In [1]:

```python
import os
import string

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import RFE, RFECV
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression, Ridge, RidgeCV
from sklearn.metrics import make_scorer
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    ShuffleSplit,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import (
    OneHotEncoder,
    OrdinalEncoder,
    PolynomialFeatures,
    StandardScaler,
)
from sklearn.svm import SVC, SVR

%matplotlib inline
```

Feature engineering

One of the most important aspects which influences performance of machine learning models is the features used to represent the problem. If your underlying representation is bad whatever fancy model you use is not going to help. With a better feature representation, a simple and a more interpretable model is likely to perform reasonably well.

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models.

The code below reads the data CSV.

In [2]:

```python
df = pd.read_csv("tweets.csv", usecols=["keyword", "text", "target", "location"])
train_df, test_df = train_test_split(df, test_size=0.2, random_state=2)
train_df.head()
```

Out[2]:

	keyword	location	text	target
3289	debris	NaN	Unfortunately, both plans fail as the 3 are im...	0
2672	crash	SLC	I hope this causes Bernie to crash and bern. S...	0
2436	collide	NaN	—pushes himself up from the chair beneath to r...	0

	keyword	location	text	target
9622	suicide%20bomb	NaN	Widow of CIA agent killed in 2009 Afghanistan ...	1
8999	screaming	Azania	As soon as God say yes they'll be screaming we...	0

In [3]:

```
X_train, y_train = train_df.drop(columns=["target"]), train_df["target"]
X_test, y_test = test_df.drop(columns=["target"]), test_df["target"]
```

The challenge here is correctly classifying the real disaster tweets. The data set includes tweets about disasters as well as keyword that relate the disaster, such as crash, suicide bomb, and so on and also the location. The prediction problem we're attempting to tackle is determining whether a tweet is connected to a real disaster or is merely a joke/movie review in a disaster-related environment.

In [4]:

```
train_df["target"].value_counts()
```

Out[4]:

```
0    7395
1    1701
Name: target, dtype: int64
```

Yes, there is a class imbalance in the given data set. As can be seen from the target value counts above, there are only 1701 tweets of genuine disaster, accounting for less than 20% of the whole data set. To cope with this, we'll need to employ a different scoring measure than accuracy, one that focuses on judging the model's performance based on actual disaster tweets.

In [5]:

```
scoring = ["precision" , "f1", "recall", "roc_auc" , "average_precision"]
```

As seen above there is a class imbalance in the given data set.The scoring metric `accuracy` cannot be used to judge the model performance. For the given use case i.e. classifying the actual disaster tweets correctly, we need to be confident that the tweet is actually a disaster, a suitable metric here would be `precision` but we also need to avoid the false negatives which means the `recall` is equally important. So in order to account for the trade off `f1` score sounds a better scoring metric for the given use case. To examine how successfully the model discriminate between the two classes, we utilize the `auc roc` score.

The location feature

The location feature seems quite messy.

In [6]:

```
train_df["location"].unique()
```

Out[6]:

```
array([nan, 'SLC', 'Azania', ..., 'Santiago de Chile', 'she/her 🌈',
       'Greater Manchester'], dtype=object)
```

In [7]:

```
len(train_df["location"].unique())
```

Out[7]:

```
3747
```

In [8]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9096 entries, 3289 to 7336
```

```
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   keyword   9096 non-null   object
 1   location  6370 non-null   object
 2   text      9096 non-null   object
 3   target    9096 non-null   int64 
dtypes: int64(1), object(3)
memory usage: 355.3+ KB
```

In [9]:

```
X_train["location"].isnull().value_counts()
```

Out[9]:

```
False    6370
True     2726
Name: location, dtype: int64
```

In [10]:

```
X_train["location"].value_counts()
```

Out[10]:

```
United States                     80
Australia                         68
London, England                   66
UK                                62
India                             60
                                  ..
Arizona City, AZ                   1
Yorkshire & Scotland               1
th: hakuna matata                  1
Tacloban City, Eastern Visayas     1
Greater Manchester                 1
Name: location, Length: 3746, dtype: int64
```

1) There are many null values in the feature Location i.e. 2726 which might be a challenge to impute. 2) Throughout the location feature, there are countries, cities, regions, and other meaningless terms jumbled in. 3) There are a lot of instances in the feature location with special characters that aren't in an acceptable format. 4) Furthermore, the information is not standardized. 5) Because there are 3747 distinct values, applying one hot encoding is difficult.

We may divide the location feature into four categories: country, city, region, and other, and then run OHE on each of them. The category "OTHER" can be allocated to null and non-meaningful words. But in given case we choose to drop the feature

Identifying feature types

In [11]:

