters** using regex.
- **Replace non ASCII characters** with a **placeholder**. Helped the performance massively.
- **Remove stop-words**.
- Drop rows with **null values**. A tweet could completely be filled with stop-words. We don’t need that.
- Calculate post-cleaning length.



Pre Clean Length



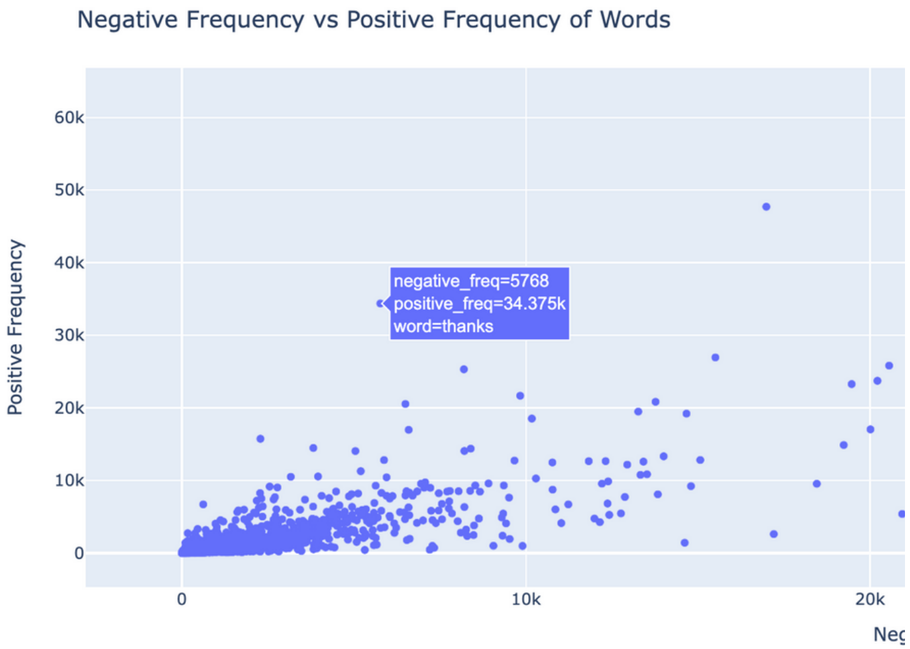
Post Clean Length

Word Cloud - Negative

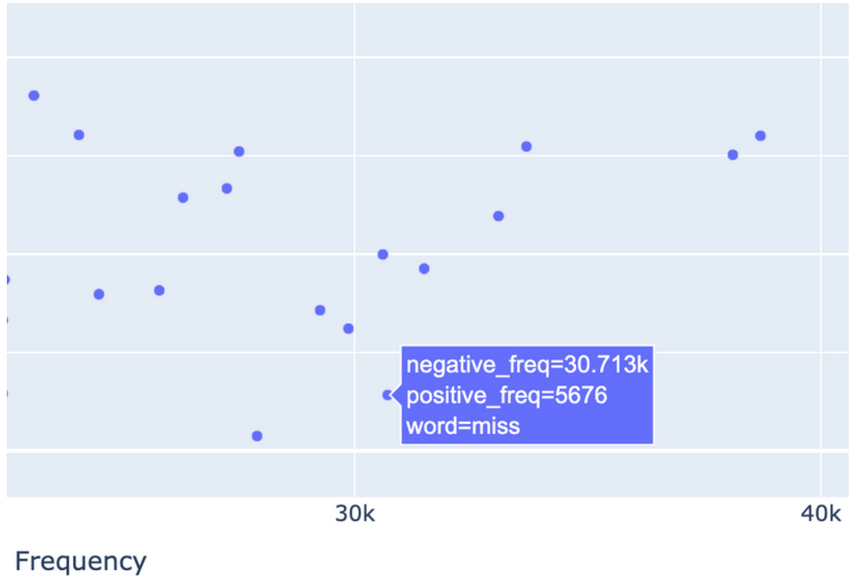
Word Cloud - Positive

ADITYA PATKAR

Thanks



Miss



Word Frequency

ADITYA PATKAR

Frameworks Used to handle Big Data

- **CuDF: GPU supported DataFrame** operations. Tried to use for data cleaning but failed due to version conflicts with other libraries
- **Dask**: For **parallelized and lazy computing**. Used during EDA and Data cleaning, mostly because of **8x faster performance of apply()** function as compared to pandas
- **PySpark**: DataFrame, Rdd and PySpark ML used during the model training process of classical ML algorithms

