IEOR 142 Final Report
Battling COVID-19 Falsehoods with Machine Learning

Ahmet Turunc, Catherine Lei, Pranav Viswanathan,
Vishnu Karukonda, Wako Morimoto

Motivation

With the rise of coronavirus disease (COVID-19), global social media usage has increased alongside consistent misinformation surrounding public health and COVID-19 (Naem et al. 2021). In fact, false narratives surrounding everything from the origin of the virus itself to fake cures and deadly preventatives have been able to freely run rampant on top social media platforms like Facebook, Twitter, and Instagram. Misinformation in the form of conspiracy theories about the coronavirus disease have even shown to significantly affect political orientation, turnout, and voting. For instance, a recent study on a representative sample of 3019 individuals in the United States found that nearly 85% of participants believed in one or more COVID-19 conspiracy theories (Agley and Xiao, 2021). As a result, content moderation on social media platforms has taken center stage in the global conversation surrounding efforts to mitigate COVID-19 and public health misinformation spread. While companies like Twitter have implemented systems targeted at combating this misinformation, efforts to minimize its spread have been largely lacking in impact, with several inaccurate posts surrounding COVID-19 gaining wide popularity on platforms (Kouzy et al. 2020). In this report, we aim to combat this problem by creating classification models that predict if a given short text post contains COVID-19 related falsehoods. Such classifiers may be used in flagging short text social media posts for further review.

Data

Background and Sources: For our data, we primarily relied on two different sources. One data set contains several COVID-19 related tweets and short text posts with post timestamps, locations, and occasional "True", "False", or "Unverified" post labels based on fact checking from fact-checking entities. Our second data set is a COVID-19 data set that contains metrics related to daily vaccinations, hospitalizations, available ICU beds, new cases, etc.
Tweet Data Set: https://github.com/MickeysClubhouse/COVID-19-rumor-datase
OWID COVID-19 Data Set: https://github.com/owid/covid-19-data/tree/master/public/data

Data Selection & NLP Cleaning: In the COVID-19 rumor dataset, we removed all non-tweet sources, such as news headlines and text from other websites, as well as unverified and missing labels. We then used the NLTK library to perform common NLP cleaning steps such as removing all digits, capitalization, punctuation, and stop words. When removing punctuation, we decided to keep the "#" symbol since hashtags have an indexing purpose on Twitter. Hence, tweets with certain hashtags may reach more people, making it important to differentiate between words when hashtagged or not. We tokenized and stemmed the cleaned text, and then created a

document-term matrix using CountVectorizer (scikit-learn). The time of each tweet and a True or False label were added to each row of document-term matrix.

Feature Selection: Within the OWID dataset, we decided to only use data about the United States. We made this choice because all tweets in our NLP dataset are in English, and over 77% of Twitter users from English speaking countries are based in the United States (Statista, 2021). We also removed features that were cumulative counts, as the number would always increase, and repetitive features. When looking through the dataset, we noticed that many features had daily values, as well as "smoothed" 7-day averages. We experimented with using one, the other, and both. Using both had the best performance for most models, likely due to each column supplementing missing data in the other.

Readability Scores: Several scores for quantifying the readability of text exist. We decided to experiment with multiple readability scores, such as Flesch Reading Ease and the SMOG Index. We found that the Dale-Chall readability formula was the most straightforward readability metric to use with our tweet data (Dale and Chall, 1949):

$$readability = 15.79\frac{\#\ of\ difficult\ words}{sentence} + 0.0496\frac{\#\ of\ words}{\#\ of\ sentences}$$

A visualization of Dale-Chall readability distributions for True and False tweets can be found in our appendix.

Merge Preparation: The only commonality between the COVID-19 rumor dataset and the OWID COVID-19 dataset was a time column. However, times were in different formats, even within each dataset. The COVID-19 rumor dataset used 3 different time formats, along with having minor errors in the data collection such as missing digits or random words. We removed all items that did not have a timestamp in the NLP-prepared COVID-19 rumor dataset and unified the time formats to allow for merging with the US COVID-19 feature data we isolated. We then added a readability score to each tweet's representative feature vector.

Cleaned Dataset: There were 1775 tweets represented in our final curated dataset. Each tweet is described by 234 features, consisting of NLP tokens, US COVID-19 data indexed by tweet date, and a readability score for the tweet. We used a 80/20 Train-Test split, with our training set having 1,420 tweets (939 True, 481 False) and our test set having 355 tweets (255 True, 100 False).

Analytics Models

Model Choice and Training Priorities: We decided to test the performance of all classification models learned in IEOR 142 — Logistic Regression, Linear Discriminant Analysis (LDA), Random Forests, and Gradient Boosting. With Logistic Regression, we experimented with different regularization methods (LASSO, Ridge, and Elastic Net). For Random Forests and Gradient Boosting, cross-validation was used to determine several parameters; we chose whichever combinations of cross-validated parameters resulted in the lowest False Positive Rate (FPR). Specifics about training algorithms (e.g. gradient descent) and cross validation procedures can be found in the appendix. Since we are particularly concerned with accidentally labeling misinformation as truthful (i.e. classifying a tweet as true when it is actually false), we prioritized FPR minimization when performing cross-validation to determine model parameters. The Area Under the Curve (AUC) for Receiver Operator Curves (ROC) of models is also an important consideration since AUC quantifies the trade off between FPR and TPR. Accordingly, we hoped to train models that have both a low FPR and relatively high ROC AUC on test data.

Model Results and Performance:

Figure 1. ROC curves for all trained models

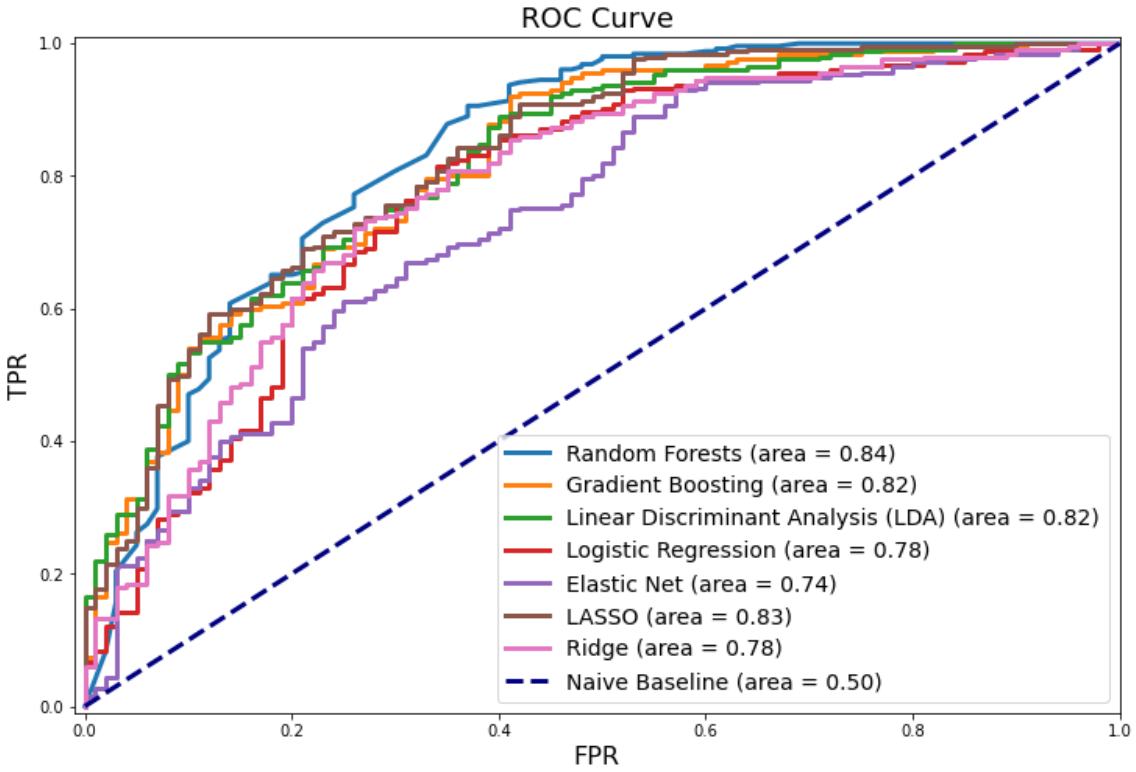


Fig. 1 illustrates ROC curves for the various models trained. Curves that conform best to the top left corner of the ROC grid reflect general pairs of high model TPR and low FPR with different label probability cutoffs. Ultimately, an ideal model would conform perfectly to the top left corner of the ROC grid, making an inverted "L" shape. It is worth noting that the Elastic Net Logistic Regression model conformed the least to the top left corner of the ROC curve,

indicating higher rates of false positives (FPR) as the true positive rate (TPR) increases. With visual inspection, the models that best conform to the top left corner of the graph are: Random Forests, Gradient Boosting, LDA, and LASSO-regularized Logistic Regression.