```
drop_features = ["location"]
text_feature = "text"
key_word = "keyword"

preprocessor = make_column_transformer(
    (CountVectorizer(stop_words="english"), text_feature),
    (CountVectorizer(stop_words="english"), key_word)
)


data=preprocessor.fit_transform(X_train,y_train)
data.shape
```

Out[11]:

```
(9096, 23627)
```

Due to null values and values that will not contribute significance to the model, we are removing the feature `location`.

We use the `Count Vectorizer` feature to turn the `text` feature into vectorized values for each meaningful word that the model can interpret.

We use a separate `Count Vectorizer` on the `keyword` feature so that the model may take into account the most common disaster-related keywords when making predictions.

DummyClassifier

In [12]:

```
results = {}
```

In [13]:

```
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [14]:

```
dummy = make_pipeline(preprocessor, DummyClassifier())
```

In [15]:

```
results['Dummy Classifier']= mean_std_cross_val_scores(dummy, X_train, y_train, scoring =s
pd.DataFrame(results)
```

```
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[15]:

	Dummy Classifier
fit_time	0.122 (+/- 0.004)
score_time	0.053 (+/- 0.002)
test_precision	0.000 (+/- 0.000)
test_f1	0.000 (+/- 0.000)
test_recall	0.000 (+/- 0.000)
test_roc_auc	0.500 (+/- 0.000)
test_average_precision	0.187 (+/- 0.000)

Logistic regression

In [16]:

```
LR = make_pipeline(preprocessor, LogisticRegression(max_iter =2000))
LR
```

Out[16]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('countvectorizer-1',
                                                  CountVectorizer(stop_words='english'),
                                                  'text'),
                                                 ('countvectorizer-2',
                                                  CountVectorizer(stop_words='english'),
                                                  'keyword')])),
                ('logisticregression', LogisticRegression(max_iter=2000))])
```

In [17]:

```
results['Logistic Regression']= mean_std_cross_val_scores(LR, X_train, y_train, scoring =s
pd.DataFrame(results)
```

Out[17]:

	Dummy Classifier	Logistic Regression
fit_time	0.122 (+/- 0.004)	0.301 (+/- 0.016)
score_time	0.053 (+/- 0.002)	0.058 (+/- 0.002)
test_precision	0.000 (+/- 0.000)	0.811 (+/- 0.012)
test_f1	0.000 (+/- 0.000)	0.628 (+/- 0.026)
test_recall	0.000 (+/- 0.000)	0.513 (+/- 0.036)
test_roc_auc	0.500 (+/- 0.000)	0.898 (+/- 0.011)
test_average_precision	0.187 (+/- 0.000)	0.747 (+/- 0.018)

Hyperparameter optimization

In [18]:

```
param_grid_gamma_random = {"columntransformer__countvectorizer-1__max_features": [1000,50(
                           "columntransformer__countvectorizer-2__max_features": [50,100,2
                           "logisticregression__C": 10.0 ** np.arange(-3, 4),
                           "logisticregression__class_weight": [None, "balanced"]}

random_search = RandomizedSearchCV(
    LR, param_distributions=param_grid_gamma_random, n_jobs=-1,scoring ="f1",random_state
)
random_search.fit(X_train, y_train)

print("Best Hyper-Parameters are:", random_search.best_params_)
```

```
Best Hyper-Parameters are: {'logisticregression__class_weight': 'balanced', 'logisticregre
ssion__C': 1.0, 'columntransformer__countvectorizer-2__max_features': 50, 'columntransform
er__countvectorizer-1__max_features': 15000}
```

In [19]:

```
print("Best Score is:", random_search.best_score_)
```

```
Best Score is: 0.6717594271881064
```

The best hyper parameters are: {'logistic regression - class_weight': 'balanced', 'logistic regression- C': 1.0, 'Count Vectorizer-text-max_features': 50, 'Count Vectorizer-keyword-max_features': 15000}

The best cross validation f1 score is 0.6717594271881064

Feature engineering

In [20]:

```
import nltk
from nltk.corpus import stopwords

nltk.download("vader_lexicon")
nltk.download("punkt")
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/varadrajrameshpoojary/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/varadrajrameshpoojary/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [21]:

```
def get_relative_length(text, TWITTER_ALLOWED_CHARS=280.0):
    """
    Returns the relative length of text.

