A Voting Classifier Approach For Disasters Tweets Classification

Wong Qi Yuan, Jeffrey
Capstone Project (Project A)

Practical Natural Language Processing (MS9007)
Specialist Diploma in Vision & Language Analytics

What is the significance of the disaster tweet classification project?

Uses a voting classifier for disaster tweet classification that improves the disaster management by accurately identifying relevant tweets from the social media, enabling efficient emergency response and resource mobilisation.

What is the objective of the disaster tweet classification project?

To develop an accurate and less computational resources voting classifier to automatically classify disaster tweets, leveraging machine learning to save time and resources compared to manual classification.

Dataset Preparation

1.Read CSV file

```
# create the file path
fpath = "/content/drive/My Drive/SDVLA/nlp/capstone_project/disasters_social_media.csv"
# read the csv file
disasters_df = pd.read_csv(fpath)
# display the first five rows of the data
disasters_df.head()
```

	Unnamed: 0	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	choose_one	choose_one:confidence	choose_one_gold	keyword	location	text	tweetid	userid
0	0	778243823	True	golden	156	NaN	Relevant	1.0000	Relevant	NaN	NaN	Just happened a terrible car crash	1.0	NaN
1	1	778243824	True	golden	152	NaN	Relevant	1.0000	Relevant	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	13.0	NaN
2	2	778243825	True	golden	137	NaN	Relevant	1.0000	Relevant	NaN	NaN	Heard about #earthquake is different cities, s...	14.0	NaN
3	3	778243826	True	golden	136	NaN	Relevant	0.9603	Relevant	NaN	NaN	there is a forest fire at spot pond, geese are...	15.0	NaN
4	4	778243827	True	golden	138	NaN	Relevant	1.0000	Relevant	NaN	NaN	Forest fire near La Ronge Sask. Canada	16.0	NaN

- The dataset used for this project is called "**disasters_social_media.csv**", which contains tweets with respective keywords.
- Use **read_csv()** function from the Pandas library to read the csv file.

Datasets Preparation

2.Display Data Information

```
# display an information of the disaster dataset
disasters_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10876 entries, 0 to 10875
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0              10876 non-null  int64
 1   _unit_id                10876 non-null  int64
 2   _golden                 10876 non-null  bool
 3   _unit_state             10876 non-null  object
 4   _trusted_judgments      10876 non-null  int64
 5   _last_judgment_at       10792 non-null  object
 6   choose_one              10876 non-null  object
 7   choose_one:confidence   10876 non-null  float64
 8   choose_one_gold         87 non-null     object
 9   keyword                 10789 non-null  object
 10  location                7238 non-null   object
 11  text                    10876 non-null  object
 12  tweetid                 10876 non-null  float64
 13  userid                  10789 non-null  float64
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 1.1+ MB
```

Datasets Preparation

3.Extract Relevant Columns

```
# select the "keyword" and "text" columns and
# store it into a new variable
disasters_df2 = disasters_df[['keyword', 'text']]
# display 5 random rows in the dataset
disasters_df2.sample(5)
```

	keyword	text
8424	sandstorm	Watch This Airport Get Swallowed Up By A Sands...
5849	hailstorm	Severe weather and a rare hailstorm pummel Bos...
4398	electrocute	@Adanne___ kindly follow back
1294	bloody	@TradCatKnight (1) Russia may have played into...
10107	typhoon	Typhoon Soudelor taking dead aim at Taiwan htt...

- The "**keyword**" column was chosen to classify the disasters classification (i.e. natural disaster or non-natural disaster).
- The **"text"** column was chosen for model training and evaluation purpose.

Data Cleansing

1.Define a function to perform a specific task.

```python
# define a function to remove empty rows in the dataframe
def remove_empty_rows(dataframe):
  return dataframe.dropna(axis = 0, inplace = False)

# define a function to convert input text into lowercase
def lowercase_normalization(text):
  return text.lower()

# define a function to remove the twitter handle from the input text
def remove_twitter_handle(text):
  return re.sub(r'@\w+', '', text)

# define a function to remove url/ hyperlinks from the input text
def remove_url_hyperlinks(text):
  return re.sub(r"http\S+|www\S+|https\S+", '', text)

# define a function to remove the month abbreviations from the input text
def remove_months_abbre(text):
  return re.sub(r"jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec", '', text)

# define a function to remove newline character from the input text
def remove_newline_char(text):
  return re.sub(r'\n', '', text)
```