Table 1. Model performance on the testing data set (355 observations; 255 True, 100 False)

	Random Forest	Gradient Boosting	LDA	Logistic Regression	Elastic Net	LASSO	Ridge
Accuracy	0.84	0.80	0.81	0.79	0.73	0.81	0.79
AUC	0.84	0.82	0.82	0.78	0.74	0.83	0.78
FPR	0.41	0.41	0.41	0.6	0.51	0.43	0.57
TPR	0.94	0.89	0.89	0.95	0.83	0.91	0.93

NOTE: all Table 1 values fell in 95% Confidence Intervals from metric bootstrapping (n=5000)

Table 1 shows the various testing data set performance metrics calculated for the models trained. As noted below the table, all performance metrics observed on our testing set fell into the two-tailed 95% confidence interval from bootstrapping metrics. Accordingly, we are quite confident in the results presented by the table.

Best Performing Models: Based on our criteria of minimizing FPR while also maintaining a relatively high AUC, it seems that our models for Random Forests, Gradient Boosting, Linear Discriminant Analysis, and LASSO-regularized Logistic Regression all performed similarly well with relatively low FPRs near 0.4 and high AUCs of 0.83 +/- 0.01.

Impact

Current Work: The models presented in this report can be deployed on social media platforms in order to flag short text posts for further review if a post gets labeled as False by our models. Models like ours can be further used in efforts to identify and minimize the spread of COVID-19 misinformation and ultimately help in protecting public health.

Future Directions: As multiple models were high performing, it makes sense to experiment with ensemble models built on our existing models. In addition, as misinformation is not present only on Twitter, a classifier that caters to various platforms is more impactful. To adequately counter misinformation on social media platforms, it is not sufficient to only focus on text. Instead, future models should expand to cover images, videos and other forms of media. Such models could employ popular neural network image classification architectures such as convolutional neural networks (CNNs). As misinformation is rampant in many other countries, our analysis can be modified to focus on other languages too. Currently our analysis is focused on COVID-19 related misinformation. Going forward, future analyses might also focus on other categories of misinformation.

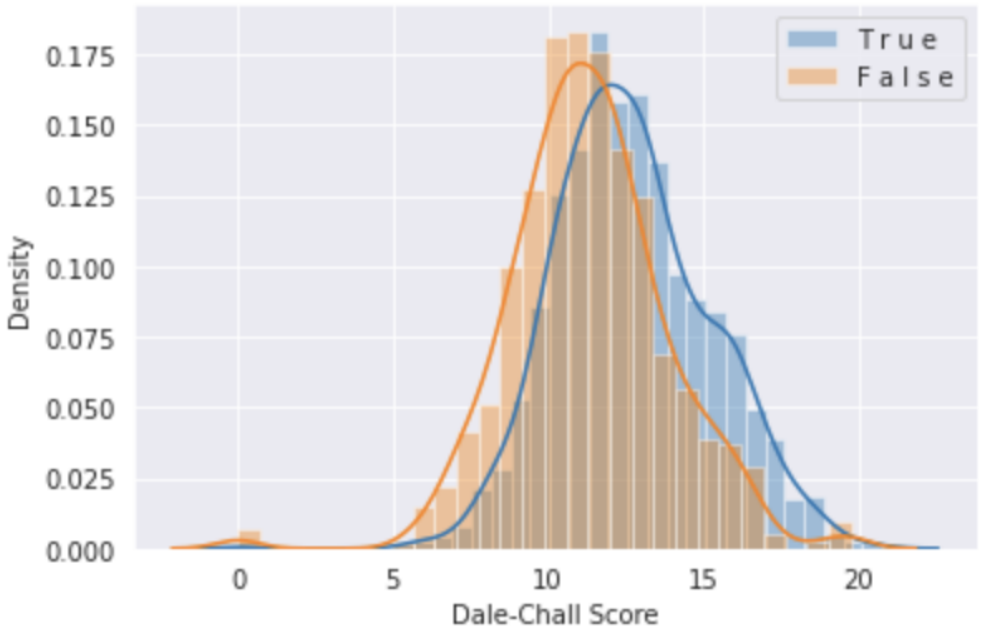
Appendix

References:

1. Naeem, Salman Bin et al. "An exploration of how fake news is taking over social media and putting public health at risk." *Health information and libraries journal* vol. 38,2 (2021): 143-149. doi:10.1111/hir.12320
2. Kouzy, Ramez et al. "Coronavirus Goes Viral: Quantifying the COVID-19 Misinformation Epidemic on Twitter." *Cureus* vol. 12,3 e7255. 13 Mar. 2020, doi:10.7759/cureus.7255
3. Agley, J., Xiao, Y. Misinformation about COVID-19: evidence for differential latent profiles and a strong association with trust in science. *BMC Public Health* 21, 89 (2021). https://doi.org/10.1186/s12889-020-10103-x
4. Published by Statista Research Department, and Nov 19. "Twitter: Most Users by Country." *Statista*, 19 Nov. 2021, https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/.
5. Dale, Edgar, and Jeanne S. Chall. "The Concept of Readability." *Elementary English*, vol. 26, no. 1, National Council of Teachers of English, 1949, pp. 19–26, http://www.jstor.org/stable/41383594.

Supplemental Diagrams and Visualizations:

1. Distribution Plot of Readability Scores between True and False Tweets.



GridSearchCV Specifics - RandomForestClassifier and GradientBoostingClassifier:

- RandomForestClassifier
    - 5 Fold GridSearch Cross Validation
        - Scoring Criteria: False Positive Rate minimization
    - Parameter Grid:
        - 'max_features': [20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190]
    - Parameter value chosen by CV: 40
- GradientBoostingClassifier
    - 5 Fold GridSearch Cross Validation
        - Scoring Criteria: False Positive Rate minimization
    - Parameter Grid:
        - 'learning_rate': [0.1, 0.2, 0.3, 0.4]
        - 'n_estimators': [100, 280, 460, 640, 820]
    - Parameters values chosen by CV: 0.4, 460 respectively

Logistic Regression - Regularized Model Training:

- Ridge and LASSO: Used scikit-learn's 'liblinear' solver, 1000 iterations allowed for training
    - 'Library for Large Linear Classification" is a coordinate descent algorithm from the Machine Learning Group at National Taiwan University
        - https://www.csie.ntu.edu.tw/~cjlin/liblinear/
- Elastic Net: Used 'SAGA' solver, 1000 iterations allowed for training; 50/50 mix between Ridge and LASSO regularization
    - 'SAGA' is a stochastic gradient optimization algorithm
        - https://www.di.ens.fr/~fbach/Defazio_NIPS2014.pdf

CovidRumors_142FinalProject

December 17, 2021

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[5]:
```
```

IEOR 142 Final Project - Battling COVID-19 Falsehoods with Machine Learning Approaches

Ahmet Turunc, Catherine Lei, Pranav Viswanathan, Vishnu Karukonda, Wako Morimoto

Data Sources

Tweet and text data: https://github.com/MickeysClubhouse/COVID-19-rumor-dataset

COVID case and vaccination data by country: https://github.com/owid/covid-19-data/tree/master/public/data

[6]:
```
from google.colab import drive
drive.mount('/content/gdrive/')
import os
os.chdir("/content/gdrive/My Drive/IEOR 142 Final Project")
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call
drive.mount("/content/gdrive/", force_remount=True).
```

0.1 Read in data

text_data: tweets and news articles with some additional values

case_data: COVID case data from OWID resource

[7]:
```
text_data = pd.read_csv('en_dup.csv')
case_data = pd.read_csv('owid-covid-data.csv')
```

0.2 Filtering data that has T/F labels, and a date

[8]:
```
filter_data = text_data[text_data['label'].isnull()==False]
filter_data = filter_data[filter_data["label"] != "U"]
filter_data = filter_data[filter_data['time'].isnull()==False]
filter_data.index = np.arange(1, len(filter_data) + 1)
filter_data
```

