Covid-19 Fake News Detection

In [17]:
```
import numpy as np
import re
import nltk
from sklearn.datasets import load_files
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB, MultinomialNB, GaussianNB
from sklearn.svm import SVC
import pickle
from nltk.corpus import stopwords
import pandas as pd
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, GRU, LSTM
from IPython.display import Image
```

Data

We took 3127 tweets with twint by filterring Covid-19. In order to get real news we used relaible sources such as:

CNN,washington post,CDCgov,NYGovCuomo,WSJ,NBCNews,NewYorker.

and the other tweets are from random users.

we classified each tweet True or False.

In total we have 2032 True and 1095 False.

In [6]:
```
Tweets_df = pd.read_csv('C:/Users/poop/PycharmProjects/tweeterProject/out - out.csv
',encoding="utf8")
Tweets_df.head(5)
```

Out[6]:

	id	username	text	label
0	0	cnn	. @JakeTapper investigates what really happene...	T
1	1	cnn	A German company working with US pharmaceutica...	T
2	2	cnn	A German company working with US pharmaceutica...	T
3	3	cnn	While many businesses are allowed to reopen in...	T
4	4	cnn	As politicians begin hashing out the next coro...	T

Part 1 - Algorithms

Preprocessing

We took the tweets and cleaned them by removing unnecessary signs such as @,#... and removed stopwords(part of the code of the bag of words).

In [21]:
```
stemmer = WordNetLemmatizer()
def clean_str(s):
    # Remove all the special characters
    sentence = re.sub(r'\W', ' ', str(s))

    # remove all single characters
    sentence = re.sub(r'\s+[a-zA-Z]\s+', ' ', sentence)

    # Remove single characters from the start
    sentence = re.sub(r'\^[a-zA-Z]\s+', ' ', sentence)

    # Substituting multiple spaces with single space
    sentence = re.sub(r'\s+', ' ', sentence, flags=re.I)

    # Removing prefixed 'b'
    sentence = re.sub(r'^b\s+', '', sentence)

    # Converting to Lowercase
    sentence = sentence.lower()

    # Lemmatization
    sentence = sentence.split()

    sentence = [stemmer.lemmatize(word) for word in sentence]
    sentence = ' '.join(sentence)

    return sentence

documents = []
x=list(Tweets_df['text'])
for sen in range(0, len(x)):
    documents.append(clean_str(x[sen]))
documents[:5]
```

Out[21]:
```
['jaketapper investigates what really happened in the s fight against covid 19 d
iscover what he learned watch cnn special report the pandemic the president on s
unday at 10 m et pt pic twitter com ejtm1v7xl5',
 'a german company working with u pharmaceutical giant pfizer ha begun human tri
al of potential covid 19 vaccine that could supply million by the end of the yea
r according to the two firm http cnn it 3anxslb',
 'a german company working with u pharmaceutical giant pfizer ha begun human tri
al of potential covid 19 vaccine that could supply million by the end of the yea
r according to the two firm http cnn it 3brlhfs',
 'while many business are allowed to reopen in georgia some black small business
owner are struggling with the decision especially since black american appear to
be at higher risk when it come to covid 19 http cnn it 2zk6mph',
 'a politician begin hashing out the next coronavirus relief bill debate is brew
ing over whether business should be protected from lawsuit related to covid 19 o
utbreak http cnn it 2ygm4vy']
```

We created from the tweets Bag of words.

In [22]:
```
vectorizer = CountVectorizer(max_features=1500, min_df=2, max_df=0.7, stop_words=st
opwords.words('english'))
X = vectorizer.fit_transform(documents).toarray()
X
```

Out[22]:
```
array([[0, 0, 1, ..., 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]], dtype=int64)
```

from the bag of words we calculated tf-idf.

In [42]:
```
tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
X
```

Out[42]:
```
array([[0.        , 0.        , 0.21489032, ..., 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.        ]])
```

The labels we turned into numbers 0-F and 1-T.

In [41]:
```
y=[]

for label in list(Tweets_df['label']):
    if label is 'T':
        y.append(1)
    else:
        y.append(0)

y= np.array(y)
y[:5]
```

Out[41]:
```
array([1, 1, 1, 1, 1])
```

We split the data to 80% train and 20% test.