2.Call a function to perform a specific task on a specific column in every row.

```python
# call the function to remove empty rows in the dataframe
disasters_df2_copy = remove_empty_rows(disasters_df2_copy)

# call the function to convert the input text and keyword into lowercase
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: lowercase_normalization(x))
disasters_df2_copy['keyword'] = disasters_df2_copy['keyword'].apply(lambda x:
lowercase_normalization(x))

# call the function to remove twitter handle in the text column values of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_twitter_handle(x))

# call the function to remove URLs starting with 'http' or 'https' in the text and keyword columns of
the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_url_hyperlinks(x))
disasters_df2_copy['keyword'] = disasters_df2_copy['keyword'].apply(lambda x: remove_url_hyperlinks(x))

# call the function to remove month abbreviations such as jan, feb, etc, in the text column of the
dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_months_abbre(x))

# call the function to remove newline character (i.e. \n) in the text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_newline_char(x))
```

Specific Tasks:
remove empty rows, lowercase normalization, remove mentions, remove url hyperlinks, remove months abbreviations, and remove newline characters

Data Cleansing

1.Define a function to perform a specific task.

```
# define a function to remove non-ascII characters from the input text
def remove_non_ascii_char(text):
  return re.sub(r'[^\x00-\x7F]+', '', text)

# define a function to expand the contractions
def contractions_expansion(text):
  return contractions.fix(text)

# define a function to remove punctuations from input text
def remove_punctuations(text):
  text_no_punctuations = ''.join([' ' if char in string.punctuation else char for char in text]) #
replace punctuations with space
  return text_no_punctuations

# define a function to remove digits from the input text
def remove_digits(text):
  return re.sub(r"\d+", '', text)
```

2.Call a function to perform a specific task on a specific column in every row.

```
# call the function to remove non-ascII characters in the text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_non_ascii_char(x))

# call the function to expand the contractions in the text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: contractions_expansion(x))

# call the function to remove the punctuations in the text column and the keyword column of the
dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_punctuations(x))
disasters_df2_copy['keyword'] = disasters_df2_copy['keyword'].apply(lambda x: remove_punctuations(x))

# call the function to remove digits in the text column and keyword column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x: remove_digits(x))
disasters_df2_copy['keyword'] = disasters_df2_copy['keyword'].apply(lambda x: remove_digits(x))
```

Specific Tasks:
Remove non-ascii characters, contractions expansion, remove punctuations, an remove digits

Data Cleansing

```python
import emoji

# define a function to convert emoji and emoticon to words
def convert_emoji_emoticon_to_words(text):

  emoticon_dict = {
    ":)": "happy",
    ":D": "laughing",
    ":(": "sad",
    ":|": "neural",
    ":P": "tongue out",
    ";)": "winking",
    "<3": "love",
    ":O": "surprised",
    ":/": "confused",
    ":*": "kissing",
    ":')": "tears of joy",
    ":@": "angry",
    ":S": "confused",
    ">:(": "angry",
    ":|": "indifferent",
    ":$": "embarrased",
    ":^)": "raised eyebrow",
    "B-)": "cool",
    "O:)": "angel",
    ":3": "cute",
    "-_-": "bored"}
  # convert emojis to text representation
  converted_text = emoji.demojize(text)
  # replace emoticons with their corresponding words
  for emoticon, text_rep in emoticon_dict.items():
    converted_text = converted_text.replace(emoticon, text_rep)
  # return the converted text
  return converted_text
```

Screensho

1.Define a function to perform a specific task.

2.Call a function to perform a specific task on a specific column in every row.

```python
# call the function to apply the emojis and emoticons conversion in text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x:
convert_emoji_emoticon_to_words(x))
```