[8]:
```
     label  …                              time
1        T  …  Thu Feb 13 08:04:47 +0000 2020
2        T  …  Sun Feb 09 13:59:29 +0000 2020
3        T  …  Sun Feb 16 18:52:51 +0000 2020
4        T  …  Tue Jan 28 15:16:02 +0000 2020
5        T  …  Tue Feb 11 23:25:29 +0000 2020
…    …  …                                 …
1771     T  …  Thu Jan 30 14:28:56 +0000 2020
1772     T  …  Wed Mar 04 20:14:48 +0000 2020
1773     T  …           8:45 PM · Mar 8, 2020
1774     F  …          3:57 PM · Feb 10, 2020
1775     F  …         10:00 AM · Feb 17, 2020

[1775 rows x 5 columns]
```

["time"] columns

###As seen below, time is stored in multiple different formats. We want time to match our "Case_data" dataset, so we want to reformat time to a yyyy-mm-dd format

[9]:
```
filter_data["time"].tail(10)
```

[9]:
```
1766                          2020/4/4
1767                          2020/4/7
1768                          2020/4/7
1769                          2020/4/7
1770    Mon Jan 27 12:30:00 +0000 2020
1771    Thu Jan 30 14:28:56 +0000 2020
1772    Wed Mar 04 20:14:48 +0000 2020
1773              8:45 PM · Mar 8, 2020
1774             3:57 PM · Feb 10, 2020
1775            10:00 AM · Feb 17, 2020
Name: time, dtype: object
```

[10]:
```
def time_fixer(df):
  df = df.copy()

  #Making all yyyy/mm/dd into yyyy-mm-dd
  x = df["time"].str.replace("/","-")
```

```
 #Splitting into columns accesible by RangeIndex
 y = x.str.split(expand = True)
 y = y.fillna(value = 0)

 #Column 4 - Removing commas at the end of days + Replacing values in other␣
↪formats with 0
 y[4] = y[4].replace("+0000", 0)
 y[4] = y[4].str.replace(',','')

 #Columns 3 - Remapping months, and removing values with :
 y[3] = np.where(y[3].str.contains(":", na=False), 0, y[3])
   #These are the only month values in y[4]
 m2d = {"Jan": "01", "Feb": "02", "Mar": "03", "Oct": "10"}
 y[3] = y[3].map(m2d)

 #Column 0 - Removing values without -
 y[0] = np.where(~y[0].str.contains("-", na=False), 0, y[0])

 #Column 1 - Remapping months, Replacing AM/PM with NaN
 m2d2 = {"Jan": "01", "Feb": "02", "Mar": "03", "Oct": "10"}
 y[1] = y[1].map(m2d2)

 #creating a list of nans or those already in the correct format
 y_arr = np.array(y[0])
 y_arr


 #creating a list that has the correct format for things in the AM/PM ...␣
↪format and nans for all other values

 join_list = np.array([])
 for i in range(1,1776):
   if y[5][i] != 0:
     if pd.isnull(y[4][i]) == False:
       if pd.isnull(y[3][i]) == False:
         format = y[5][i] + "-" + y[3][i] + "-" + y[4][i]
         join_list = np.append(join_list, format)
   if len(join_list) != i:
     join_list = np.append(join_list, "none")

 #creating a list that has the correct format for the final format

 join_list_2 = np.array([])
 for i in range(1,1776):
   if y[5][i] != 0:
     if pd.isnull(y[1][i]) == False:
       if pd.isnull(y[2][i]) == False:
```

```
        format = y[5][i] + "-" + y[1][i] + "-" + y[2][i]
        join_list_2 = np.append(join_list_2, format)
    if len(join_list_2) != i:
      join_list_2 = np.append(join_list_2, "none")

  #making all lists have the same "null values"
  y_arr = y_arr.astype(str)
  y_arr = np.where(y_arr =="0", "none", y_arr)
  y_arr

  #merging the three lists to make a final list of the correct formats
  full_list = np.array([])
  for i in range(1775):
    if y_arr[i] != "none":
      full_list = np.append(full_list, y_arr[i] )
    elif join_list[i] != "none":
      full_list = np.append(full_list, join_list[i] )
    elif join_list_2[i] != "none":
      full_list = np.append(full_list, join_list_2[i] )
    else:
      full_list = np.append(full_list, "none")

  #Some we want month values to be 0x instead of x (03 instead of 3) -- same␣
↪thing for days.
  final_list = np.array([])
  for i in full_list:
    if i != "none":
      z = i.split("-")
    if len(z[1]) == 1:
      z[1] = '0' + z[1]
    if len(z[2]) == 1:
      z[2] = '0' + z[2]
    combined = z[0] + "-" + z[1] + "-" + z[2]
    final_list = np.append(final_list, combined)
  return final_list
```

[11]:
```
time_reform = time_fixer(filter_data)
```

[12]:
```
#Do we have any "none" values? Nope!
time_reform[time_reform == "none"]
```

[12]:
```
array([], dtype='<U32')
```

[13]:
```
'''
Looking through y, we have 2 wierd data entries:

'2020·Sprinklr-03-03'
```

```
'202-01-22'

'''
np.where(time_reform == '2020·Sprinklr-03-03'), np.where(time_reform ==␣
↪'202-01-22')
```

[13]:
```
((array([547]),), (array([887]),))
```

[14]:
```
filter_data.iloc[547], filter_data.iloc[887]
```

[14]:
```
(label                                                      T
 content    Health experts are widely skeptical of the num…
 source                                                   NaN
 author                                                  TIME
 time                          7:32 AM · Mar 3, 2020·Sprinklr
 Name: 548, dtype: object,
 label                                                      T
 content    Breaking: The CDC is expected to announce the …
 source                                                   NaN
 author                                     PM Breaking News
 time                                    2:26 AM · Jan 22, 202
 Name: 888, dtype: object)
```

[15]:
```
time_reform = np.where(time_reform == '2020·Sprinklr-03-03', "2020-03-03",␣
↪time_reform)
#https://twitter.com/PMBreakingNews/status/1219687852486942721 found tweet to␣
↪verify date
time_reform = np.where(time_reform == '202-01-22', "2020-01-22", time_reform)
filter_data["time"] = time_reform
filter_data
```

[15]:
```
      label  …        time
1         T  …  2020-02-13
2         T  …  2020-02-09
3         T  …  2020-02-16
4         T  …  2020-01-28
5         T  …  2020-02-11
…     …  …  …         …
1771      T  …  2020-01-30
1772      T  …  2020-03-04
1773      T  …  2020-03-08
1774      F  …  2020-02-10
1775      F  …  2020-02-17

[1775 rows x 5 columns]
```

0.3 NLP Text Cleaning

[16]:
```
#FUNCTIONS FROM LABS
from string import punctuation

def remove_punctuation(document):

    punct2 = punctuation.replace('#','')
    no_punct = ''.join([character for character in document if character not in␣
 ↪punct2])

    return no_punct

def remove_digit(document):

    no_digit = ''.join([character for character in document if not character.
 ↪isdigit()])

    return no_digit

import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

def remove_stopwords(document):

    words = [word for word in document if not word in stop_words]

    return words

from nltk.stem import PorterStemmer

porter = PorterStemmer()

def stemmer(document):

    stemmed_document = [porter.stem(word) for word in document]

    return stemmed_document

from nltk.tokenize.treebank import TreebankWordDetokenizer
from sklearn.feature_extraction.text import CountVectorizer
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
```

```
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[17]:
```
#inspect raw data
{print(x) for x in filter_data.head()['content']}
```

```
Only few people will stay in campus tho?? https://t.co/Aq5huubeMB
#coronavirusuk Coronavirus: Fourth patient in UK diagnosed - BBC News
https://t.co/lI4pa7jQjX
@TPE_connect Yet later proved true even though the initial study was flawed. One
example: https://t.co/GORnTkfPBv
Centre for Health Protection details on the eight confirmed cases of the Wuhan
coronavirus in #HongKong, as of 4pm Tuesday. https://t.co/h69d7j7UXK
#China reported that 94 more people died on Tuesday due to the #coronavirus,
raising the death toll in the country to at least 1,105. #Wuhan #COVID19
#WuhanCoronavirus https://t.co/hDNcPqXywM
```

[17]:
```
{None}
```

Proposed text cleaning steps: remove t.co links, remove twitter @ mentions, COVID mentions, followed by typical NLP steps from lecture

Typical NLP cleaning steps

1) convert all chars to lowercase
2) remove all punctuation
3) remove any numeric values
4) tokenize text entries
5) remove stopwords from tokenized text
6) Stem all words
7) Detokenize text and use countvectorizer to create document-term matrix

[18]:
```
#Convert to lowercase, remove t.co media links and remove all twitter @ mentions
pre_cleaned_text = filter_data['content'].str.lower().str.replace('@[^ .]+',␣
↪'',regex=True).str.replace('https://t.co/[A-Za-z0-9]+', '',regex=True)
{print(x) for x in pre_cleaned_text.head()}
```