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
te=0)
```

The models

1.SVC

In [46]:
```
clf = SVC()
clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, kernel='rbf', ma
x_iter=-1, probability=False,
    random_state=None, shrinking=True, tol=0.001, verbose=False)
y_pred = clf.predict(X_test)
y_pred[:5]
```

Out[46]:
```
array([1, 1, 1, 1, 1])
```

2.Logistic Regression

In [47]:
```
clf2 = LogisticRegression(random_state=0)
clf2.fit(X, y)
y_pred = clf2.predict(X_test)
y_pred[:5]
```

Out[47]:
```
array([0, 1, 1, 1, 1])
```

3.Bernoulli Naive Bayes

In [48]:
```
clf3 = BernoulliNB()
clf3.fit(X, y)
y_pred = clf3.predict(X_test)
y_pred[:5]
```

Out[48]:
```
array([0, 0, 1, 1, 1])
```

4.Multinomial Naive Bayes

In [49]:
```
clf4 = MultinomialNB()
clf4.fit(X, y)
y_pred = clf4.predict(X_test)
y_pred[:5]
```

Out[49]:
```
array([1, 1, 1, 1, 1])
```

5.Gaussian Naive Bayes

In [50]:
```
clf5 = GaussianNB()
clf5.fit(X, y)
y_pred = clf5.predict(X_test)
y_pred[:5]
```

Out[50]:
```
array([0, 0, 1, 0, 1])
```

6.Random forest

In [51]:
```
classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred[:5]
```

Out[51]:
```
array([0, 0, 1, 1, 1])
```

Part 2 - Neural networks

Preprocessing

We cleaned and converted the words into numbers using tokenizer.

In [19]:
```
max_features = 1500
tokenizer = Tokenizer(num_words = max_features, filters='!"#$%&()*+,-./:;<=>?@[\\]^
_`{|}~\t\n', lower = True, split = ' ')
tokenizer.fit_on_texts(texts = Tweets_df['text'])
X = tokenizer.texts_to_sequences(texts = Tweets_df['text'])
```

We applied padding to make them even shaped.

In [8]:
```
X = pad_sequences(sequences = X, maxlen = max_features, padding = 'pre')
X
```

Out[8]:
```
array([[  0,   0,   0, ...,  17,  14,  11],
       [  0,   0,   0, ...,   5,  20,  12],
       [  0,   0,   0, ...,   5,  20,  12],
       ...,
       [  0,   0,   0, ...,   1,  44,   9],
       [  0,   0,   0, ...,  14,  11, 154],
       [  0,   0,   0, ...,  17,  14,  11]])
```

The labels we turned into numbers 0-F and 1-T.

In [11]:
```
y=[]

for label in Tweets_df['label']:
    if label is 'T':
        y.append(1)
    else:
        y.append(0)

y= np.array(y)
y[:5]
```

Out[11]:
```
array([1, 1, 1, 1, 1])
```

We split the data to 80% train and 20% test.

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
tate = 101)
```

The neural networks

1.LSTM

In [15]:
```
lstm_model = Sequential(name = 'lstm_nn_model')
lstm_model.add(layer = Embedding(input_dim = max_features, output_dim = 120, name =
'1st_layer'))
lstm_model.add(layer = LSTM(units = 120, dropout = 0.2, recurrent_dropout = 0.2, na
me = '2nd_layer'))
lstm_model.add(layer = Dropout(rate = 0.5, name = '3rd_layer'))
lstm_model.add(layer = Dense(units = 120,  activation = 'relu', name = '4th_layer
'))
lstm_model.add(layer = Dropout(rate = 0.5, name = '5th_layer'))
lstm_model.add(layer = Dense(units = len(set(y)),  activation = 'sigmoid', name = '
output_layer'))
# compiling the model
lstm_model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', me
trics = ['accuracy','mse'])

lstm_model.summary()

lstm_model_fit = lstm_model.fit(X_train, y_train, epochs = 1)


y_pred = lstm_model.predict(X_test)
y_pred[:5]
```

```
Model: "lstm_nn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
1st_layer (Embedding)        (None, None, 120)         180000
_________________________________________________________________
2nd_layer (LSTM)             (None, 120)               115680
_________________________________________________________________
3rd_layer (Dropout)          (None, 120)               0
_________________________________________________________________
4th_layer (Dense)            (None, 120)               14520
_________________________________________________________________
5th_layer (Dropout)          (None, 120)               0
_________________________________________________________________
output_layer (Dense)         (None, 2)                 242
=================================================================
Total params: 310,442
Trainable params: 310,442
Non-trainable params: 0
_________________________________________________________________
79/79 [==============================] - 352s 4s/step - loss: 0.5441 - accuracy:
0.7101 - mse: 0.3102
```

