Tweet Author Classification

Shreyas Piplani

Rituraj Singh

18752 Project

Problem Statement

The expansion of social media platforms has allowed people to share their thoughts, comments, and feelings with the world. Twitter, for example, now has more than 300 million active users. One could also argue that it has had a disproportionate influence on the world, mainly because it has a huge number of politicians, celebrities, and journalists.

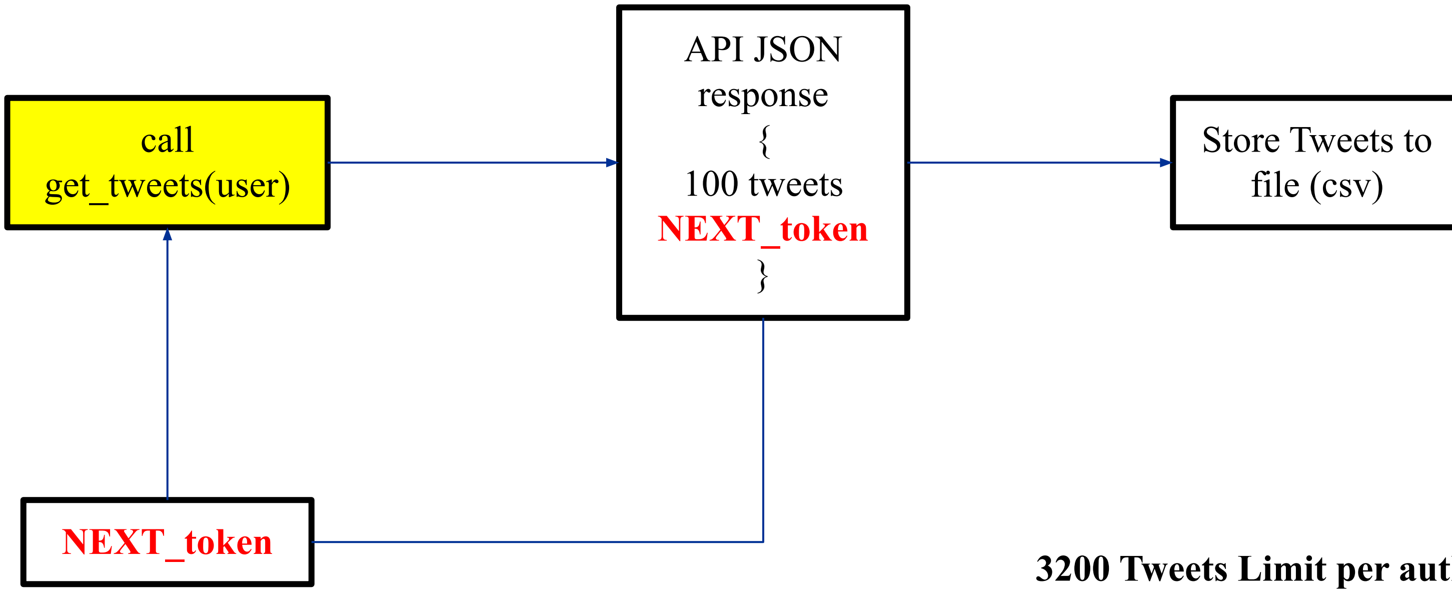
To get a deeper insight of user activity on Twitter, we have created a "Tweet Author Classifier." This project involves the following steps:

- Data collection using Twitter API + Pre-processing
- Feature Extraction + Data visualization
- Classification of tweets + comparing performance of different ML methods

Data Collection

- Selected Twitter public figures in Tech, Sports and Politics
  - Eg. Elon Musk, Bill Gates, Barack Obama, Joe Biden, Manish Bhasin
- Used Twitter API v2 to retrieve tweets for each user
- Tweet Retrieval Limit per API call - 100
- Used Pagination to get > 100 tweets
- Total Tweet Retrieval Limit after Pagination - 3200
- Data format- CSV and .npy file
- 38,230 Tweets retrieved
- Note on Pagination on next slide

Pagination for Tweet Retrieval in v2



3200 Tweets Limit per author

Next token is needed to trigger Pagination and retrieve consecutive tweets in chunk size of 100