```
only few people will stay in campus tho??
#coronavirusuk coronavirus: fourth patient in uk diagnosed - bbc news
 yet later proved true even though the initial study was flawed. one example:
centre for health protection details on the eight confirmed cases of the wuhan
coronavirus in #hongkong, as of 4pm tuesday.
#china reported that 94 more people died on tuesday due to the #coronavirus,
raising the death toll in the country to at least 1,105. #wuhan #covid19
#wuhancoronavirus
```

[18]: {None}

[19]:
```
#Typical steps as performed in lab 8b and the NLP HW
punctless = pre_cleaned_text.apply(remove_punctuation)
digitless = punctless.apply(remove_digit)
tokens = digitless.apply(word_tokenize)
stopless = tokens.apply(remove_stopwords)
only_stems = stopless.apply(stemmer)
concat_text = only_stems.apply(TreebankWordDetokenizer().detokenize)
ct_vec = CountVectorizer(min_df=0.01) #ARBITRARILY CHOSEN MIN_DF; COME BACK TO␣
↪DECIDE ON A VALUE
sparse = ct_vec.fit_transform(concat_text)
text_features = pd.DataFrame(sparse.toarray(), columns=ct_vec.
↪get_feature_names(), index=pre_cleaned_text.index)
text_features
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function get_feature_names is deprecated; get_feature_names is
deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out
instead.
  warnings.warn(msg, category=FutureWarning)
```

[19]:

	accord	affect	airport	allow	…	wuhan	wuhancoronaviru	year	yet
1	0	0	0	0	…	0	0	0	0
2	0	0	0	0	…	0	0	0	0
3	0	0	0	0	…	0	0	0	1
4	0	0	0	0	…	1	0	0	0
5	0	0	0	0	…	1	1	0	0
…	…	…	…	…	…	…	…	…	…
1771	0	0	0	1	…	0	0	0	0
1772	0	0	0	0	…	0	0	0	0
1773	0	0	0	0	…	0	0	0	0
1774	0	0	0	0	…	0	0	0	0
1775	0	0	0	0	…	0	0	0	0

[1775 rows x 200 columns]

[20]:
```
#Create merged table with labels, date, and text_features
text_final = text_features.copy()
text_final['label'] = filter_data['label']
text_final['date']= filter_data['time']
text_final.head()
```

[20]:

	accord	affect	airport	allow	…	year	yet	label	date
1	0	0	0	0	…	0	0	T	2020-02-13
2	0	0	0	0	…	0	0	T	2020-02-09
3	0	0	0	0	…	0	1	T	2020-02-16

```
4       0       0        0      0  …     0    0      T  2020-01-28
5       0       0        0      0  …     0    0      T  2020-02-11

[5 rows x 202 columns]
```

[21]:
```
text_final['label']
```

[21]:
```
1       T
2       T
3       T
4       T
5       T
       ..
1771    T
1772    T
1773    T
1774    F
1775    F
Name: label, Length: 1775, dtype: object
```

##Filtering covid data set to get per capita features from different income groups

The case data groups several countries into income categories and contains covid statistics across those groups of countries; for the sake of convenience, these entries can be used from every date.

[22]:
```
#PREVIEWING CASE DATA
case_data.head(5)
```

[22]:
```
  iso_code continent  … excess_mortality
excess_mortality_cumulative_per_million
0      AFG      Asia  …               NaN
NaN
1      AFG      Asia  …               NaN
NaN
2      AFG      Asia  …               NaN
NaN
3      AFG      Asia  …               NaN
NaN
4      AFG      Asia  …               NaN
NaN

[5 rows x 67 columns]
```

[23]:
```
#Getting all US features
US_cases = case_data[case_data['location'] == 'United States']

#features that maybe do not add much value (daily cumulative counts primarily)
```

```
features_to_drop =␣
↪['total_cases','total_cases_per_million','total_deaths_per_million','total_tests','total_te
                  ␣
↪'total_vaccinations_per_hundred','total_boosters_per_hundred','population_density','median_
                  ␣
↪'cardiovasc_death_rate','diabetes_prevalence','female_smokers','male_smokers','handwashing_
                  ␣
↪'excess_mortality_cumulative_absolute','excess_mortality_cumulative','excess_mortality','ex
↪'tests_units']
filtered_US_features = US_cases.drop(features_to_drop, axis=1).
↪drop(['iso_code', 'continent', 'location'],axis=1)
```

##Merge covid case data with text feature data set based on date

[24]:
```
text_final.head()
```

[24]:
```
   accord  affect  airport  allow  …  year  yet  label        date
1       0       0        0      0  …     0    0      T  2020-02-13
2       0       0        0      0  …     0    0      T  2020-02-09
3       0       0        0      0  …     0    1      T  2020-02-16
4       0       0        0      0  …     0    0      T  2020-01-28
5       0       0        0      0  …     0    0      T  2020-02-11

[5 rows x 202 columns]
```

[25]:
```
filtered_US_features.head()
```

[25]:
```
              date  …  new_people_vaccinated_smoothed_per_hundred
130487  2020-01-22  …                                         NaN
130488  2020-01-23  …                                         NaN
130489  2020-01-24  …                                         NaN
130490  2020-01-25  …                                         NaN
130491  2020-01-26  …                                         NaN

[5 rows x 33 columns]
```

[26]:
```
combined_data = text_final.merge(filtered_US_features, how = 'left', left_on =␣
↪'date', right_on = 'date').drop(['date', 'date'], axis = 1)
combined_data.tail()

#combined_data = combined_data.replace(0.000, np.nan)
```

[26]:
```
      accord  …  new_people_vaccinated_smoothed_per_hundred
1770       0  …                                         NaN
1771       0  …                                         NaN
1772       0  …                                         NaN
1773       0  …                                         NaN
```

```
1774       0  …                                                NaN

[5 rows x 233 columns]
```

We notice that between each column and its "smoothed" version, values of NaN are represented as 0. This is an issue, as seen when looking ath the new_deaths and new_deaths_smoothed column. Having 0 deaths over a 7 day period is far different from having an unreported amount of deaths for 7 consecutive days. We will use the smooth features... (explain after discussing)

[27]:
```
#Dropping features that have a smooth column or a per_million/per thousand/per␣
↪hundred col
filtered_US_features_smooth = filtered_US_features.
↪drop(['new_cases','new_cases_smoothed','new_deaths',
                                                        ␣
↪'new_cases_per_million','new_deaths_per_million',
                                                        ␣
↪'icu_patients','hosp_patients','weekly_icu_admissions',
                                                        ␣
↪'weekly_hosp_admissions','new_tests','new_tests_per_thousand',
                                                        ␣
↪'new_tests_smoothed','new_vaccinations','new_vaccinations_smoothed',␣
↪'new_people_vaccinated_smoothed'],axis=1)
filtered_US_features_smooth
```

[27]:
```
             date  …  new_people_vaccinated_smoothed_per_hundred
130487  2020-01-22  …                                         NaN
130488  2020-01-23  …                                         NaN
130489  2020-01-24  …                                         NaN
130490  2020-01-25  …                                         NaN
130491  2020-01-26  …                                         NaN
…              …  …                                          …
131167  2021-12-02  …                                       0.094
131168  2021-12-03  …                                       0.097
131169  2021-12-04  …                                       0.095
131170  2021-12-05  …                                       0.090
131171  2021-12-06  …                                       0.075

[685 rows x 18 columns]
```

[28]:
```
#combined_data_smooth = text_final.merge(filtered_US_features, how = 'left',␣
↪left_on = 'date', right_on = 'date').drop(['date', 'date'], axis = 1)
#combined_data_smooth = combined_data_smooth.replace(0.000,np.nan)
#combined_data_smooth = combined_data_smooth.fillna(0) Slightly raise
```

[76]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

#IMPUTE IN MEAN VALUE FOR MISSING VAL
imputer = SimpleImputer()
imputed_data = imputer.fit_transform(combined_data.drop(['label'], axis=1))
#imputed_data = imputer.fit_transform(combined_data_smooth.drop(['label'],␣
 ↪axis=1))
labels = combined_data['label'].str.replace('T','1').str.replace('F','0').
 ↪astype(int)
#labels = combined_data_smooth['label'].str.replace('T','1').str.
 ↪replace('F','0').astype(int)
imputed_data
```

[76]:
```
array([[ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 12.1267    ],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 16.42745385],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 13.30152222],
       …,
       [ 0.        ,  0.        ,  0.        , …,  0.06      ,
        16.7       , 10.13831887],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497,  9.64383333],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 10.74815   ]])
```

Readibility Scoring Tweets

We want to see if there is a difference in the readibility level of text between True and False tweets. However, most common readibility scores, such as the use Flesch and Gunning-fog use sentence related features in their analysis. Since tweets, unlike news articles or headlines have no expectation of being written well, it may be hard to identify sentences. Hence, we will use the smog formula.