Specific Tasks:
Convert emoji and emoticon to words

Data Cleansing

```
def updated_stopwords():
  # load the existing set of stopwords
  stopwordsList = list(stopwords.words("english"))
  # additional stopwords to append
  additional_stopwords = ['via','like','build','get','would','one','two','feel',
              'fuck','take','way','may','first','latest','want',
              'make','back','see','know','let','look','come','got',
              'still','say','think','great','please','amp', 'new', 'lie', 'pm', 'am', 'please',
'beings',
              'california', 'video']
  # append the additional stopwords to the existing stopwords
  stopwordsList.extend(additional_stopwords)
  return stopwordsList

# define a function to remove stopwords from the input text
def remove_stopwords(sentence):
  list_of_tokens = sentence.split(" ")
  sentence_no_stop_words = " ".join([i for i in list_of_tokens if i not in updated_stopwords()])
  return sentence_no_stop_words

# load the english model for spacy
nlp = spacy.load("en_core_web_sm")
# define a function
def lemmatization(text):
  # apply lemmatization to the input text
  lemmatize_text = " ".join([w.lemma_.lower() for w in nlp(text)])
  return lemmatize_text

# define a function to filter the words with a length of less than 3 characters
def filter_words_less_than_3_characters(text):
  # use list comprehension to filter words based on the lengths
  # join the filtered words back into a string uses spaces as characters
  filteredText = " ".join([word for word in text.split(" ") if len(word) >= 3])
  return filteredText # returned the filtered text
```

Screenshot

1.Define a function to perform a specific task.

2.Call a function to perform a specific task on a specific column in every row.

```
# call the function to remove stopwords in text column of the dataframe
disasters_df2_copy['text']= disasters_df2_copy['text'].apply(lambda x:remove_stopwords(x))

# call the function to lemmatize the text in the text column of the dataframe
disasters_df2_copy['text']= disasters_df2_copy["text"].apply(lambda x: lemmatization(x))

# call the function to filter words less than 3 characters to the text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x :
filter_words_less_than_3_characters(x))
```

Specific Tasks:
Remove stopwords, lemmatization, and filter words with less than 3 characters

Data Cleansing

1.Define a function to perform a specific task.

```
# define a function to remove consecutive duplicate words from the input text
def remove_consecutive_duplicate_words(text):
  return re.sub(r'\b(\w+)(\s+\1)+\b', '', text)

# define a function to remove duplicate rows
def remove_duplicate_rows(text):
  return text.drop_duplicates(subset = "text")
```

Specific Tasks:
Remove consecutive duplicate words and remove duplicate rows

2.Call a function to perform a specific task on a specific column in every row.

```
# call the function to remove the duplicated consecutive words in the text column of the dataframe
disasters_df2_copy['text'] = disasters_df2_copy['text'].apply(lambda x:
remove_consecutive_duplicate_words(x))

# call the function to remove duplicate rows in the text column of the dataframe
disasters_df2_copy = remove_duplicate_rows(disasters_df2_copy)
```

Classification of Disaster Types

```python
# define a function to classify the disasters (i.e. natural or non-natural disaster)
def classify_disaster(disaster):
  natural_disasters = ['aftershock',
                       'avalanche',
                       'blizzard',
                       'bush fires',
                       'cyclone',
                       'drought',
                       'earthquake',
                       'hailstorm',
                       'hurricane',
                       'landslide',
                       'lava',
                       'mudslide',
                       'natural disaster',
                       'nuclear disaster',
                       'oil spill',
                       'rainstorm',
                       'sandstorm',
                       'snowstorm',
                       'thunderstorm',
                       'tornado',
                       'tsunami',
                       'typhoon',
                       'violent storm',
                       'volcano',
                       'wild fires',
                       'wildfire',
                       'windstorm']
  if disaster in natural_disasters:
    class_label = "natural disaster"
  else:
    class_label = "non-natural disaster"
  return class_label
```

Screenshot

- Define and call the **classify_disaster** function to perform classification on the different types of disasters as either "natural" or "non-natural" based on whether they appear in a predefined list of natural disaster keywords. The classification results are stored in a new column for each row called 'target' in the DataFrame.

```python
# apply the classification function based on the keyword column and store the classification into a new column called the target
disasters_df2_copy['target'] = disasters_df2_copy['keyword'].apply(lambda x: classify_disaster(x))
```

	keyword	text	target
7198	natural disaster	snea america spoil natural disaster humble	natural disaster
5486	flames	devastation shoce firefighter continue battle ...	non-natural disaster
9265	sunk	wow alright sansa shoo head bline rapidly info...	non-natural disaster
825	battle	lie nucle punch death battle	non-natural disaster
5291	fear	love fear	non-natural disaster

Data Partitioning

```
from sklearn.model_selection import train_test_split
# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(disasters_df3_copy['text'],
                                                    disasters_df3_copy['target'],
                                                    test_size = 0.20,
                                                    random_state = 42,
                                                    stratify = disasters_df3_copy['target'])
```