Frameworks Used to handle Big Data

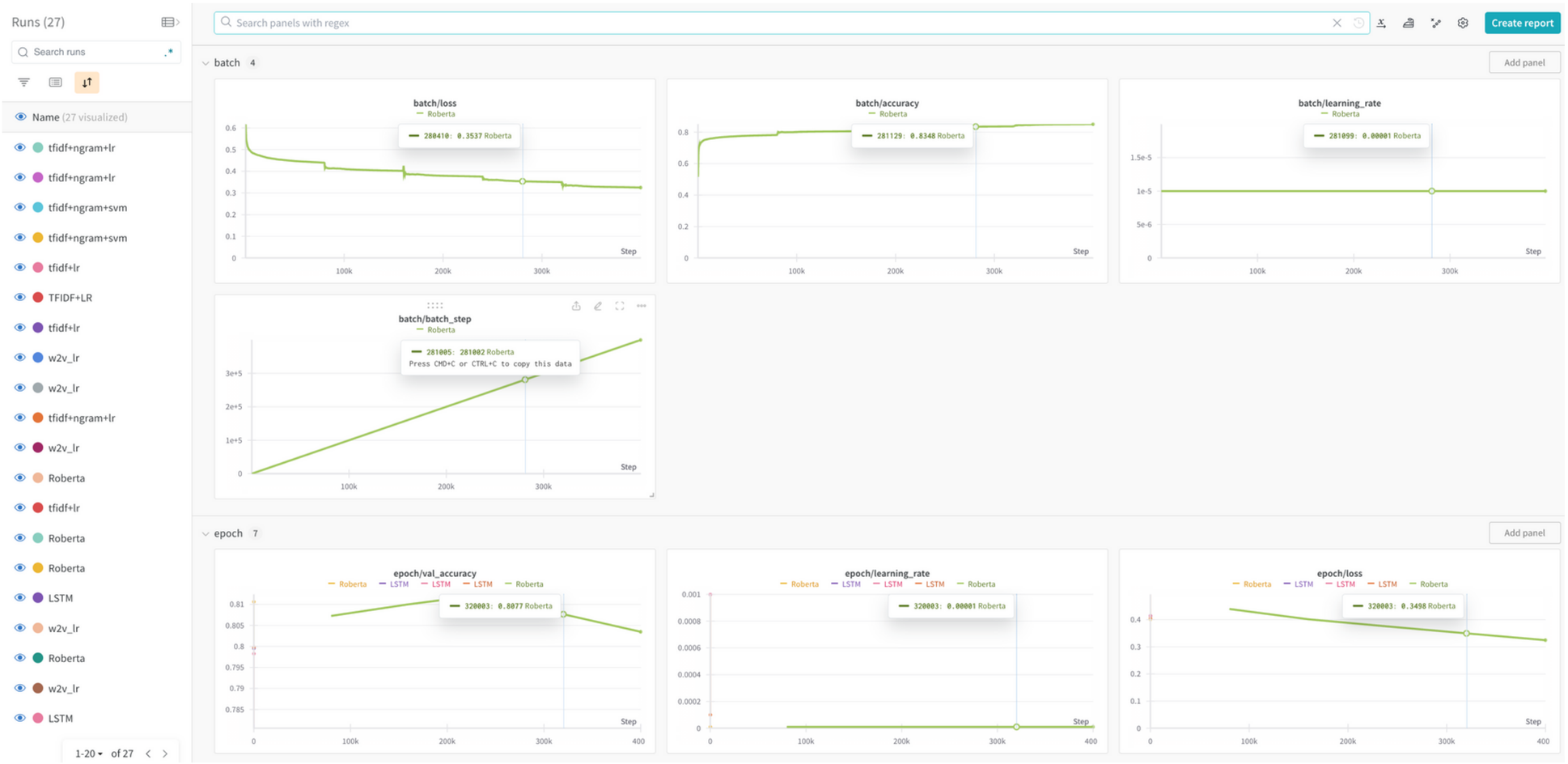
- **AWS S3:** For storing intermediate dataset files, model files, embeddings etc. because of ability of **quick retrieval** and **high reliability**.
- **AWS Sagemaker:** For **training the deep learning models** using GPU backed instances. Sped up the training process as compared to free version of colab.
- **Apache Parquet**: Much **faster compared to CSVs**. Designed for efficient data storage and retrieval