SMOG grading $= 3 + \sqrt{(\text{polysyllable count})}$

[33]:
```
pip install https://github.com/andreasvc/readability/tarball/master
```

```
shell-init: error retrieving current directory: getcwd: cannot access parent
directories: No such file or directory
shell-init: error retrieving current directory: getcwd: cannot access parent
directories: No such file or directory
Collecting https://github.com/andreasvc/readability/tarball/master
  Downloading https://github.com/andreasvc/readability/tarball/master
     - 34 kB 4.1 MB/s
Building wheels for collected packages: readability
```

```
  Building wheel for readability (setup.py) … done
  Created wheel for readability: filename=readability-0.3.1-py3-none-any.whl
size=35795
sha256=47a9f760a564d8b397bd67f513798b2337c549bc37ce7f706a7acae41e012405
  Stored in directory: /tmp/pip-ephem-wheel-cache-
nmemz40s/wheels/58/12/f0/bea5aea4b2145544d5b9ef8276243974ffd337999948172fb9
Successfully built readability
Installing collected packages: readability
Successfully installed readability-0.3.1
```

[34]:
```
import readability
```

[35]:
```
all_grades = (filter_data["content"]).apply(readability.getmeasures)
T_grades = (filter_data[filter_data["label"] == "T"]["content"]).
 ↪apply(readability.getmeasures)
F_grades = (filter_data[filter_data["label"] == "F"]["content"]).
 ↪apply(readability.getmeasures)
all_grades.items
```

[35]:
```
<bound method Series.items of 1       {'readability grades': {'Kincaid':
0.890000000…
2       {'readability grades': {'Kincaid': 9.449230769…
3       {'readability grades': {'Kincaid': 5.852222222…
4       {'readability grades': {'Kincaid': 9.011666666…
5       {'readability grades': {'Kincaid': 11.64, 'ARI…
                              …
1771    {'readability grades': {'Kincaid': 14.59666666…
1772    {'readability grades': {'Kincaid': 20.666, 'AR…
1773    {'readability grades': {'Kincaid': 20.66490566…
1774    {'readability grades': {'Kincaid': 5.206666666…
1775    {'readability grades': {'Kincaid': 13.43666666…
Name: content, Length: 1775, dtype: object>
```

[36]:
```
#To change the read_ratings, ['DaleChallIndex'] with a value in here https://
 ↪pypi.org/project/readability/
read_ratings = [i['readability grades']['DaleChallIndex'] for i in all_grades]
T_read_ratings = [i['readability grades']['DaleChallIndex'] for i in T_grades]
F_read_ratings = [i['readability grades']['DaleChallIndex'] for i in F_grades]
read_ratings[0:5]
```

[36]:
```
[12.1267,
 16.42745384615385,
 13.301522222222223,
 13.379816666666667,
 12.131825000000001]
```

[37]:
```
[np.mean(T_read_ratings),np.std(T_read_ratings)], [np.mean(F_read_ratings),np.
↪std(F_read_ratings)]
```

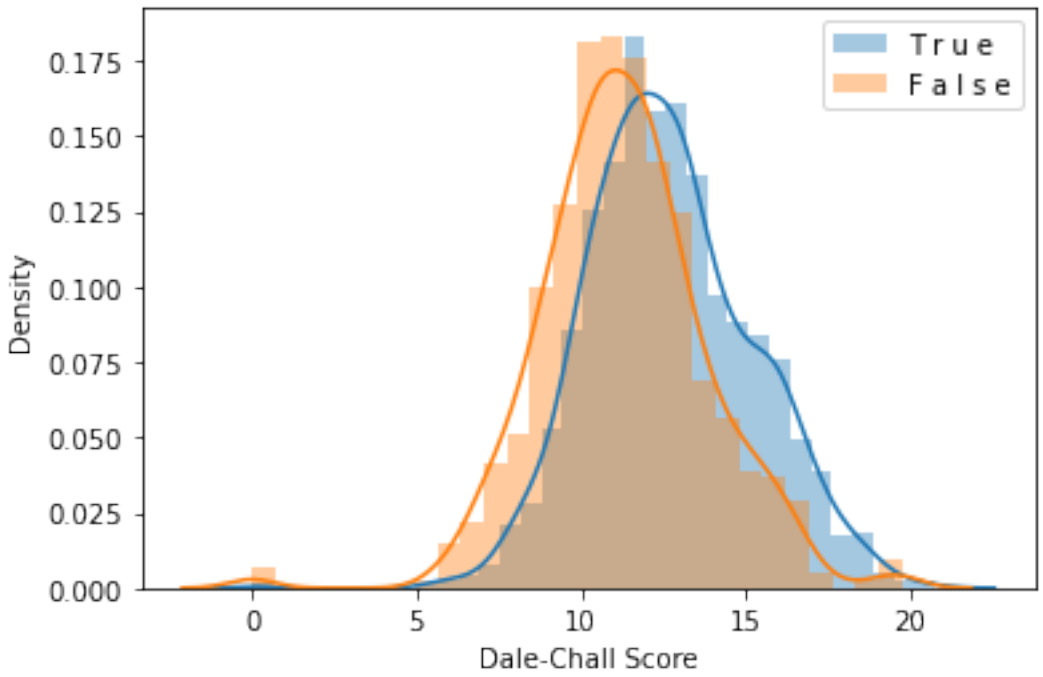
[37]:
```
([12.69533025390202, 2.491044245435879],
 [11.295110779763483, 2.5151115532842754])
```

[38]:
```
import seaborn as sns

x = sns.distplot(T_read_ratings, label='T r u e')
sns.distplot(F_read_ratings, label='F a l s e')
plt.legend()
x.set_xlabel("Dale-Chall Score", fontsize = 10)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[38]:
```
Text(0.5, 0, 'Dale-Chall Score')
```



Both True and False tweets have a similar score. Both scores, according to the SMOG index conversion table represent that the average covid-related tweet, regardless of True or False label, needs only a 6th grade reading level needed to fully understand.

https://readabilityformulas.com/smog-readability-formula.php

[39]:
```
combined_data["Readability"] = read_ratings
combined_data
```

[39]:
```
      accord  affect  …  new_people_vaccinated_smoothed_per_hundred
Readability
0          0       0  …                                         NaN
12.126700
1          0       0  …                                         NaN
16.427454
2          0       0  …                                         NaN
13.301522
3          0       0  …                                         NaN
13.379817
4          0       0  …                                         NaN
12.131825
…      …      …  …                                          …
…
1770       0       0  …                                         NaN
15.124833
1771       0       0  …                                         NaN
13.379900
1772       0       0  …                                         NaN
10.138319
1773       0       0  …                                         NaN
9.643833
1774       0       0  …                                         NaN
10.748150

[1775 rows x 234 columns]
```

Imputed Missing Data with Feature Means; Convert T/F labels to 1/0 respectively

[40]:
```
from sklearn.impute import SimpleImputer
#IMPUTE IN MEAN VALUE FOR MISSING VAL
imputer = SimpleImputer()
imputed_data = imputer.fit_transform(combined_data.drop(['label'], axis=1))
#imputed_data = imputer.fit_transform(combined_data_smooth.drop(['label'],␣
 ↪axis=1))
labels = combined_data['label'].str.replace('T','1').str.replace('F','0').
 ↪astype(int)
```