- Uses **train_test_split()** function from scikit-learn library to split the data into **80% training set** and **20% testing set**.
- The training set is used to train the model, while the test set is used to assess its performance.
- The random state parameter has been set to 42 to ensure its reproducibility.
- The 'stratify' parameter uses the class label to ensure that **both sets represent the same proportion of different classes on imbalanced datasets.**

TF-IDF Text Representation

```
from sklearn.feature_extraction.text import TfidfVectorizer
# create an instance of TfidfVectorizer with specified parameters
# - stop_words: Remove common English words
# - ngram_range: Generate n-grams of size 1 to 3
tfidf_vectorizer = TfidfVectorizer(stop_words = 'english', ngram_range = (1, 3))
# fit the TfidfVectorizer on the training data and transform it
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
# transform the test data using the fitted TfidfVectorizer
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

- Uses **TfidfVectorizer()** function from scikit-learn library to convert a series of text in the training data into a numerical vector representation, where each element of the vector corresponds to the TF-IDF weight of a particular term.

Why uses TFIDF instead of other embeddings such as Word2Vec, GloVe, and Sentence Transformer?

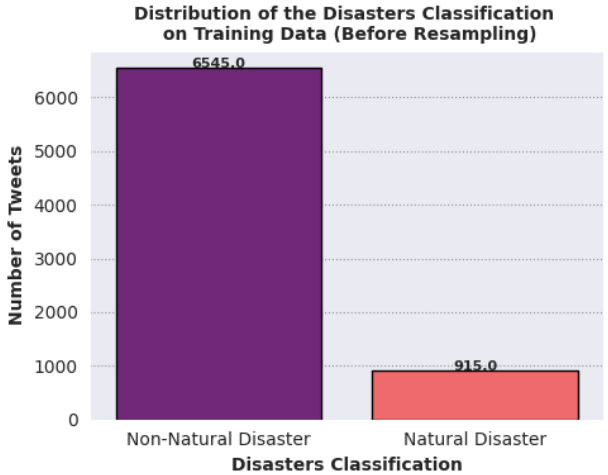
- During the experiments, it was found that TF-IDF worked better for the natural disaster category compared to Word2Vec, GloVe, and Sentence Transformers.
- TF-IDF made use of all the available words in the training set.
- As such, TF-IDF gained more information from the training set as compared to the embedding methods.

SMOTE-Tomek Resampling

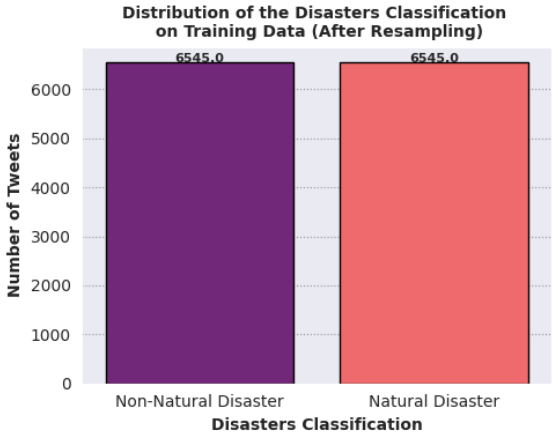
```
from imblearn.combine import SMOTETomek
# create an instance of SMOTETomek with a random state for reproducibility
smt = SMOTETomek(random_state = 42)
# apply SMOTETomek to perform oversampling and undersampling
X_train_res, y_train_res = smt.fit_resample(X_train_tfidf, y_train)
```

- Uses **SMOTETomek()** function from imblearn library to generate the synthetic samples for the minority class **(SMOTE)** and remove instances that may cause misclassification **(Tomek)** by identifying the pairs of examples from different classes that are close to each other in the feature space.

SMOTE-Tomek Resampling

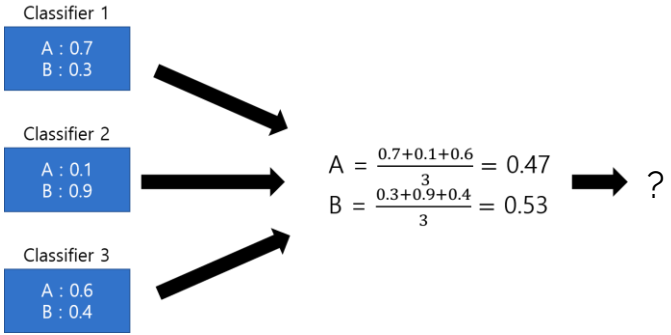


Apply SMOTE-Tomek



- After equalizing the number of instances in each class through resampling, the resulting dataset provides a balanced representation of both classes, allowing for more accurate and unbiased model training.

Voting Classifier



Model Training