    Parameters:
    ------
    text: (str)
    the input text

    Keyword arguments:
    ------
    TWITTER_ALLOWED_CHARS: (float)
    the denominator for finding relative length
```

```python
    Returns:
    -------
    relative length of text: (float)

    """
    return len(text) / TWITTER_ALLOWED_CHARS


def get_length_in_words(text):
    """
    Returns the length of the text in words.

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    -------
    length of tokenized text: (int)

    """
    return len(nltk.word_tokenize(text))


def get_sentiment(text):
    """
    Returns the compound score representing the sentiment of the given text: -1 (most extr
    The compound score is a normalized score calculated by summing the valence scores of e

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    -------
    sentiment of the text: (str)
    """
    scores = sid.polarity_scores(text)
    return scores["compound"]
```

In [22]:

```python
train_df = train_df.assign(n_words=train_df["text"].apply(get_length_in_words))
train_df = train_df.assign(vader_sentiment=train_df["text"].apply(get_sentiment))
train_df = train_df.assign(rel_char_len=train_df["text"].apply(get_relative_length))

test_df = test_df.assign(n_words=test_df["text"].apply(get_length_in_words))
test_df = test_df.assign(vader_sentiment=test_df["text"].apply(get_sentiment))
test_df = test_df.assign(rel_char_len=test_df["text"].apply(get_relative_length))
```

In [23]:

```python
#mention_count
train_df['mention_count'] = train_df['text'].apply(lambda x: len([c for c in str(x) if c =
test_df['mention_count'] = test_df['text'].apply(lambda x: len([c for c in str(x) if c ==

# punctuation_count
train_df['punctuation_count'] = train_df['text'].apply(lambda x: len([c for c in str(x) if
test_df['punctuation_count'] = test_df['text'].apply(lambda x: len([c for c in str(x) if c

# unique_word_count
train_df['unique_word_count'] = train_df['text'].apply(lambda x: len(set(str(x).split())))
test_df['unique_word_count'] = test_df['text'].apply(lambda x: len(set(str(x).split())))

# stop_word_count
```

```
train_df['stop_word_count'] = train_df['text'].apply(lambda x: len([w for w in str(x).lowe
test_df['stop_word_count'] = test_df['text'].apply(lambda x: len([w for w in str(x).lower

# url_count
train_df['url_count'] = train_df['text'].apply(lambda x: len([w for w in str(x).lower().sp
test_df['url_count'] = test_df['text'].apply(lambda x: len([w for w in str(x).lower().spli

# mean_word_length
train_df['mean_word_length'] = train_df['text'].apply(lambda x: np.mean([len(w) for w in s
test_df['mean_word_length'] = test_df['text'].apply(lambda x: np.mean([len(w) for w in str

# char_count
train_df['char_count'] = train_df['text'].apply(lambda x: len(str(x)))
test_df['char_count'] = test_df['text'].apply(lambda x: len(str(x)))

# hashtag_count
train_df['hashtag_count'] = train_df['text'].apply(lambda x: len([c for c in str(x) if c =
test_df['hashtag_count'] = test_df['text'].apply(lambda x: len([c for c in str(x) if c ==
```

Mention count reasoning:

In the event of a crisis, people frequently mention their loved ones and other authorities in attempt to disseminate the message.

Punctuation count reasoning:

When there is an emergency or a calamity, people prefer to use exclamation marks and other punctuation.

In [24]:

```
import nltk
from spacymoji import Emoji
import en_core_web_md  # pre-trained model
import spacy

nlp = en_core_web_md.load()
```

In [25]:

```
nlp.add_pipe("emoji", first=True);

def get_emoji_count(text):
    """
    Returns the count of emojis in specified text