```
#labels = combined_data_smooth['label'].str.replace('T','1').str.
 ↪replace('F','0').astype(int)
imputed_data
```

[40]:
```
array([[ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 12.1267    ],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 16.42745385],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 13.30152222],
       …,
       [ 0.        ,  0.        ,  0.        , …,  0.06      ,
        16.7       , 10.13831887],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497,  9.64383333],
       [ 0.        ,  0.        ,  0.        , …,  0.12676809,
        10.75788497, 10.74815   ]])
```

##Preliminary modeling

[41]:
```
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.metrics import confusion_matrix, make_scorer, roc_auc_score
```

[42]:
```
def calculate_metrics(y_true, y_pred):
  cmtx = confusion_matrix(y_true, y_pred)
  tn, fn, tp, fp = cmtx[0][0],cmtx[1][0],cmtx[1][1],cmtx[0][1]
  metrics = {}
  metrics['tpr'] = tp / (tp + fn)
  metrics['fpr'] = fp / (fp + tn)
  metrics['accuracy'] = np.mean(y_true==y_pred)
  return metrics

#Code from lab 4 (adapted)
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
def plot_roc(y_test, predict_probas, model_names):
  plt.figure(figsize=(12, 8))
  plt.title('ROC Curve', fontsize=18)
  plt.xlabel('FPR', fontsize=16)
  plt.ylabel('TPR', fontsize=16)
  plt.xlim([-0.01, 1.00])
  plt.ylim([-0.01, 1.01])
  for i,y_proba in enumerate(predict_probas):
```

```
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=3, label=model_names[i]+' (area = {:0.2f})'.
↪format(roc_auc))
  plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--', label='Naive␣
↪Baseline (area = 0.50)')
  plt.legend(loc='lower right', fontsize=14)
  #plt.show()

#Adapted from Lab 8b
def bootstrap_validation(test_data, test_label, model, metrics_list,␣
↪sample=5000, random_state=66):
    n_sample = sample
    n_metrics = len(metrics_list)
    output_array=np.zeros([n_sample, n_metrics])
    output_array[:]=np.nan
    print(output_array.shape)
    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index),␣
↪replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_predicted = model.predict(bs_data)
        bs_proba = model.predict_proba(bs_data)[:,1]
        for metrics_iter in range(n_metrics):
            metrics = metrics_list[metrics_iter]
            output_array[bs_iter,␣
↪metrics_iter]=metrics(bs_label,bs_predicted,bs_proba)
    output_df = pd.DataFrame(output_array)
    return output_df

#custom written metric functions
def get_tpr(y_true, y_pred,y_proba):
  cmtx = confusion_matrix(y_true, y_pred)
  tn, fn, tp, fp = cmtx[0][0],cmtx[1][0],cmtx[1][1],cmtx[0][1]
  return tp / (tp + fn)
def get_fpr(y_true, y_pred,y_proba):
  cmtx = confusion_matrix(y_true, y_pred)
  tn, fn, tp, fp = cmtx[0][0],cmtx[1][0],cmtx[1][1],cmtx[0][1]
  return fp / (fp + tn)
def get_accuracy(y_true, y_pred,y_proba):
  return np.mean(y_true==y_pred)
def get_AUC(y_true, y_pred,y_proba):
  return roc_auc_score(y_true, y_proba)

  #FOR CROSS VAL
def cv_fpr(y_true, y_pred):
```

```
  cmtx = confusion_matrix(y_true, y_pred)
  tn, fn, tp, fp = cmtx[0][0],cmtx[1][0],cmtx[1][1],cmtx[0][1]
  return fp / (fp + tn)
```

[43]:
```
#Construct 80/20 train-test split
X_train, X_test, y_train, y_test = train_test_split(imputed_data, labels,␣
↪test_size=0.2, random_state=142)
```

###RandomForestClassifier

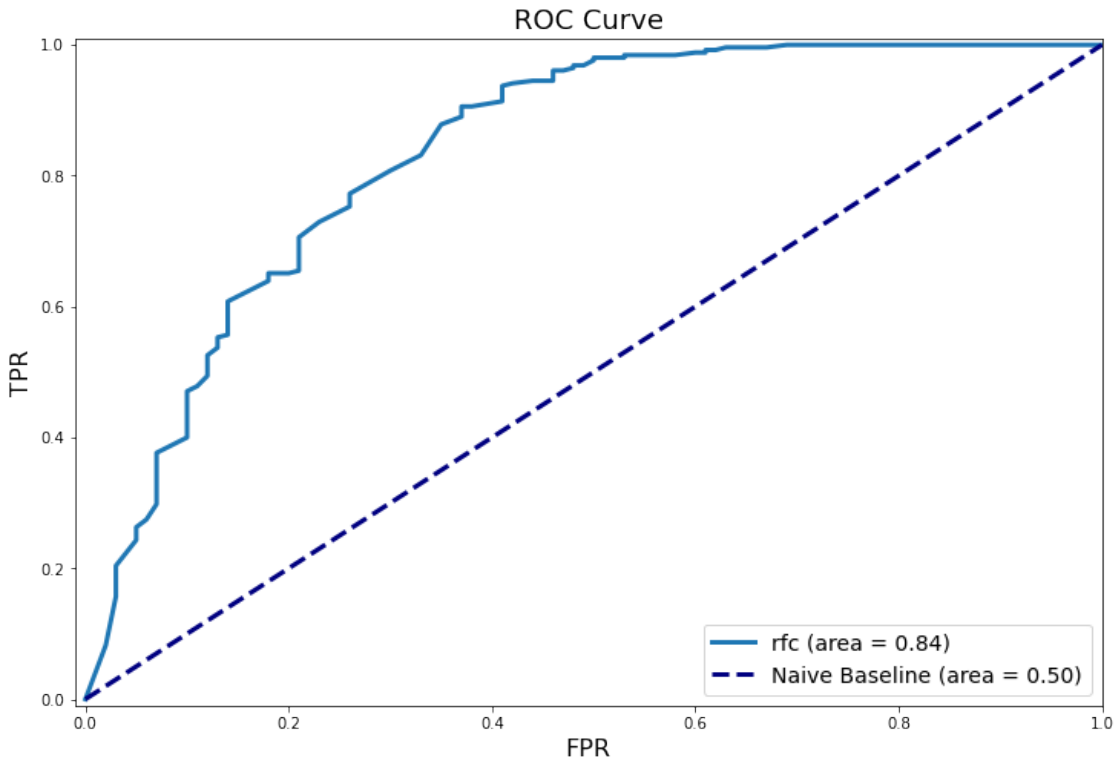
[44]:
```
rfc = RandomForestClassifier(random_state=142)
parameters = {'max_features': np.arange(20,200,10)} #cv with ccp_alpha?
cv = KFold(n_splits=5, random_state=142, shuffle=True)
rfc = GridSearchCV(rfc, param_grid=parameters, scoring=make_scorer(cv_fpr,␣
↪greater_is_better = False), cv=cv, verbose=0)
rfc.fit(X_train, y_train)
```

[44]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=142, shuffle=True),
             estimator=RandomForestClassifier(random_state=142),
             param_grid={'max_features': array([ 20,  30,  40,  50,  60,  70,
80,  90, 100, 110, 120, 130, 140,
       150, 160, 170, 180, 190])},
             scoring=make_scorer(cv_fpr, greater_is_better=False))
```

[45]:
```
rfc_predict_proba = rfc.predict_proba(X_test)[:,1]
rfc_predictions = rfc.predict(X_test)
rfc.best_params_
```

[45]:
```
{'max_features': 40}
```

[46]:
```
#Code from lab 4
plot_roc(y_test, [rfc_predict_proba], ['rfc'])
```



[47]:
```
rfc_performance = calculate_metrics(y_test, rfc_predictions)
print({f"{x}: {str(rfc_performance[x])}" for x in rfc_performance})
```

```
{'tpr: 0.9372549019607843', 'accuracy: 0.8394366197183099', 'fpr: 0.41'}
```

###Logistic Regression (regular, LASSO, Ridge, Elastic Net)

[48]:
```
logistic_regression = LogisticRegression(random_state = 142, penalty = 'none',␣
↪max_iter = 1000)
logistic_regression.fit(X_train, y_train)

lasso = LogisticRegression(random_state = 142, penalty = 'l1', solver =␣
↪'liblinear', max_iter = 1000)
lasso.fit(X_train, y_train)

ridge = LogisticRegression(random_state = 142, penalty = 'l2', solver =␣
↪'liblinear', max_iter = 1000)
ridge.fit(X_train, y_train)

#Elastic net: 50/50 mix of Lasso and Ridge regularization
elastic_net = LogisticRegression(random_state = 142, solver = 'saga', penalty =␣
↪'elasticnet', l1_ratio = 0.5, max_iter = 1000)
elastic_net.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:354:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  ConvergenceWarning,
```