```python
from sklearn.ensemble import VotingClassifier
# define the individual classifiers
clf1 = LogisticRegression(C = 1, solver = 'sag', random_state = 42)
clf2 = RandomForestClassifier(n_estimators = 10, random_state = 42)
clf3 = ExtraTreesClassifier(n_estimators = 10, random_state = 42)

# train and fit the voting classifier
vc = VotingClassifier(estimators = [('lr', clf1),
                                    ('rfc', clf2),
                                    ('svc', clf3)], voting = 'soft')
vc.fit(X_train_res, y_train_res)
```

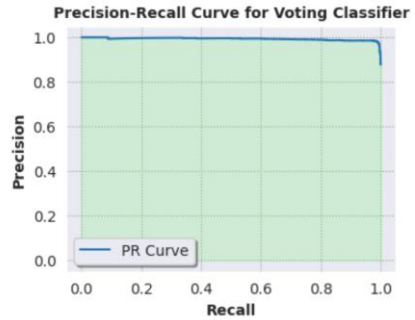
- Each individual classifier is trained independently on the training dataset.
- Using the **soft-voting** approach, each classifier predicts the **probability** of the input belonging to each class.
- Because it uses the **uniform weighing**, the final prediction is made by **averaging the predicted probabilities across all classifiers** and selecting the class with the highest average probability.

Voting Classifier

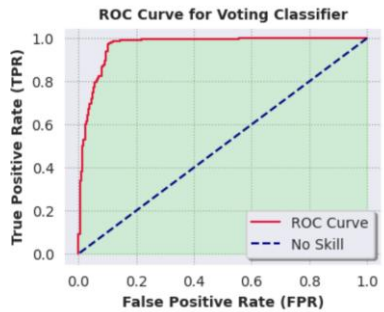
Model Evaluation on Test Data

Classification Report for Voting Classifier

	precision	recall	f1-score	support
Natural Disaster	0.92	0.86	0.89	229
Non-Natural Disaster	0.98	0.99	0.99	1636
accuracy			0.97	1865
macro avg	0.95	0.93	0.94	1865
weighted avg	0.97	0.97	0.97	1865



The PR AUC for Voting Classifier is 0.9933



The ROC AUC for Voting Classifier is 0.9659

Interpretation:

- The model achieved accuracy of 97%, meaning it correctly classified 97% of all instances in the test set.
- A high PR-AUC means that the model is capable of achieving high precision while maintaining a high level of recall.
- A high ROC-AUC means that the model has achieved a good performance in terms of distinguishing between the natural and non-natural disasters.

What does the results tells us?

- By combining the predictions of multiple classifiers, it can capture different aspects of the data and reduce the impact of individual classifier biases or errors.
- It leads to better generalisation and improved performance compared to using a single classifier.

Bayes Search CV

- In general, Bayes Search CV uses probabilistics models to approximate the objective function (typically the model performance metrics) and determine the next set of hyperparameters to evaluate. It models the hyperparameter space and uses the information from past evaluations to guide the search towards more promising regions.
- More efficient and requires fewer evaluations compared to Grid Search, making it suitable for complex and high dimensional hyperparameter spaces.
- Grid Search CV is computational expensive as it searches through a predefined set of hyperparameters, covering all possible combinations.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import f1_score, make_scorer
from sklearn.skopt import BayesSearchCV

# define the individual classifiers
clf1 = LogisticRegression(solver = 'sag', random_state = 42)
clf2 = RandomForestClassifier(random_state = 42)
clf3 = ExtraTreesClassifier(random_state = 42)

# define the param space
param_space = {'lr__C': [1, 2, 3],
               'rfc__n_estimators': [5, 10, 15],
               'etc__n_estimators': [5, 10, 15]}

# define scoring function using the f1-score
f1_scorer = make_scorer(f1_score, average = 'macro')

# define the voting classifier
voting_classifier = VotingClassifier(estimators=[('lr', clf1),
                                                 ('rfc', clf2),
                                                 ('etc', clf3)], voting='soft')

# perform bayes search optimization
optimizer = BayesSearchCV(voting_classifier,
                          param_space,
                          cv=3,
                          scoring = f1_scorer,
                          verbose = 3,
                          n_jobs = 1,
                          return_train_score = True,
                          random_state = 42)
optimizer.fit(X_train_res, y_train_res)
```

Screenshot

Bayes Search CV

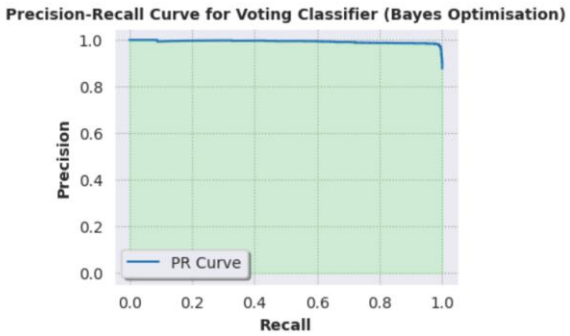
```
print("Best Score: ", optimizer.best_score_)
print("Best Parameters: ", optimizer.best_params_)
print("Best Estimators: ", optimizer.best_estimator_)
```

```
Best Score:  0.9938122078077075
Best Parameters:  OrderedDict([('etc__n_estimators', 5), ('lr__C', 3), ('rfc__n_estimators', 15)])
Best Estimators:  VotingClassifier(estimators=[('lr',
                                LogisticRegression(C=3, random_state=42,
                                                   solver='sag')),
                               ('rfc',
                                RandomForestClassifier(n_estimators=15,
                                                       random_state=42)),
                               ('etc',
                                ExtraTreesClassifier(n_estimators=5,
                                                     random_state=42))],
                   voting='soft')
```

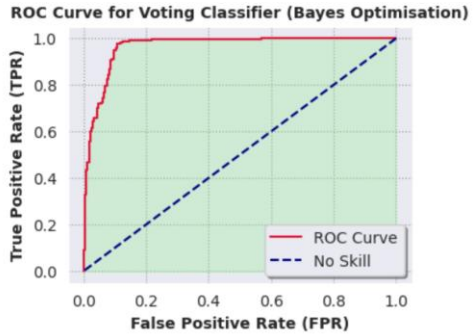
Bayes Search CV

Classification Report for Voting Classifier (Bayes Optimisation)

	precision	recall	f1-score	support
natural disaster	0.93	0.85	0.89	229
non-natural disaster	0.98	0.99	0.99	1636
accuracy			0.97	1865
macro avg	0.96	0.92	0.94	1865
weighted avg	0.97	0.97	0.97	1865



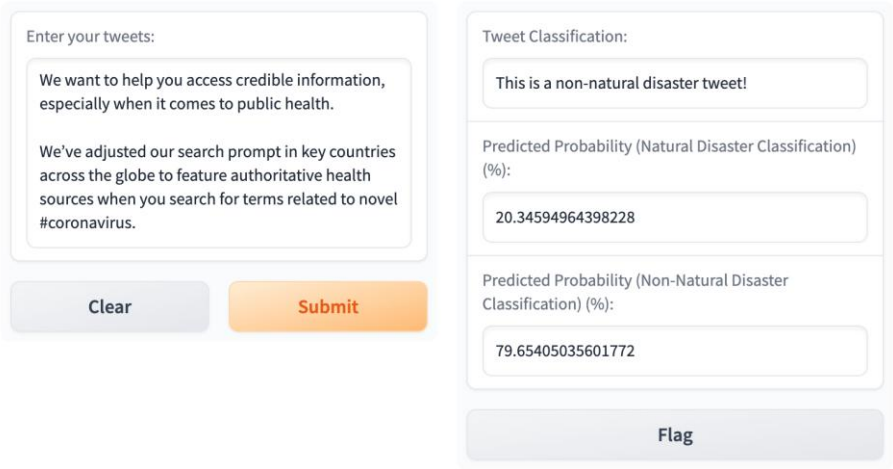
The PR AUC for Voting Classifier (Bayes Optimisation) is 0.9931



The ROC AUC for Voting Classifier (Bayes Optimisation) is 0.9646

Deploy a Voting Classifier Model on Gradio

An Example of Non-Natural Disaster Tweet



An Example of Natural Disaster Tweet