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    ------
    count of emojis in specified text : int
    """
    doc = nlp(text)
    return len(doc._.emoji)

train_df['emoji_count']=train_df["text"].apply(get_emoji_count)
test_df['emoji_count']=test_df["text"].apply(get_emoji_count)
```

Emoji count reasoning:

When there is an emergency or a calamity, people prefer to use less emoji's while tweeting as compared to writing a joke or a movie review.

In [26]:
```
train_df.head()
```

Out[26]:

	keyword	location	text	target	n_words	vader_sentiment	rel_char_len	mention_cou
3289	debris	NaN	Unfortunately, both plans fail as the 3 are im...	0	22	-0.7650	0.425000	
2672	crash	SLC	I hope this causes Bernie to crash and bern. S...	0	18	-0.5697	0.267857	
2436	collide	NaN	—pushes himself up from the chair beneath to r...	0	21	0.0000	0.439286	
9622	suicide%20bomb	NaN	Widow of CIA agent killed in 2009 Afghanistan ...	1	20	-0.9460	0.428571	
8999	screaming	Azania	As soon as God say yes they'll be screaming we...	0	14	0.2960	0.203571	

Pipeline with all features

In [27]:
```
drop_features = ['location']
text_feature = "text"
key_word= "keyword"
target = "target"
numeric_features = list(
    set(train_df.columns)
    - set(drop_features)
    - set([text_feature])
    - set([key_word])
    -set([target])
)

preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (CountVectorizer(stop_words="english", max_features= 15000), text_feature),
    (CountVectorizer(stop_words="english", max_features = 50), key_word)

)
```

In [28]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9096 entries, 3289 to 7336
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   keyword            9096 non-null   object
 1   location           6370 non-null   object
 2   text               9096 non-null   object
 3   target             9096 non-null   int64
 4   n_words            9096 non-null   int64
```

```
 5   vader_sentiment     9096 non-null   float64
 6   rel_char_len        9096 non-null   float64
 7   mention_count       9096 non-null   int64
 8   punctuation_count   9096 non-null   int64
 9   unique_word_count   9096 non-null   int64
 10  stop_word_count     9096 non-null   int64
 11  url_count           9096 non-null   int64
 12  mean_word_length    9096 non-null   float64
 13  char_count          9096 non-null   int64
 14  hashtag_count       9096 non-null   int64
 15  emoji_count         9096 non-null   int64
dtypes: float64(3), int64(10), object(3)
memory usage: 1.2+ MB
```

In [29]:

```python
X_train, y_train = train_df.drop(columns=["target"]), train_df["target"]
X_test, y_test = test_df.drop(columns=["target"]), test_df["target"]
```

In [30]:

```python
data= preprocessor.fit_transform(X_train, y_train)
```

In [31]:

```python
data.shape
```

Out[31]:

```
(9096, 15062)
```

In [32]:

```python
pipe_lr = make_pipeline(preprocessor, LogisticRegression(class_weight= 'balanced', C = 1.(
pipe_lr
```

Out[32]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('standardscaler',
                                                  StandardScaler(),
                                                  ['mention_count', 'n_words',
                                                   'char_count',
                                                   'stop_word_count',
                                                   'mean_word_length',
                                                   'unique_word_count',
                                                   'rel_char_len', 'url_count',
                                                   'vader_sentiment',
                                                   'hashtag_count',
                                                   'emoji_count',
                                                   'punctuation_count']),
                                                 ('countvectorizer-1',
                                                  CountVectorizer(max_features=15000,
                                                                  stop_words='english'),
                                                  'text'),
                                                 ('countvectorizer-2',
                                                  CountVectorizer(max_features=50,
                                                                  stop_words='english'),
                                                  'keyword')])),
                ('logisticregression',
                 LogisticRegression(class_weight='balanced', max_iter=2000))])
```

In [33]:

```python
results['LR_feature-engineered']= mean_std_cross_val_scores(pipe_lr, X_train, y_train, sco
pd.DataFrame(results)
```

Out[33]:

	Dummy Classifier	Logistic Regression	LR_feature-engineered
fit_time	0.122 (+/- 0.004)	0.301 (+/- 0.016)	0.552 (+/- 0.079)
score_time	0.053 (+/- 0.002)	0.058 (+/- 0.002)	0.063 (+/- 0.003)
test_precision	0.000 (+/- 0.000)	0.811 (+/- 0.012)	0.665 (+/- 0.018)

	Dummy Classifier	Logistic Regression	LR_feature-engineered
test_f1	0.000 (+/- 0.000)	0.628 (+/- 0.026)	0.672 (+/- 0.022)
test_recall	0.000 (+/- 0.000)	0.513 (+/- 0.036)	0.678 (+/- 0.032)
test_roc_auc	0.500 (+/- 0.000)	0.898 (+/- 0.011)	0.893 (+/- 0.010)
test_average_precision	0.187 (+/- 0.000)	0.747 (+/- 0.018)	0.737 (+/- 0.018)

Interpretation

Yes, as observed above, there is an improvement following feature engineering, i.e. the f1 score has grown dramatically after adding the additional features. In addition, while recall has grown significantly, precision has dropped. However, the model has delivered a superior f1 score, which compensates for the precision-recall tradeoff.

In [34]:

```
col_names = numeric_features + pipe_lr.named_steps["columntransformer"].named_transformers

pipe_lr.fit(X_train, y_train)

lr_coeffs = pipe_lr.named_steps["logisticregression"].coef_

cefficients= pd.DataFrame(
    data=lr_coeffs.T,index= col_names, columns=["Coefficients"]
).sort_values(by="Coefficients", ascending=False)

cefficients.head(10)
```

Out[34]:

	Coefficients
windstorm	2.593822
rescued	2.228706
thunderstorm	2.155966
whirlwind	1.952249
influenza	1.936107
died	1.927072
survived	1.919522
carried	1.881456
ukrainian	1.860744
sinkhole	1.781671

Yes, the coefficients match my intuitions; as can be seen above, the attributes windstorm, rescued, thunderstorm, died, and so on have the greatest coefficients; these features reflect a disaster.

Tree Based Model

In [35]:

```
from catboost import CatBoostClassifier
pipe_catboost_all = make_pipeline(
    preprocessor, CatBoostClassifier(random_state=123, verbose = 0)
)
results['CatBoost_feature-engineered']= mean_std_cross_val_scores(pipe_catboost_all, X_tra
pd.DataFrame(results)
```

Out[35]:

	Dummy Classifier	Logistic Regression	LR_feature-engineered	CatBoost_feature-engineered
fit_time	0.122 (+/- 0.004)	0.301 (+/- 0.016)	0.552 (+/- 0.079)	13.816 (+/- 0.061)
score_time	0.053 (+/- 0.002)	0.058 (+/- 0.002)	0.063 (+/- 0.003)	0.081 (+/- 0.001)
test_precision	0.000 (+/- 0.000)	0.811 (+/- 0.012)	0.665 (+/- 0.018)	0.839 (+/- 0.025)
test_f1	0.000 (+/- 0.000)	0.628 (+/- 0.026)	0.672 (+/- 0.022)	0.485 (+/- 0.030)
test_recall	0.000 (+/- 0.000)	0.513 (+/- 0.036)	0.678 (+/- 0.032)	0.342 (+/- 0.026)
test_roc_auc	0.500 (+/- 0.000)	0.898 (+/- 0.011)	0.893 (+/- 0.010)	0.851 (+/- 0.004)
test_average_precision	0.187 (+/- 0.000)	0.747 (+/- 0.018)	0.737 (+/- 0.018)	0.667 (+/- 0.015)

Test results

In [36]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_a

print('F1 Score')
print(f1_score(y_test, pipe_lr.predict(X_test)))

print('Precision Score')
print(precision_score(y_test, pipe_lr.predict(X_test)))

print('Recall Score')
print(recall_score(y_test, pipe_lr.predict(X_test)))

print('ROC AUC Score')
print(roc_auc_score(y_test, pipe_lr.predict(X_test)))

print('Average Precision Score')
print(average_precision_score(y_test, pipe_lr.predict(X_test)))
```

```
F1 Score
0.7031431897555296
Precision Score
0.6771300448430493
Recall Score
0.7312348668280871
ROC AUC Score
0.8269285564661661
Average Precision Score
0.5439537630737555
```

The f1 score and the recall scores are good for the test set also the roc -auc score looks good for the test set.