[48]:
```
LogisticRegression(l1_ratio=0.5, max_iter=1000, penalty='elasticnet',
                   random_state=142, solver='saga')
```

[49]:
```
logreg_predict_proba = logistic_regression.predict_proba(X_test)[:,1]
logreg_predictions = logistic_regression.predict(X_test)
elastic_net_predict_proba = elastic_net.predict_proba(X_test)[:,1]
elastic_net_predictions = elastic_net.predict(X_test)
lasso_predict_proba = lasso.predict_proba(X_test)[:,1]
lasso_predictions = lasso.predict(X_test)
ridge_predict_proba = ridge.predict_proba(X_test)[:,1]
ridge_predictions = ridge.predict(X_test)
```

[50]:
```
logreg_performance = calculate_metrics(y_test, logreg_predictions)
print({f"{x}: {str(logreg_performance[x])}" for x in logreg_performance})
```

```
{'tpr: 0.9450980392156862', 'fpr: 0.6', 'accuracy: 0.7915492957746478'}
```

[51]:
```
elastic_net_performance = calculate_metrics(y_test, elastic_net_predictions)
print({f"{x}: {str(elastic_net_performance[x])}" for x in␣
 ↪elastic_net_performance})
```

```
{'accuracy: 0.7323943661971831', 'fpr: 0.51', 'tpr: 0.8274509803921568'}
```

[52]:
```
lasso_performance = calculate_metrics(y_test, lasso_predictions)
print({f"{x}: {str(lasso_performance[x])}" for x in lasso_performance})
```

```
{'tpr: 0.9098039215686274', 'fpr: 0.43', 'accuracy: 0.8140845070422535'}
```

[53]:
```
ridge_performance = calculate_metrics(y_test, ridge_predictions)
print({f"{x}: {str(ridge_performance[x])}" for x in ridge_performance})
```

```
{'accuracy: 0.7887323943661971', 'fpr: 0.57', 'tpr: 0.9294117647058824'}
```

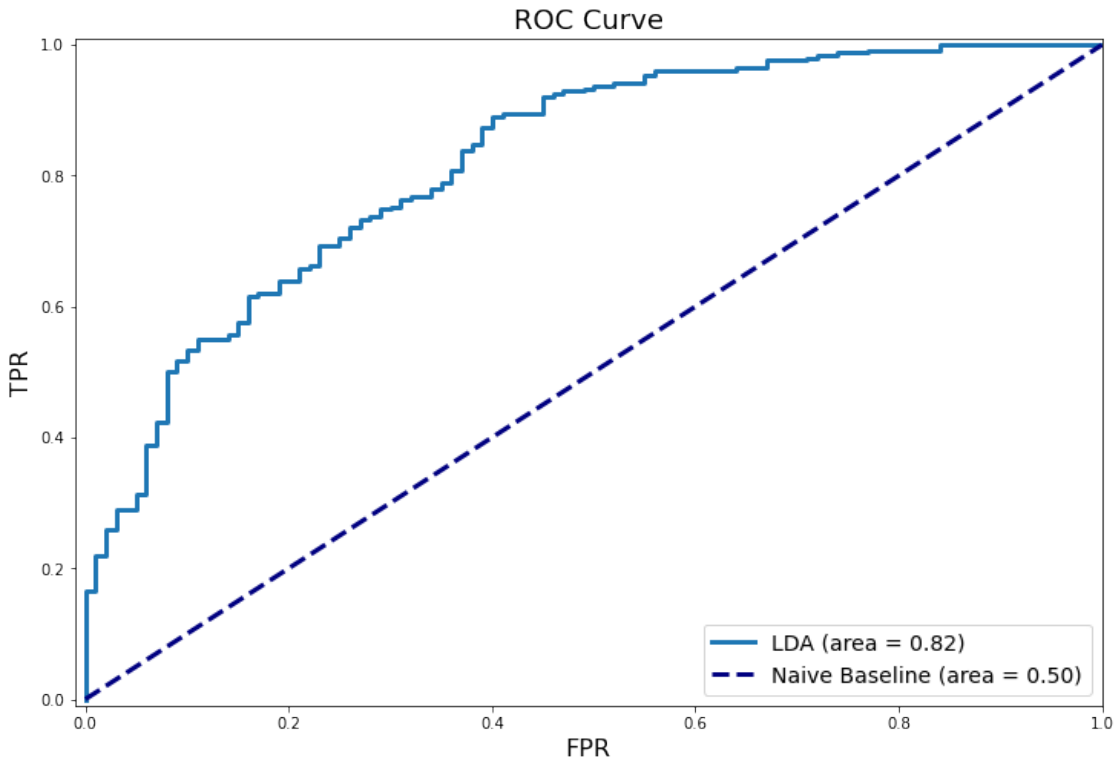
###LDA

[54]:
```
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train, y_train)
```

[54]:
```
LinearDiscriminantAnalysis()
```

[55]:
```
LDA_predict_proba = LDA.predict_proba(X_test)[:,1]
LDA_predictions = LDA.predict(X_test)
```

[56]:
```
plot_roc(y_test, [LDA_predict_proba], ['LDA'])
```



[57]:
```
lda_performance = calculate_metrics(y_test, LDA_predictions)
print({f"{x}: {str(lda_performance[x])}" for x in lda_performance})
```

```
{'tpr: 0.8941176470588236', 'accuracy: 0.8084507042253521', 'fpr: 0.41'}
```

###GradientBoostingClassifier

[58]:
```
gbc = GradientBoostingClassifier(random_state=142)
parameters = {'n_estimators': np.arange(100,1000,180), 'learning_rate':np.
 ↪arange(0.1, 0.5, 0.1)}
cv = KFold(n_splits=5, random_state=142, shuffle=True)
gbc = GridSearchCV(gbc, param_grid=parameters, scoring=make_scorer(cv_fpr,␣
 ↪greater_is_better = False), cv=cv, verbose=0)
gbc.fit(X_train, y_train)
```

[58]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=142, shuffle=True),
             estimator=GradientBoostingClassifier(random_state=142),
             param_grid={'learning_rate': array([0.1, 0.2, 0.3, 0.4]),
                         'n_estimators': array([100, 280, 460, 640, 820])},
             scoring=make_scorer(cv_fpr, greater_is_better=False))
```

[59]:
```
gbc_predict_proba = gbc.predict_proba(X_test)[:,1]
gbc_predictions = gbc.predict(X_test)
```