Frameworks Used to handle Big Data

- **Weights & Biases**:
  - Cloud based platform to **log hyper-parameters, save samples, save models** and to keep track of **evaluation metrics**.
  - Helpful in **comparing** models
  - Utilized to keep track of evaluation metrics with respect to hyper-parameter tuning
  - One single place to view everything related to each model built
  - Easy **storage of model weights** for easy loading

ADITYA PATKAR

Weights & Biases



Models Trained

- **Classical ML:**
  - Logistic Regression
  - SVM
  - Random Forest
  - All trained with various types of features
  - Used **PySpark ML + PySpark DataFrames**
- **Deep Learning:**
  - LSTM
  - RoBERTa
  - Used **Keras + Tensorflow**

ADITYA PATKAR

Classical ML Training Process

1. Connect to a **Spark Context** and create a run in **W&B**
2. Set **Config** Parameters
3. Train-Val-Test Split
4. Create a pyspark pipeline of
    a. Tokenizer
    b. **Feature Creation Step (n-gram/tfidf/w2v)**
    c. Label Indexing
    d. Model Fitting
5. Calculate **evaluation metrics** (F1, Recall, Precision, AUC, Accuracy)
6. **Log metrics** to W&B

Logistic Regression

- **Hashing TF + IDF:**
  - Hashing TF Converts text into a numerical representation by **counting occurrences of each word**.
  - IDF weighs down the frequent terms while scaling up the rare ones which helps in **diminishing the importance of words that occur very frequently** and thus offer little to no unique information about the document.
  - Efficient in handling large datasets due to the **hashing trick**.
  - Avoids the need for a vocabulary fit, hence **faster and scalable**.
  - Makes it **suitable for real time streaming applications**.

Logistic Regression

- **1 + 2 + 3 gram + Count Vectorizer + IDF:**
  - The different values of n (1, 2, 3) capture **single words, bi-grams** (pairs of consecutive words), and **tri-grams** (three consecutive words) which helps in **capturing context** and word order.
  - Count Vectorizer **transforms text documents into vectors** where each dimension corresponds to a specific n-gram in the corpus. IDF used in conjunction.
- **3 gram + Word2Vec**
  - **Word2Vec creates word embeddings** that capture **semantic meanings based on the context** in which words appear, here working with tri-grams.
  - The resulting vectors capture many linguistic regularities and patterns, with **similar words having similar embeddings**.

ADITYA PATKAR

Logistic Regression Results

(max_iter = 200; reg_param=0.001; train_size=0.95, val_size=test_size=0.025; elasticnet_param=0.001)

TF + IDF (baseline)		Ngram + CV + IDF		Ngram + W2V	
76.96% F-1 (Macro)	77.03% Precision (Macro)	78.05% F-1 (Macro)	78.17% Precision (Macro)	61.66% F-1 (Macro)	61.64% Precision (Macro)
76.97% Recall (Macro)	84.07% AUC	78.03% Recall (Macro)	85.62% AUC	63.43% Recall (Macro)	67.16% AUC
76.58% Accuracy	2m 31s Runtime	78.06% Accuracy	6m 14s Runtime	63.47% Accuracy	7m 35s Runtime

SVM and Random Forest

- **SVM + Ngram + CV + IDF**
  - Works by finding the **hyperplane** that best divides a dataset into classes.
  - Effective in **high-dimensional spaces**
- **RF + Ngram + CV + IDF**
  - Operates by **constructing multiple decision trees** during training and outputting the class.
  - Each tree is **trained on a random subset of features** and samples, making the model **robust to noise and overfitting**.

ADITYA PATKAR

SVM and RF Results

(max_iter = 100; reg_param=0.03; train_size=0.95, val_size=test_size=0.025; max_depth=5)

77.37%
F-1 (Macro)

77.92%
Precision (Macro)

77.37%
Recall (Macro)

85.28%
AUC

77.46%
Accuracy

4m 25s
Runtime

SVM

57.74%
F-1 (Macro)

64.47%
Precision (Macro)

60.56%
Recall (Macro)

67.01%
AUC

60.61%
Accuracy

6m 14s
Runtime

Random Forest

ADITYA PATKAR

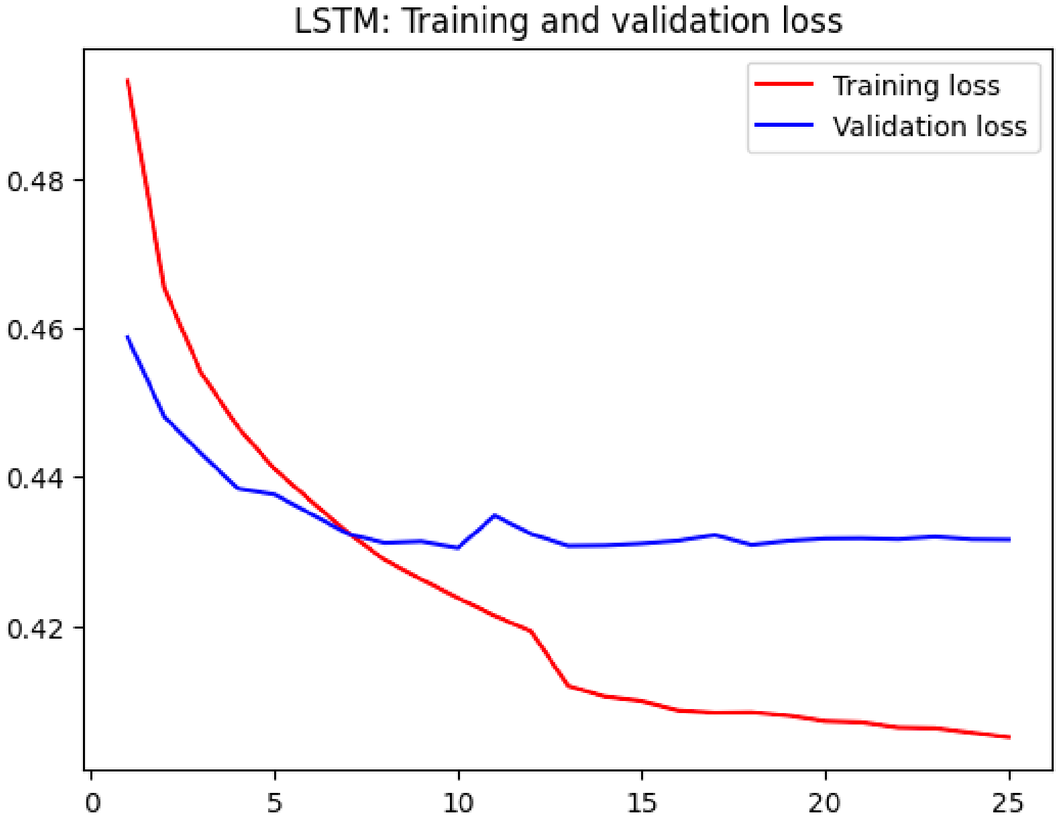
LSTM + GloVe Embeddings

- LSTM learns **long term dependencies**
- Each module is made of four interacting layers, including a **memory cell** and three gates (input, output, and forget gate).
- Highly effective for **tasks involving sequences**
- **GloVe** representations capture **semantic and syntactic meanings** of words using a **co-occurrence matrix**.
- Unlike word2vec, GloVe does not rely just on local context but on global word-word co-occurrence
- GloVe embeddings are used to convert words in text data into their corresponding vector representations. These vectors then serve as input to the LSTM network.

LSTM Training

- Pad sequences to max length, load embeddings from file and get weights matrix.
- Create an LSTM model with:
  - **Embedding layer** with weights matrix
  - **Dropout** for robustness
  - **LSTM** layer (200 modules)
  - 2 Dense layers, first with **leaky relu** and final being with **sigmoid**
- Train with **Binary Cross Entropy** loss with **Adam Optimizer** for **25** epochs with a **ReduceOnPlateau LR scheduler** with minimum LR of 0.0001.
- Evaluate and log to W&B.

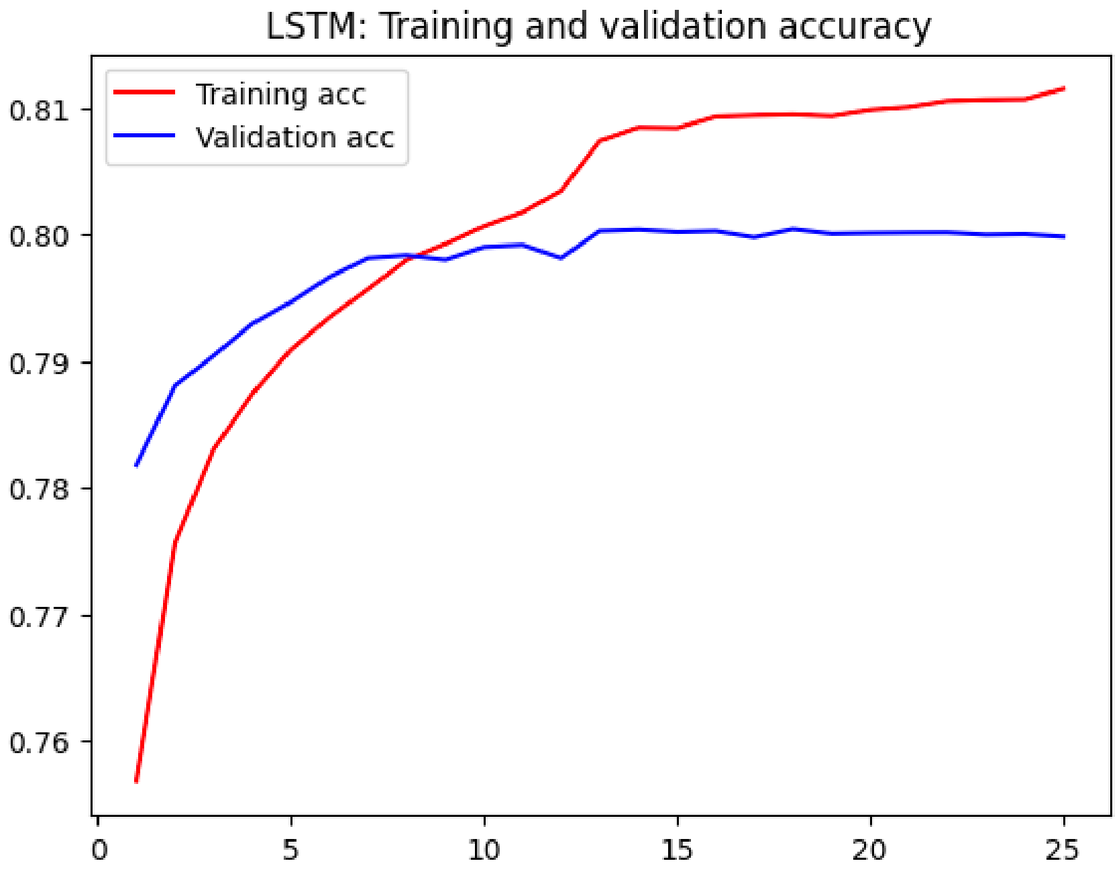
Training Outcome



Loss

- As we trained the model, our **training loss kept going down,** but after a point, the **validation loss** did not go down.

- But as we had implemented **early stopping**, the model **loaded** the **"best" weights**, that is, the weights corresponding to the lowest validation loss.



Training Outcome: Accuracy

Just like the loss, the **accuracy** on the **training data went up** with each epoch. The **validation accuracy**, however, **increased** a lot during the **first 10 epochs** and **then it marginally increased**, and mostly **flattened**.

ADITYA PATKAR

RoBERTa

RoBERTa builds on BERT’s language masking strategy. It modifies key hyperparameters in BERT, including removing BERT’s next-sentence pretraining objective, and training with much larger mini-batches and learning rates. It is known for good performance in

https://ai.facebook.com/blog/roberta-an-optimized-method-for-pretraining-self-supervised-nlp-systems/

ADITYA PATKAR

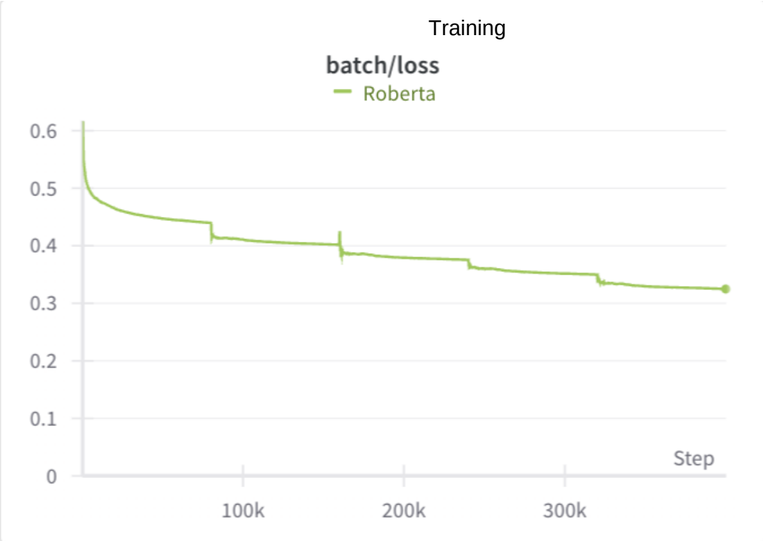
RoBERTa Training

- Load **model and tokenizer** of roberta-base from HuggingFace
- Tokenize tweets
- Convert data to tensorflow dataset and batch it
- Compile and train model on **A100 GPU** with **Adam optimizer** and **Binary Cross Entropy loss** for **5 epochs** with a static learning rate of **1e-5**.
- Save the best model corresponding to least validation loss.
- Evaluate the model and log metrics to W&B.

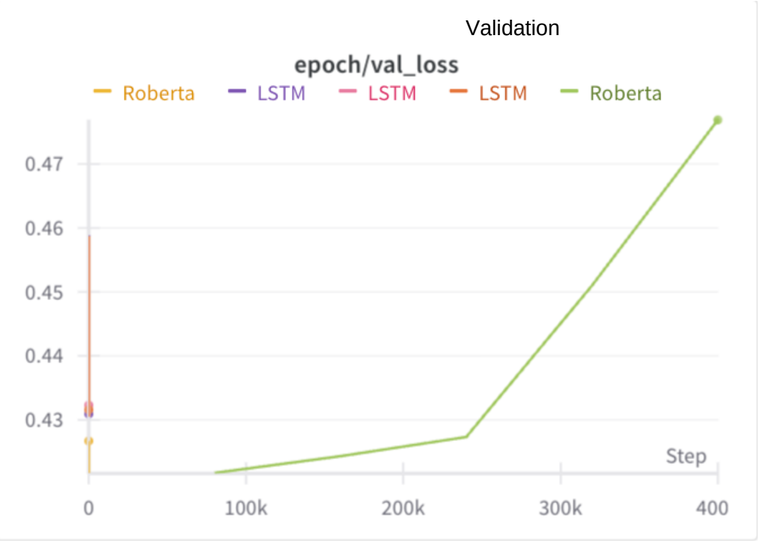
ADITYA PATKAR

Training Outcome

Training

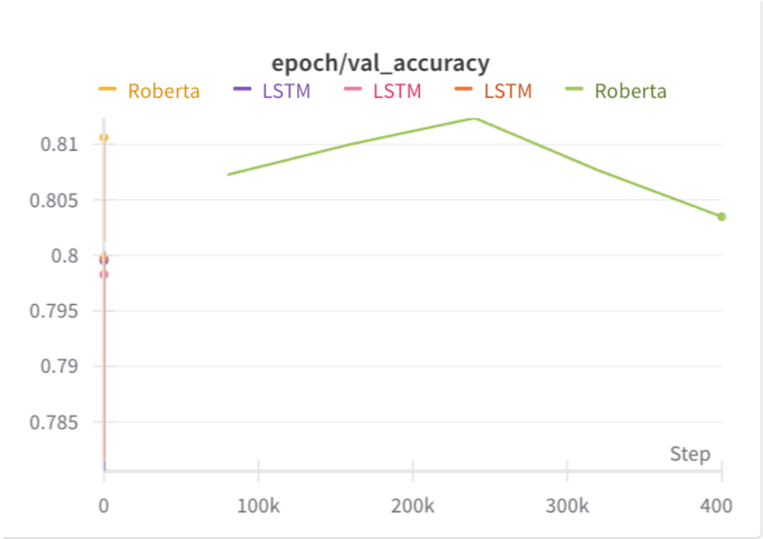


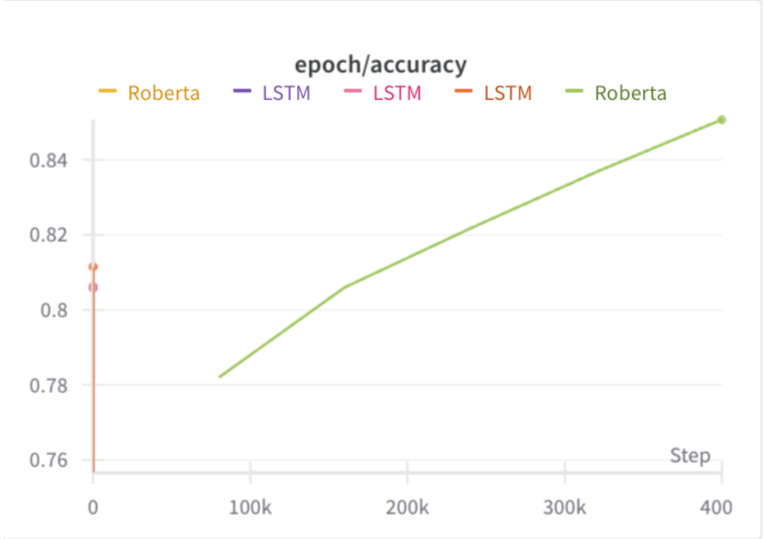
Validation



Loss

- While **training loss went down**, **validation loss increased** after first 2 epochs or 80000 steps
- This shows that the **model gradually overfit** to the training data.
- Model **co-responding to least validation loss was saved**





Training Outcome: Accuracy

Just like the loss, the **accuracy** on the **training data went up** with each epoch. The **validation accuracy**, however, **decreased** after 3rd epoch.

ADITYA PATKAR

Macro Metrics on Test Set

80.79%
F-1 (Macro)

75.29%
Precision (Macro)

87.15%
Recall (Macro)

80.09%
Accuracy

31m 37s
Runtime

LSTM

81.60%
F-1 (Macro)

77.42%
Precision (Macro)

86.26%
Recall (Macro)

80.55%
Accuracy

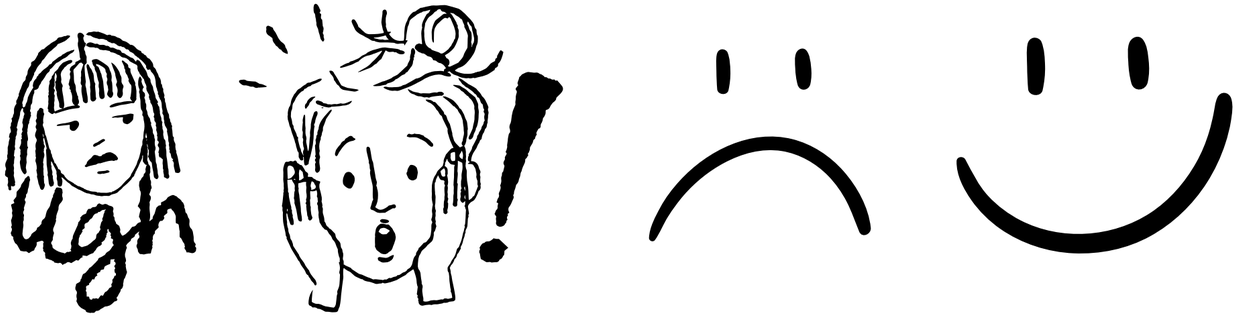
11h 15m
Runtime

RoBERTa

ADITYA PATKAR

Conclusion

ADITYA PATKAR



Conclusion

- **Logistic Regression** + N-Gram + CV + IDF has **commendable performance** and a **really fast runtime**. It gives us the **best precision** which is important for sentiment analysis where false positives are costly.
- **RoBERTa** gives the **best F1 score and accuracy** while being expensive to train both time and cost wise.
- Deep learning models fared better than the classical models.
- Using **parallelizing tools** like Dask and PySpark proved **critical** while handling such a massive dataset.
- Few things that can improve the performance:
  - More **data cleaning.**
  - More **epochs of training on RoBERTa** with a **LR Schedule.**
  - **Tuning of hyper-parameters** of both deep learning models.

ADITYA PATKAR

Thank you!