Out[15]:
```
array([[0.7491629 , 0.29439327],
       [0.06180692, 0.8019961 ],
       [0.03575563, 0.8542671 ],
       [0.14941335, 0.69379896],
       [0.0830026 , 0.7669484 ]], dtype=float32)
```

2.GRU

In [18]:
```
gru_model = Sequential(name = 'gru_nn_model')
gru_model.add(layer = Embedding(input_dim = max_features, output_dim = 120, name =
'1st_layer'))
gru_model.add(layer = GRU(units = 120, dropout = 0.2,
                          recurrent_dropout = 0.2, recurrent_activation = 'relu',
                          activation = 'relu', name = '2nd_layer'))
gru_model.add(layer = Dropout(rate = 0.4, name = '3rd_layer'))
gru_model.add(layer = Dense(units = 120, activation = 'relu', name = '4th_layer'))
gru_model.add(layer = Dropout(rate = 0.2, name = '5th_layer'))
gru_model.add(layer = Dense(units = len(set(y)), activation = 'softmax', name = 'ou
tput_layer'))
# compiling the model
gru_model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', met
rics = ['accuracy','mse'])

gru_model.summary()

gru_model_fit = gru_model.fit(X_train, y_train, epochs = 1)

y_pred = gru_model.predict(X_test)
y_pred[:5]
```

```
Model: "gru_nn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
1st_layer (Embedding)        (None, None, 120)         180000
_________________________________________________________________
2nd_layer (GRU)              (None, 120)               87120
_________________________________________________________________
3rd_layer (Dropout)          (None, 120)               0
_________________________________________________________________
4th_layer (Dense)            (None, 120)               14520
_________________________________________________________________
5th_layer (Dropout)          (None, 120)               0
_________________________________________________________________
output_layer (Dense)         (None, 2)                 242
=================================================================
Total params: 281,882
Trainable params: 281,882
Non-trainable params: 0
_________________________________________________________________
79/79 [==============================] - 313s 4s/step - loss: 0.5444 - accuracy:
0.6957 - mse: 0.3044
```

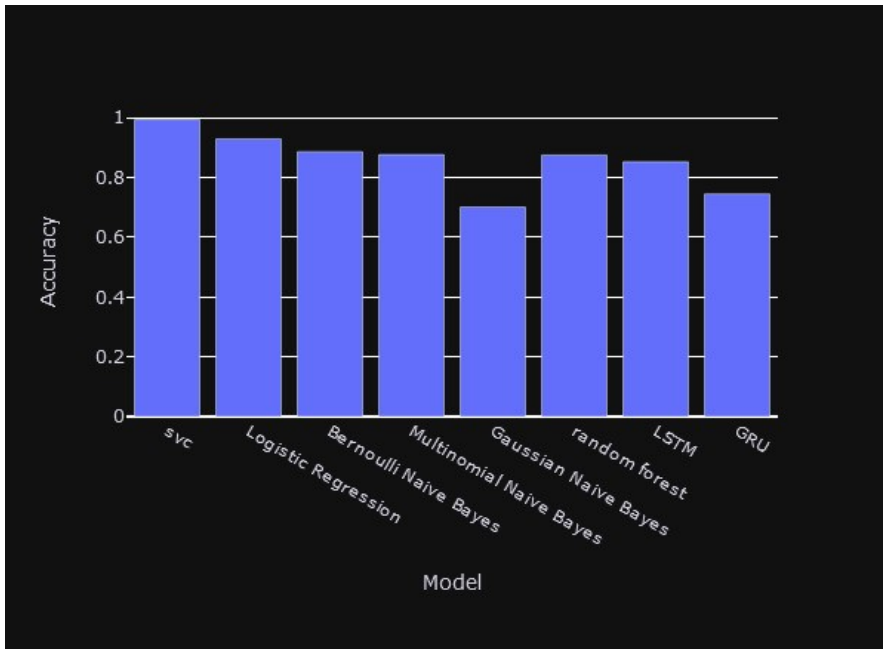
Out[18]:
```
array([[0.52957433, 0.4704256 ],
       [0.0194577 , 0.9805423 ],
       [0.01945455, 0.9805455 ],
       [0.14453873, 0.8554613 ],
       [0.16135752, 0.8386425 ]], dtype=float32)
```

Results

Model	Accuracy	F1-Score	MSE
svc	0.9936102236421726	0.99	0.006389776357827476
Logistic Regression	0.9297124600638977	0.93	0.07028753993610223
Bernoulli Naive Bayes	0.8865814696485623	0.89	0.1134185303514377
Multinomial Naive Bayes	0.8769968051118211	0.87	0.12300319488817892
Gaussian Naive Bayes	0.7012779552715654	0.70	0.2987220447284345
Random forest	0.8753993610223643	0.88	0.12460063897763578
LSTM	0.8530351519584656	0.85	0.3692074716091156
GRU	0.7451757001876831	0.74	0.3442157208919525

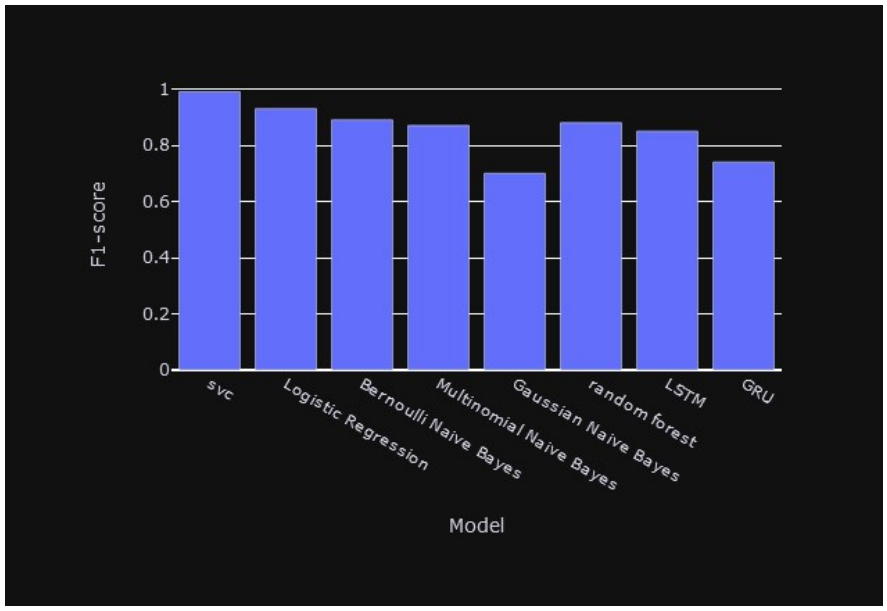
In [27]:
```
Image(filename = "C:/Users/poop/PycharmProjects/tweeterProject/acuuracy.jpeg", width=500, height=500)
```

Out[27]:



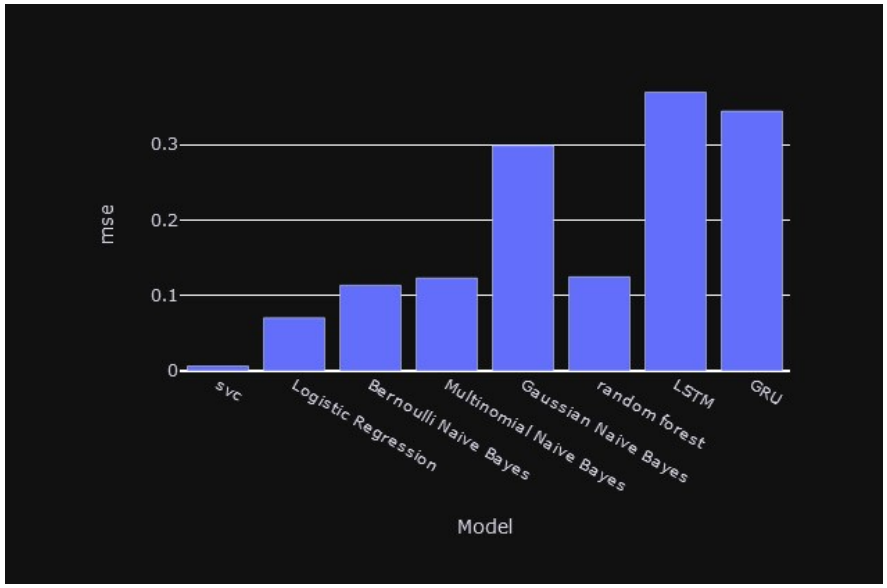
In [26]:
```
Image(filename = "C:/Users/poop/PycharmProjects/tweeterProject/f1.jpeg", width=500, height=500)
```

Out[26]:



In [28]:
```
Image(filename = "C:/Users/poop/PycharmProjects/tweeterProject/mse.jpeg", width=500, height=500)
```

Out[28]:



conclusions

After comparing all the models and networks the best was SVC with the best accuracy,f1-score and mse for classifying fake news.