Dataframe

	user
tweet	
Apple CEO Tim Cook says results in China are encouraging -- "We thought we would improve some but we improved a bit more"	emilychangtv
It's easy to forget how big the world is. It's really, really big. You could create a device that displays one person's face every second and that never shows the same person twice. In perpetuity.	fchollet
In the weekly address, the President discusses how to make it easier for communities to adopt the Fair Housing Act: http://t.co/pQyCEW3WJT	barackobama
My full sit-down with @Rakuten CEO Mickey Mikitani -- about running the "Amazon of Japan," investing in Lyft and Pinterest and sponsoring the Golden State Warriors! https://t.co/R58ij4Bqym https://t.co/aGDPNHjlUv	emilychangtv
@Abu9ala7 @EthereumMemes IMO ethereum isn't anti-Satoshi, it's a continuation of Satoshi's vision.	vitalikbuterin
@MikeBertolino1 They paid for 3 years of it, can't turn back now 😅	mkbhd
Frmr. FCC Chair Tom Wheeler joins us to discuss potential repeal of net neutrality rules on today's show. https://t.co/aE3BnElles	emilychangtv
OpenAI Five Arena is now OPEN!\n\n- Play against #OpenAIFive in competitive mode, or with it in cooperative mode: https://t.co/owmzogfOLg\n\n- Follow the leaderboard: https://t.co/L0rWNHRTpj\n\nArena is live for this weekend only. https://t.co/bGNetcx6ny	openai
@udiWertheimer @Burstup Because there are multiple clients and most likely at least one of them would implement things differently and so the chain would split?\n\nAs opposed to a one-client architecture where you always have to wait for humans to manually notice that something went wrong	vitalikbuterin
The House should mark the 50th anniversary of the Civil Rights Act by passing ENDA. @SpeakerBoehner @GOPLeader @NancyPelosi @WhipHoyer	tim_cook

Data Preprocessing

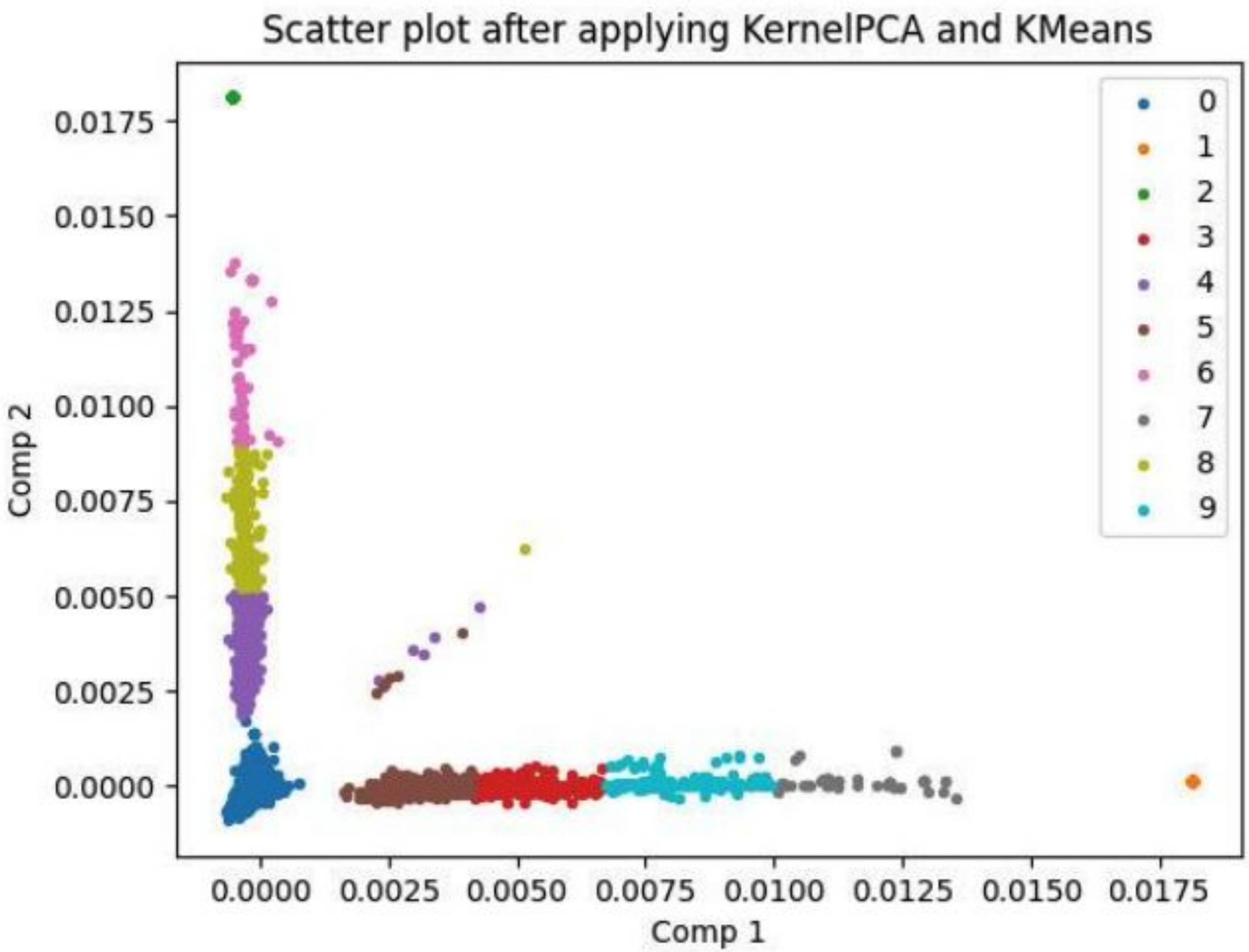
- Converted JSON responses to a CSV format
- Removed non-ASCII characters
- Removed hyperlinks from tweets
- Removed authors with < 2500 tweets
- Converted categorical labels to numerical labels
- Tokenized and removed stop words (is, are, the, in etc.) from tweets
- Total 13 authors i.e. M = 13 classes
- Omitted non-English tweets
- Ran TF-IDF on tweets and stored TF-IDF vectors in .npy format
- 70:30 Train-Test split

Feature Extraction

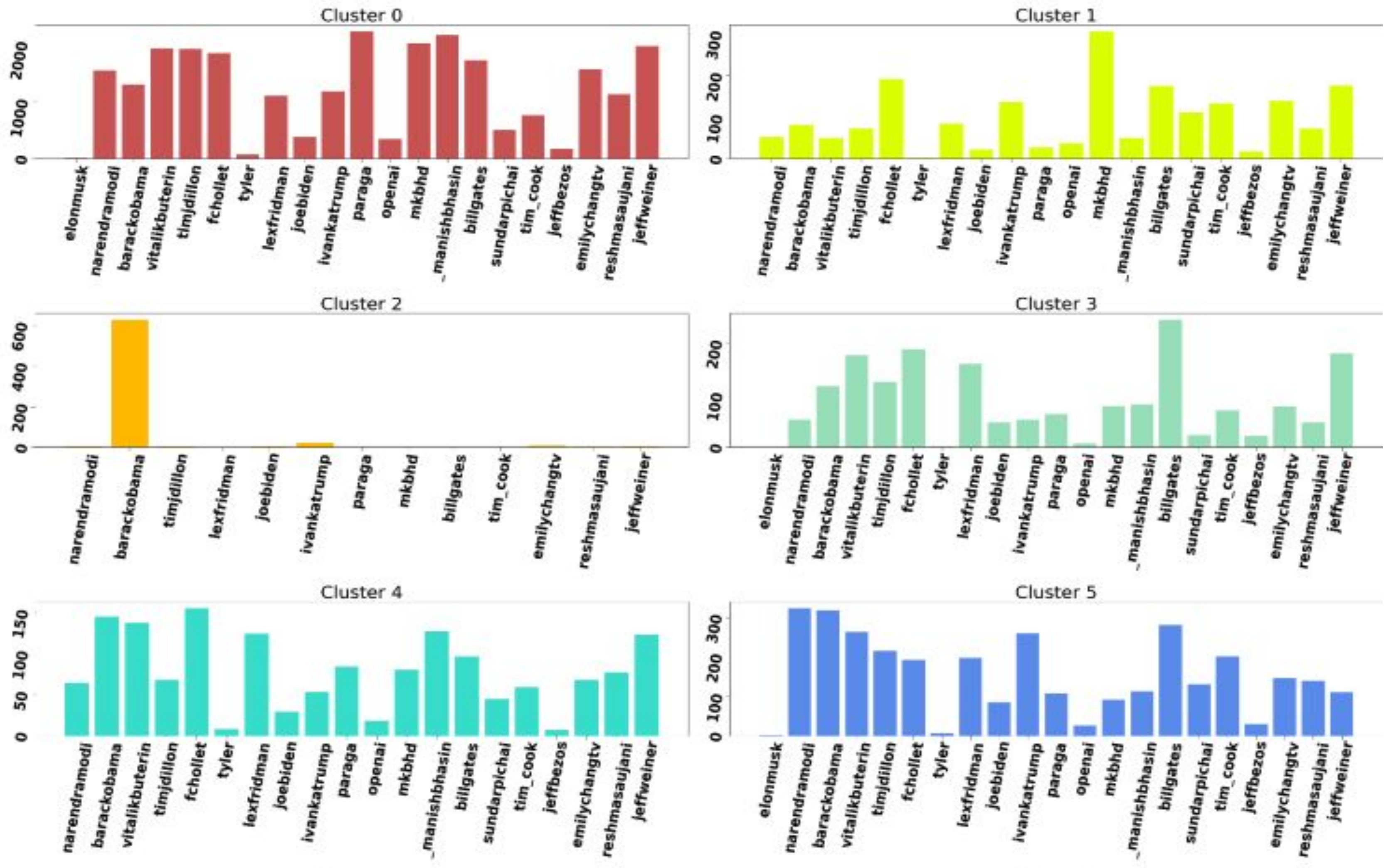
Techniques explored initially:

- TF-IDF:
  - TF: Term Frequency, IDF: Inverse Document Frequency
  - Gives more importance to words that occur more frequently in one tweet and less in other tweets
- Principal Component Analysis (PCA):
  - Used for dimensionality reduction on top of arrays generated using TF-IDF
  - Applied Kernel PCA with “rbf” kernel that stands for Radial Basis Function aka squared-exponential kernel
-

Scatter plot based on features extracted using TF-IDF and Kernel PCA:



Feature Extraction



Feature Extraction

Technique that we finally used:

BERT embeddings:

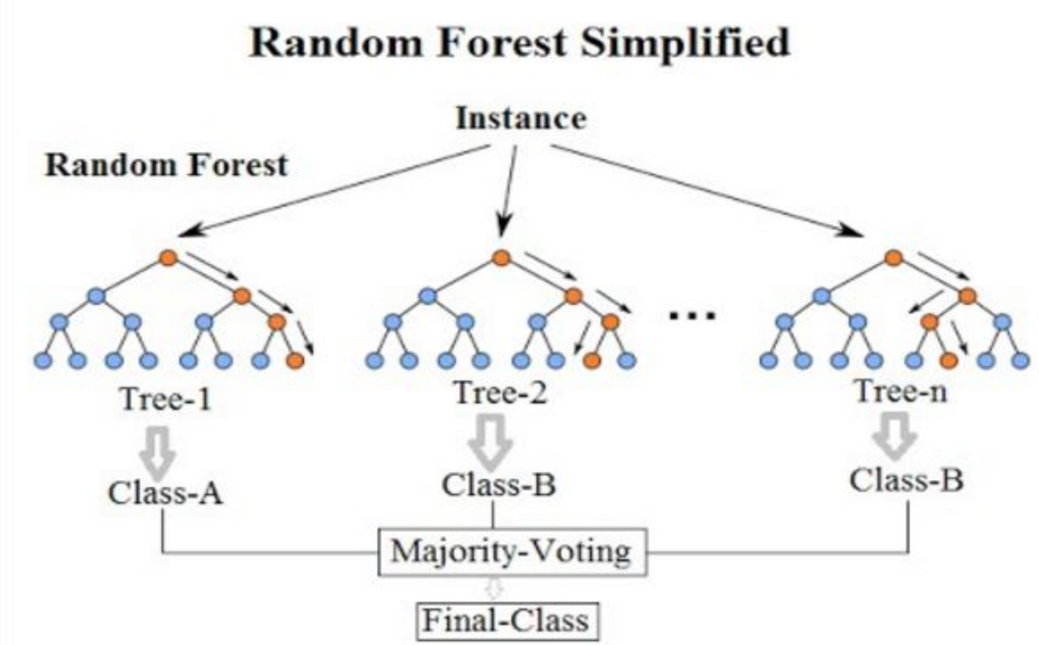
- Used SentenceTransformers module to get conext rich embeddings of size 384
- Used embeddings as features for all models
- BERT analyzes context of a word and has semantic depth

Classification Techniques

- Random Forest
- Neural Network
- Support Vector Machine
- Logistic Regression

Random Forest

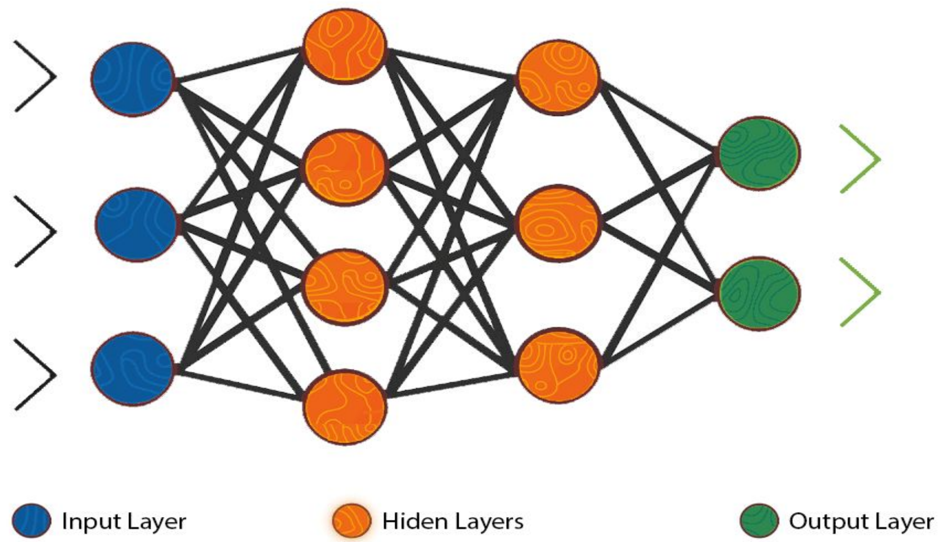
- Sklearn Random Forest Classifier
- Number of trees in the forest = 100
- Maximum depth of the tree = 10
- Training time = 32 seconds
- Accuracy = 48.61%



Source: https://en.wikipedia.org/wiki/Random_forest

Neural Network

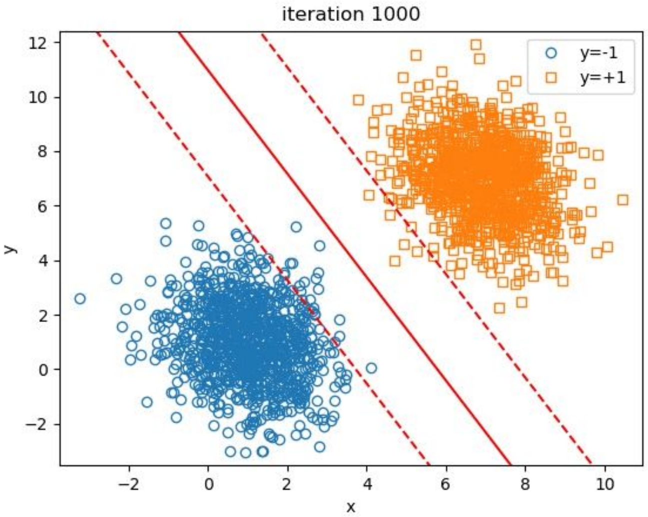
- Sklearn MLP Classifier
- Hidden layer sizes = (384, 256, 64, 13)
- Solver for weight optimization = ‘adam’
- L2 Penalty parameter (alpha) = 1e-5
- Initial learning rate = 0.001
- LR scheduler = ‘adaptive’
- Training time = 82 seconds
- Accuracy = 57.83%



Source:https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/

Support Vector Machine

- Sklearn Support Vector Classifier
- Performed standard scaling to features $$z = \frac{(x - \mu)}{\sigma}$$
- Kernel: Radial Bias Function
- Gamma = 1/num_features
- Training Time 78 seconds
- Accuracy 64.12 %



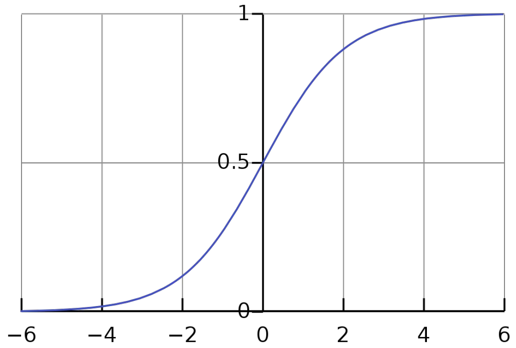
Source: https://nianlonggu.com/2019/05/24/tutorial-on-SVM/

Logistic Regression

- Sklearn Logistic Regression
- 100 iterations
- L2 regularization
- Limited memory BFGS solver
- Training Time 10.46 seconds
- Accuracy 60.26 %

$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

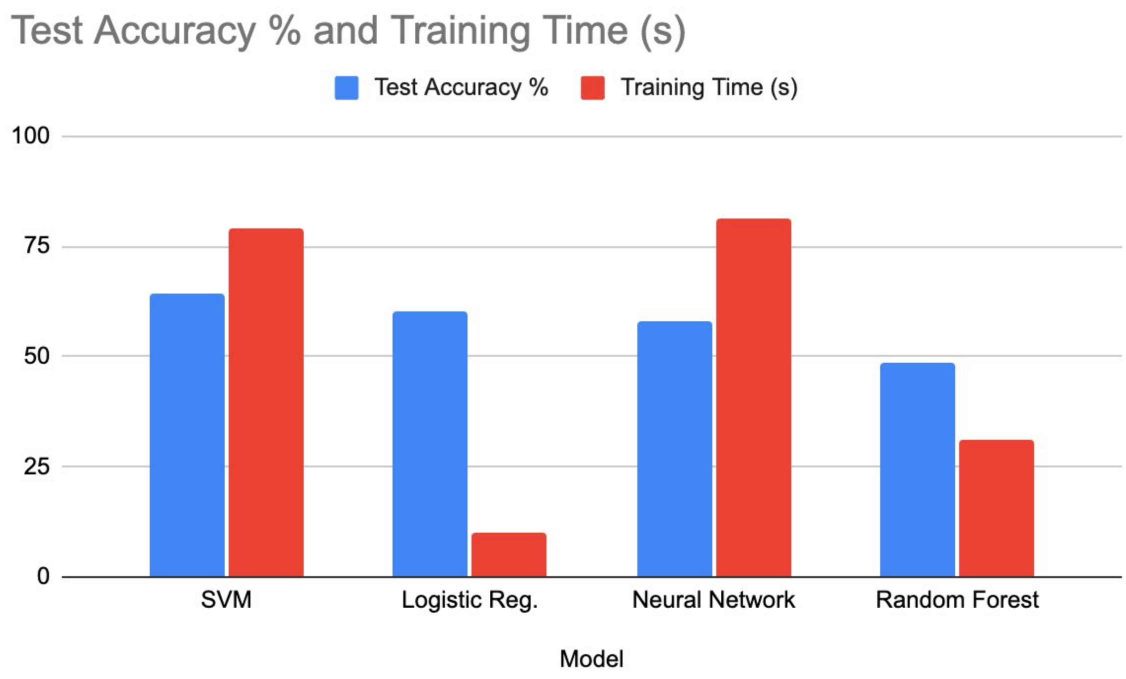
Sigmoid



Source: https://en.wikipedia.org/wiki/Sigmoid_function

Model Comparison

Model	Test Acc %	Precision	Recall	F1 Score	Training Time (s)
SVM	**64.12**	0.65	0.64	0.64	79.27
Logistic Reg.	60.26	0.60	0.60	0.60	**10.1**
Neural Net	57.83	0.60	0.58	0.58	81.11
Random Forest	48.61	0.55	0.46	0.46	31.25



Code Explanation

- Tweet Retrieval
- Preprocessing Tweets
- Training Models
- All code is pushed to Github at
  https://github.com/ritzdevp/Tweet-Author-Classification
- Contributors: Rituraj Singh (ritzdevp), Shreyas Piplani (shreyaspiplani)

Tweets Retrieval

```
import tweepy

#max_results per page can be in [5,100]
def get_tweets(user_id, client, num_tweets=10, max_results_per_page=5):
    pages = num_tweets//max_results_per_page
    tweet_text = {}

    response = client.get_users_tweets(user_id, max_results=5,
          exclude='retweets')

    for tweet in response.data:
        tweet_text[tweet.id] = tweet.text

    if ('next_token' not in response.meta):
        return {}
    next_token = response.meta['next_token']


    for i in range(pages-1):
        response = client.get_users_tweets(user_id, max_results=max_results_per_page,
            exclude='retweets', pagination_token=next_token)

        if (response.data == None):
            break
        for tweet in response.data:
            tweet_text[tweet.id] = tweet.text
        if ('next_token' not in response.meta):
            break
        next_token = response.meta['next_token']


    return tweet_text
```

Preprocessing

```
def tokenize(tweet):
    lemmatizer = nltk.stem.WordNetLemmatizer()
    tokenizer = TweetTokenizer()
    return [(lemmatizer.lemmatize(ele)) for ele in tokenizer.tokenize(tweet)]

def pp():
    df_data = pd.read_csv('data.csv')

    # All tweets
    tweet_list = df_data["tweet"].tolist()
    author = df_data["user"].tolist()

    tokenized_tweets = []
    for i in tqdm(range(len(tweet_list))):
        if(tweet_list[i] is not NaN):
            tokenized_tweets.append(tokenize(tweet_list[i]))

    for i in tqdm(range(len(tokenized_tweets))):
        tokenized_tweets[i] = [ele.lower() for ele in tokenized_tweets[i] if(not(ele.startswith("@") \
                                                                              or ele.startswith("https") \
                                                                              or ele.lower() in stopwords.words('english') \
                                                                              or ele in string.punctuation + '’‘|…'))]

    cleaned_data = list(zip(tokenized_tweets, author))
    np.save("cleaned_data.npy", cleaned_data)
    for i in range(10):
        print(cleaned_data[i])
```

Random Forest

RANDOM FOREST

```
start_time = time.time()
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(max_depth=10, random_state=0)
rf_clf.fit(X_train, y_train)
print("Time", time.time() - start_time, "s")
```

```
Time 31.250518321990967 s
```

```
pred = rf_clf.predict(X_test)
print(accuracy_score(y_test, pred))
print(classification_report(y_test, pred, labels=labels))
```

```
0.4861021331609567
              precision    recall  f1-score   support

           0       0.53      0.75      0.62       808
           1       0.50      0.63      0.56       807
           2       0.46      0.39      0.42       754
           3       0.43      0.56      0.48       825
           4       0.72      0.18      0.29       528
           5       0.40      0.33      0.36       815
           6       0.42      0.53      0.47       769
           7       0.49      0.74      0.59       804
           8       0.47      0.70      0.56       794
           9       0.83      0.23      0.36       381
          10       0.61      0.17      0.27       685
          11       0.78      0.42      0.55       511
          12       0.46      0.36      0.40       801

    accuracy                           0.49      9282
   macro avg       0.55      0.46      0.46      9282
weighted avg       0.52      0.49      0.47      9282
```

NEURAL NETWORK

```
start_time = time.time()
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, learning_rate_init=0.001, learning_rate='adaptive',
                    hidden_layer_sizes=(384
                                        , 256, 64, 13), random_state=1, verbose=False, max_iter=100)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, pred))
print(classification_report(y_test, pred, labels=labels))
print("Time", time.time() - start_time, "s")
```

```
0.5783236371471666
              precision    recall  f1-score   support

           0       0.80      0.77      0.78       808
           1       0.69      0.48      0.57       807
           2       0.53      0.51      0.52       754
           3       0.47      0.69      0.56       825
           4       0.55      0.48      0.51       528
           5       0.51      0.42      0.46       815
           6       0.44      0.60      0.50       769
           7       0.73      0.68      0.70       804
           8       0.68      0.61      0.65       794
           9       0.69      0.60      0.65       381
          10       0.51      0.45      0.48       685
          11       0.65      0.66      0.65       511
          12       0.49      0.56      0.52       801

    accuracy                           0.58      9282
   macro avg       0.60      0.58      0.58      9282
weighted avg       0.59      0.58      0.58      9282

Time 81.11099171638489 s
```

SVM

```
start_time = time.time()
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
svm_clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
svm_clf.fit(X_train, y_train)
print("Time", time.time() - start_time, "s")
```

```
Time 79.27785778045654 s
```

```
pred = svm_clf.predict(X_test)
print(accuracy_score(y_test, pred))
print(classification_report(y_test, pred, labels=labels))
```

```
0.6412411118293472
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       808
           1       0.68      0.70      0.69       807
           2       0.55      0.62      0.59       754
           3       0.66      0.59      0.62       825
           4       0.64      0.55      0.59       528
           5       0.51      0.58      0.55       815
           6       0.49      0.64      0.55       769
           7       0.74      0.73      0.74       804
           8       0.68      0.72      0.70       794
           9       0.76      0.60      0.67       381
          10       0.64      0.50      0.56       685
          11       0.75      0.67      0.71       511
          12       0.63      0.55      0.59       801

    accuracy                           0.64      9282
   macro avg       0.65      0.64      0.64      9282
weighted avg       0.65      0.64      0.64      9282
```

LOGISTIC REGRESSION

```
start_time = time.time()
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression(random_state=0, max_iter=100).fit(X_train, y_train)
print("Time", time.time() - start_time, "s")
```

```
Time 10.101628065109253 s
```

```
pred = lr_clf.predict(X_test)
print(accuracy_score(y_test, pred))
print(classification_report(y_test, pred, labels=labels))
```

```
0.6026718379659556
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       808
           1       0.65      0.65      0.65       807
           2       0.53      0.55      0.54       754
           3       0.58      0.58      0.58       825
           4       0.57      0.55      0.56       528
           5       0.49      0.47      0.48       815
           6       0.49      0.54      0.52       769
           7       0.70      0.72      0.71       804
           8       0.65      0.68      0.66       794
           9       0.69      0.63      0.66       381
          10       0.56      0.50      0.53       685
          11       0.66      0.67      0.66       511
          12       0.57      0.54      0.56       801

    accuracy                           0.60      9282
   macro avg       0.60      0.60      0.60      9282
weighted avg       0.60      0.60      0.60      9282
```

BERT

- Bidirectional Encoder Representations from Transformers by Google AI
- State-of-the-art for natural language processing tasks
- Bidirectional training of Transformer Architecture
- BERT embeddings represent context between words unlike Glove or Word2Vec
  - Masked Language Modeling
  - Next Sentence Prediction
- Base BERT has 110 million parameters
- Trained on Wikipedia (2.5B words) + BookCorpus (800M words)
- Pre-Trained BERT Sentence Transformers https://www.sbert.net/
- Sources: https://nlp.stanford.edu/seminar/details/jdevlin.pdf, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

BERT sub-tasks

- **Masked Language Modeling** (15% words masked)
  - The dog is running after the red ball in the park.
  - The [mask] is running after the red [mask] in the [mask].
  - Predict words that are masked.

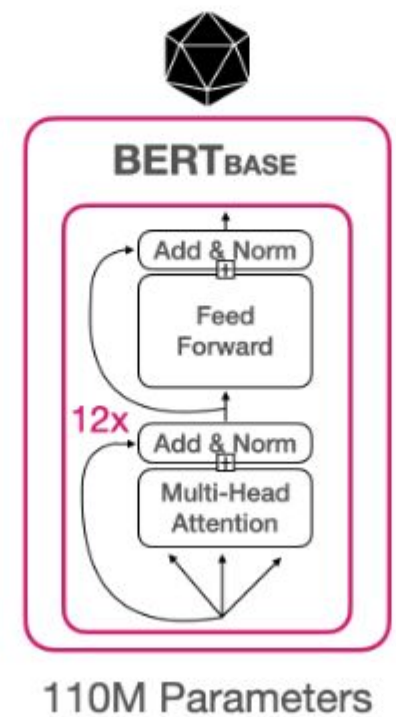
- **Next Sentence Prediction**
  - He took out a jug from the refrigerator. He poured some water in it.
    - The two sentences are logically linked.
  - He took out a jug from the refrigerator. The car is in the garage.
    - The two sentences seem to be independent in terms of context.

BERT Applications

- Question Answering
- Sentiment Analysis
- Text Prediction
- Text Summarization
- Word and Sentence Embeddings
- And more…

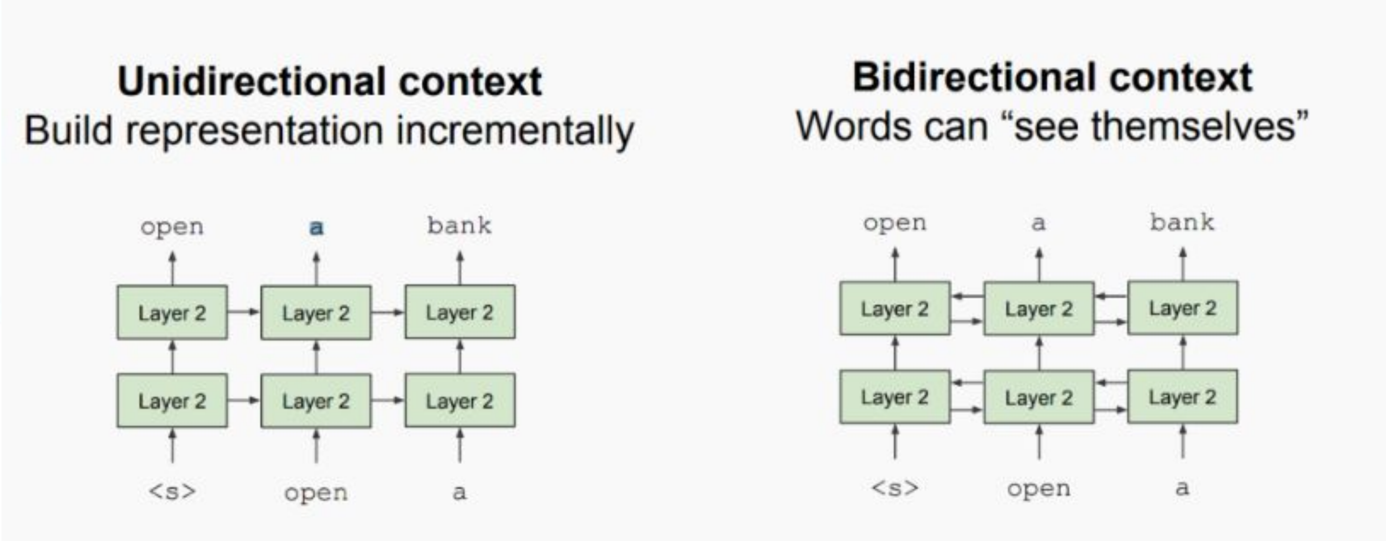
BERT Architecture

- Transformers [Attention mechanism]



Key Components
- Transformer
- Multi-Head Attention
- Normalization

Note: BERT does not use a Decoder.



Src:https://neptune.ai/blog/bert-and-the-transformer-architecture-reshaping-the-ai-landscape

Src: https://huggingface.co/blog/bert-101?text=Oh+so+how+are+you+%5BMASK%5D+today%3F

Thank You!