```
gbc.best_params_
```

[59]:
```
{'learning_rate': 0.4, 'n_estimators': 460}
```

##Model Testing

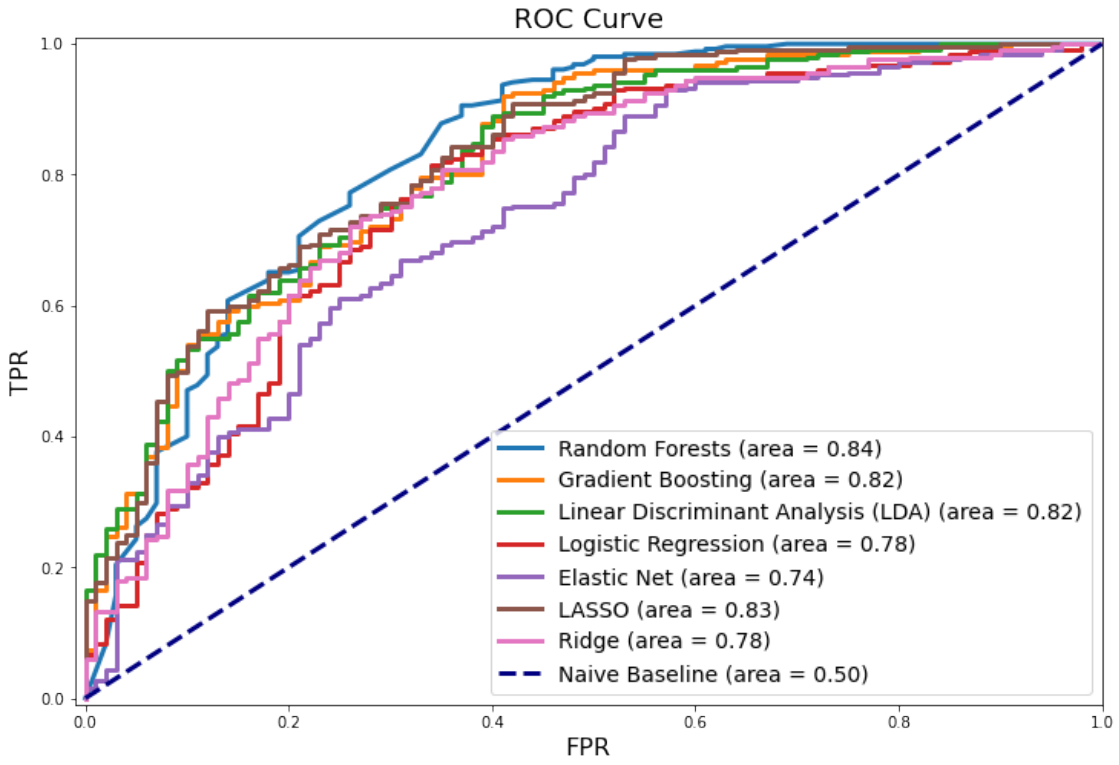
list of relevant variables:

model variable names: rfc, gbc, logistic_regression, lasso, ridge, elastic_net, LDA,

predict_proba list: rfc_predict_proba, gbc_predict_proba, LDA_predict_proba, logreg_predict_proba, elastic_net_predict_proba, lasso_predict_proba, ridge_predict_proba

predictions list: gbc_predictions, LDA_predictions, logreg_predictions, elastic_net_predictions,lasso_predictions, ridge_predictions

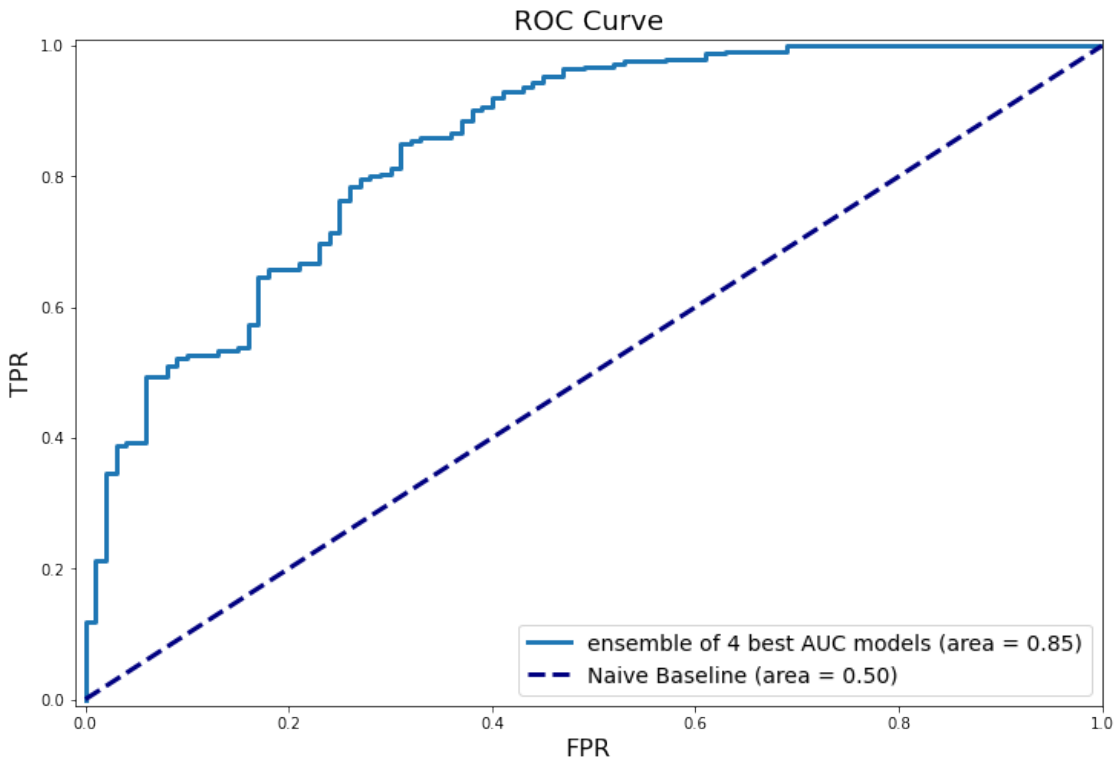
[74]:
```
plot_roc(y_test, [rfc_predict_proba, gbc_predict_proba, LDA_predict_proba,␣
↪logreg_predict_proba, elastic_net_predict_proba, lasso_predict_proba,␣
↪ridge_predict_proba],
         ['Random Forests', 'Gradient Boosting', 'Linear Discriminant Analysis␣
↪(LDA)', 'Logistic Regression', 'Elastic Net', 'LASSO', 'Ridge'])
```



[61]:
```
#Custom class for a voting classifier on our already trained models
#sklearn classifier requires unfit models, but we want to use already fit␣
↪models, so a custom function was needed
```

```
class ensembler():
  def __init__(self, models):
    self.models = models
  def predict(self, X_test):
    predictions = pd.DataFrame()
    for i, model in enumerate(self.models):
      predictions[i] = model.predict(X_test)
    return predictions.max(axis=1)
  def predict_proba(self, X_test):
    predictions = pd.DataFrame()
    for i, model in enumerate(self.models):
      predictions[i] = model.predict_proba(X_test)[:,1]
    return predictions.mean(axis=1)
```

[63]:
```
ensemble1 = ensembler([rfc, gbc, lasso, LDA])
ensemble1_proba = ensemble1.predict_proba(X_test)
ensemble1_predictions = ensemble1.predict(X_test)
plot_roc(y_test, [ensemble1_proba], ['ensemble of 4 best AUC models'])
```



[75]:
```
validation = pd.concat([pd.DataFrame(X_test), pd.Series(y_test.
↪reset_index()['label'])], axis = 1)
valx = validation.drop('label',axis=1)
```

```
valy = validation['label']
rfc_performance = bootstrap_validation(valx, valy, rfc, [get_fpr, get_tpr,␣
 ↪get_accuracy, get_AUC], sample=5000, random_state=66)
lasso_performance = bootstrap_validation(valx, valy, lasso, [get_fpr, get_tpr,␣
 ↪get_accuracy, get_AUC], sample=5000, random_state=66)
gbc_performance = bootstrap_validation(valx, valy, gbc, [get_fpr, get_tpr,␣
 ↪get_accuracy, get_AUC], sample=5000, random_state=66)
LDA_performance = bootstrap_validation(valx, valy, LDA, [get_fpr, get_tpr,␣
 ↪get_accuracy, get_AUC], sample=5000, random_state=66)
elastic_performance = bootstrap_validation(valx, valy, elastic_net, [get_fpr,␣
 ↪get_tpr, get_accuracy, get_AUC], sample=5000, random_state=66)
logreg_performance = bootstrap_validation(valx, valy, logistic_regression,␣
 ↪[get_fpr, get_tpr, get_accuracy, get_AUC], sample=5000, random_state=66)
ridge_performance = bootstrap_validation(valx, valy, ridge, [get_fpr, get_tpr,␣
 ↪get_accuracy, get_AUC], sample=5000, random_state=66)
```

```
(5000, 4)
(5000, 4)
(5000, 4)
(5000, 4)
(5000, 4)
(5000, 4)
(5000, 4)
```

[70]:
```
print(f'random forest {calculate_metrics(y_test, rfc_predictions)}')
print(f'lasso {calculate_metrics(y_test, lasso_predictions)}')
print(f'gradient boosting {calculate_metrics(y_test, gbc_predictions)}')
print(f'LDA {calculate_metrics(y_test, LDA_predictions)}')
print(f'elastic net {calculate_metrics(y_test, elastic_net_predictions)}')
print(f'logistic regression {calculate_metrics(y_test, logreg_predictions)}')
print(f'ridge regression {calculate_metrics(y_test, ridge_predictions)}')
```

```
random forest {'tpr': 0.9372549019607843, 'fpr': 0.41, 'accuracy':
0.8394366197183099}
lasso {'tpr': 0.9098039215686274, 'fpr': 0.43, 'accuracy': 0.8140845070422535}
gradient boosting {'tpr': 0.8862745098039215, 'fpr': 0.41, 'accuracy':
0.8028169014084507}
LDA {'tpr': 0.8941176470588236, 'fpr': 0.41, 'accuracy': 0.8084507042253521}
elastic net {'tpr': 0.8274509803921568, 'fpr': 0.51, 'accuracy':
0.7323943661971831}
logistic regression {'tpr': 0.9450980392156862, 'fpr': 0.6, 'accuracy':
0.7915492957746478}
ridge regression {'tpr': 0.9294117647058824, 'fpr': 0.57, 'accuracy':
0.7887323943661971}
```

[ ]: