In [1]:

```
# Downloading the packages
!pip install xgboost
!pip install CatBoost
!pip install scikit-plot
!pip install autocorrect
```

In [2]:

```
# Importing the necessary modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.utils import resample
from sklearn.ensemble import StackingClassifier
```

In [3]:

```
# Load the reviews dataset
reviews = pd.read_csv('Amazon reviews.csv')
reviews.iloc[50:76]
```

Out[3]:

	category	rating	label	text
50	Home_and_Kitchen_5	5	0	Perfect. They do exactly what I need them to d...
51	Home_and_Kitchen_5	5	0	Love this movie & the time it took to finish. ...
52	Home_and_Kitchen_5	1	0	Will not seal properly. Atrocious little ones,...
53	Home_and_Kitchen_5	5	0	Got these for the third time. I have a small ...
54	Home_and_Kitchen_5	5	0	Excellent product and a much better quality th...
55	Home_and_Kitchen_5	5	1	These are just perfect, exactly what I was loo...
56	Home_and_Kitchen_5	5	1	Such a great purchase can't beat it for the price
57	Home_and_Kitchen_5	5	1	What can you say--- cheap and it works as inte...
58	Home_and_Kitchen_5	5	1	These are so nice, sturdy, like the color choi...
59	Home_and_Kitchen_5	5	1	It is nice bowl and have had a fast shipping!
60	Home_and_Kitchen_5	5	1	Great cup. Will last forever. Keeps things coo...
61	Home_and_Kitchen_5	5	1	Love them, just thought they would be a bit bi...
62	Home_and_Kitchen_5	5	1	Excellent quality product. Perfect for my ccoz...

```
63   Home_and_Kitchen_5   5         1         This fan is really pretty and I actually use it.
64   Home_and_Kitchen_5   1         1         Super rough, not soft wash cloths, more like b...
65   Home_and_Kitchen_5   5         1         Like this little guy. Use it often. He is small.
66   Home_and_Kitchen_5   5         1         Perfect size .easy to unfold. Topremoves for s...
67   Home_and_Kitchen_5   1         1         Does not do a very good job; difficult to use.
68   Home_and_Kitchen_5   5         1         frame was beautifully crafted and looks great ...
69   Home_and_Kitchen_5   5         1         My hot cocoa is finally perfect! This product ...
70   Home_and_Kitchen_5   5         1         i bought this for a friend and he loves it.
71   Home_and_Kitchen_5   5         1         Very soft and great quality for such a low price!
72   Home_and_Kitchen_5   5         1         Purchased as a Christmas gift. She will love it.
73   Home_and_Kitchen_5   5         1         accurate description, works great, came on tim...
74   Home_and_Kitchen_5   5         1         Use it in my Duvet cover. Fluffy and perfect fit!
```

In [4]:

```python
# Text preprocessing
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
from autocorrect import Speller

nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')

# Select a random row index
row_index = 2762

# Access the 'text' column before processing
print("Before processing:")
print(reviews['text'][row_index])
def preprocess_text(text):
    # Remove non-alphabetic characters
    text = re.sub('[^A-Za-z]', ' ', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize the text
    tokens = text.split()
    # Remove stopwords
    stopwords_list = stopwords.words('english')
    filtered_tokens = [token for token in tokens if token not in stopwords_list]
    # Stem the tokens
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(token) for token in filtered_tokens]
    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in stemmed_tokens]
    # Join the tokens back into a string
    preprocessed_text = ' '.join(lemmatized_tokens)
    return preprocessed_text

reviews['text'] = reviews['text'].apply(preprocess_text)

# Access the 'text' column after processing
print("After processing:")
print(reviews['text'][row_index])
```

Out [4]:

```
Before processing:
Very reliable. Originally purchased years ago for my apt, and when I
moved, I bought a new one for a new home because (and this is my only
complaint), the screws attaching it to the cabinet floor had rusted
and stripped shut. The sliding mechanism, however, lasted years
without oiling, and took tonnes of abuse and has never failed. Dead
easy to install.
After processing:
reliabl origin purchas year ago apt move bought new one new home
complaint screw attach cabinet floor rust strip shut slide mechan
howev last year without oil took tonn abus never fail dead easi instal
```

In [5]:

```
# Separate the target variable and the text data
X = reviews['text']
y = reviews['label']
```

In [6]:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [7]:

```
# Resample the training data to address data imbalance
X_train_resampled = []
y_train_resampled = []
for label in np.unique(y_train):
    X_train_label = X_train[y_train == label]
    y_train_label = y_train[y_train == label]
    X_train_label_resampled, y_train_label_resampled =
resample(X_train_label, y_train_label, n_samples=1000, replace=True,
random_state=42)
    X_train_resampled.append(X_train_label_resampled)
    y_train_resampled.append(y_train_label_resampled)
X_train_resampled = pd.concat(X_train_resampled)
y_train_resampled = pd.concat(y_train_resampled)
```

In [8]:

```
# Apply Tfidf vectorization to the text data
vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train_resampled)
X_test_tfidf = vectorizer.transform(X_test)
```

In [9]:

```
# Apply standard scaling to the training and testing sets
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_tfidf.toarray())
X_test_scaled = scaler.transform(X_test_tfidf.toarray())
```

In [10]:

```
# Define the parameter grids for grid search
rf_param_grid = {'n_estimators': [100, 500, 1000], 'max_depth': [3, 5,
None]}
xgb_param_grid = {'n_estimators': [100, 500, 1000], 'max_depth': [3,
5, 7]}
lgb_param_grid = {'n_estimators': [100, 500, 1000], 'max_depth': [3,
5, 7]}
cb_param_grid = {'n_estimators': [100, 500, 1000], 'max_depth': [3, 5,
7]}
gb_param_grid = {'n_estimators': [100, 500, 1000], 'max_depth': [3, 5,
7]}
```

In [11]:

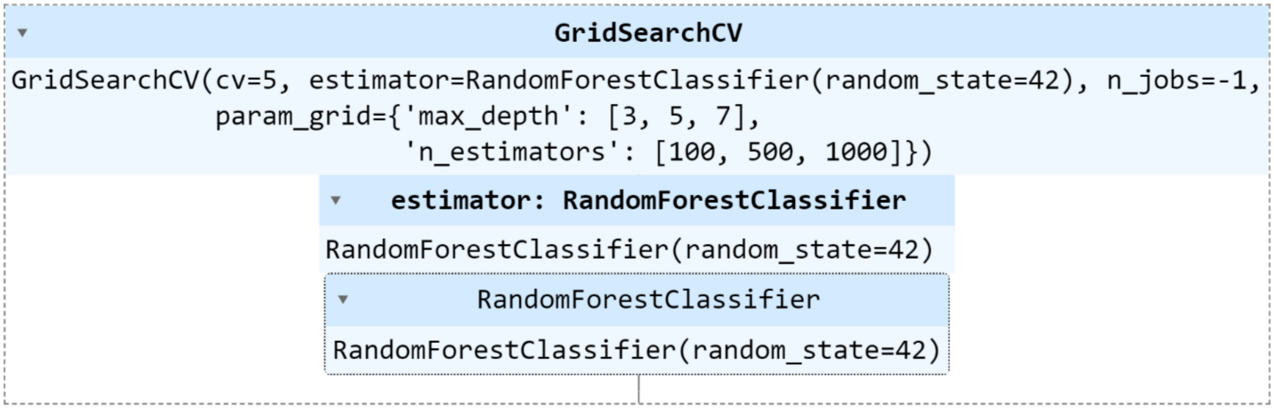
```
# Perform grid search to optimize the classifiers
rf_grid = GridSearchCV(RandomForestClassifier(random_state=42),
rf_param_grid, cv=5, n_jobs=-1)
xgb_grid = GridSearchCV(XGBClassifier(random_state=42),
xgb_param_grid, cv=5, n_jobs=-1)
```

```
lgb_grid = GridSearchCV(LGBMClassifier(random_state=42),
lgb_param_grid, cv=5, n_jobs=-1)
cb_grid = GridSearchCV(CatBoostClassifier(random_state=42),
cb_param_grid, cv=5, n_jobs=-1)
gb_grid = GridSearchCV(RandomForestClassifier(random_state=42),
gb_param_grid, cv=5, n_jobs=-1)
```

```
In [12]:
# Fit the optimized classifiers to the training data
rf_grid.fit(X_train_scaled, y_train_resampled)
xgb_grid.fit(X_train_scaled, y_train_resampled)
lgb_grid.fit(X_train_scaled, y_train_resampled)
cb_grid.fit(X_train_scaled, y_train_resampled)
gb_grid.fit(X_train_scaled, y_train_resampled)
```

```
Out [12]:
Learning rate set to 0.013851
0:    learn: 0.6909365 total: 166ms    remaining: 2m 45s
1:    learn: 0.6891085 total: 256ms    remaining: 2m 7s
2:    learn: 0.6869619 total: 373ms    remaining: 2m 3s
3:    learn: 0.6849605 total: 470ms    remaining: 1m 56s
4:    learn: 0.6828380 total: 562ms    remaining: 1m 51s
5:    learn: 0.6810862 total: 654ms    remaining: 1m 48s
6:    learn: 0.6791836 total: 742ms    remaining: 1m 45s
7:    learn: 0.6771038 total: 831ms    remaining: 1m 43s
8:    learn: 0.6748765 total: 925ms    remaining: 1m 41s
9:    learn: 0.6733539 total: 1.01s    remaining: 1m 40s
10:   learn: 0.6715230 total: 1.1s          remaining: 1m 38s
11:   learn: 0.6698818 total: 1.19s    remaining: 1m 37s
12:   learn: 0.6682977 total: 1.28s    remaining: 1m 36s
13:   learn: 0.6668198 total: 1.37s    remaining: 1m 36s
14:   learn: 0.6651077 total: 1.47s    remaining: 1m 36s
15:   learn: 0.6636890 total: 1.56s    remaining: 1m 35s
..............................
981:  learn: 0.2974422 total: 1m 37s   remaining: 1.79s
982:  learn: 0.2971875 total: 1m 37s   remaining: 1.69s
983:  learn: 0.2969678 total: 1m 37s   remaining: 1.59s
984:  learn: 0.2966740 total: 1m 38s   remaining: 1.49s
985:  learn: 0.2964466 total: 1m 38s   remaining: 1.39s
986:  learn: 0.2961845 total: 1m 38s   remaining: 1.29s
987:  learn: 0.2960369 total: 1m 38s   remaining: 1.19s
988:  learn: 0.2957993 total: 1m 38s   remaining: 1.09s
989:  learn: 0.2956468 total: 1m 38s   remaining: 996ms
990:  learn: 0.2954216 total: 1m 38s   remaining: 897ms
991:  learn: 0.2951882 total: 1m 38s   remaining: 798ms
992:  learn: 0.2949301 total: 1m 39s   remaining: 698ms
```

```
993:  learn: 0.2947020 total: 1m 39s    remaining: 599ms
994:  learn: 0.2945699 total: 1m 39s    remaining: 499ms
995:  learn: 0.2944823 total: 1m 39s    remaining: 400ms
996:  learn: 0.2943984 total: 1m 39s    remaining: 300ms
997:  learn: 0.2942040 total: 1m 39s    remaining: 200ms
998:  learn: 0.2941395 total: 1m 40s    remaining: 100ms
999:  learn: 0.2940769 total: 1m 40s    remaining: 0us
```



```
GridSearchCV
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7],
                         'n_estimators': [100, 500, 1000]})
estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [13]:

```
# Extract the best classifiers from the grid search results
rf_best = rf_grid.best_estimator_
xgb_best = xgb_grid.best_estimator_
lgb_best = lgb_grid.best_estimator_
cb_best = cb_grid.best_estimator_
gb_best = gb_grid.best_estimator_
```

In [14]:

```
# Define the ensemble classifiers using majority voting and stacking
voting_clf = VotingClassifier([('rf', rf_best), ('xgb', xgb_best),
('lgb', lgb_best), ('cb', cb_best),  ('gb', gb_best)], voting='soft')
stacking_clf = StackingClassifier([('rf',rf_best), ('xgb', xgb_best),
('lgb', lgb_best),('cb', cb_best),  ('gb', gb_best)])
```

In [15]:

```
# Fit the ensemble classifiers to the training data
voting_clf.fit(X_train_scaled, y_train_resampled)
stacking_clf.fit(X_train_scaled, y_train_resampled)
```

Out [15]:

```
Learning rate set to 0.012592
4:   learn: 0.6846217    total: 375ms    remaining: 1m 14s
5:   learn: 0.6830193    total: 463ms    remaining: 1m 16s
6:   learn: 0.6817952    total: 532ms    remaining: 1m 15s
7:   learn: 0.6799940    total: 600ms    remaining: 1m 14s
8:   learn: 0.6783643    total: 673ms    remaining: 1m 14s
```

```
9:  learn: 0.6769998    total: 749ms    remaining: 1m 14s
10: learn: 0.6756055    total: 820ms    remaining: 1m 13s
11: learn: 0.6744568    total: 894ms    remaining: 1m 13s
12: learn: 0.6730493    total: 965ms    remaining: 1m 13s
13: learn: 0.6712112    total: 1.04s    remaining: 1m 13s
14: learn: 0.6694191    total: 1.11s    remaining: 1m 13s
15: learn: 0.6681017    total: 1.19s    remaining: 1m 12s
…………………………

981:    learn: 0.3078034    total: 1m 20s   remaining: 1.47s
982:    learn: 0.3076445    total: 1m 20s   remaining: 1.39s
983:    learn: 0.3075808    total: 1m 20s   remaining: 1.3s
984:    learn: 0.3073479    total: 1m 20s   remaining: 1.22s
985:    learn: 0.3071265    total: 1m 20s   remaining: 1.14s
986:    learn: 0.3070223    total: 1m 20s   remaining: 1.06s
987:    learn: 0.3068619    total: 1m 20s   remaining: 977ms
988:    learn: 0.3066641    total: 1m 20s   remaining: 896ms
989:    learn: 0.3065057    total: 1m 20s   remaining: 814ms
990:    learn: 0.3063939    total: 1m 20s   remaining: 733ms
991:    learn: 0.3061856    total: 1m 20s   remaining: 651ms
992:    learn: 0.3060314    total: 1m 20s   remaining: 570ms
993:    learn: 0.3056331    total: 1m 20s   remaining: 488ms
994:    learn: 0.3053640    total: 1m 20s   remaining: 407ms
995:    learn: 0.3051338    total: 1m 21s   remaining: 326ms
996:    learn: 0.3050367    total: 1m 21s   remaining: 244ms
997:    learn: 0.3048031    total: 1m 21s   remaining: 163ms
998:    learn: 0.3045684    total: 1m 21s   remaining: 81.4ms
999:    learn: 0.3044225    total: 1m 21s   remaining: 0us

Learning rate set to 0.012592
0:  learn: 0.6908525    total: 88.5ms   remaining: 1m 28s
1:  learn: 0.6886115    total: 158ms    remaining: 1m 18s
2:  learn: 0.6870465    total: 233ms    remaining: 1m 17s
3:  learn: 0.6852878    total: 307ms    remaining: 1m 16s
4:  learn: 0.6840765    total: 375ms    remaining: 1m 14s
5:  learn: 0.6821594    total: 458ms    remaining: 1m 15s
6:  learn: 0.6804935    total: 534ms    remaining: 1m 15s
7:  learn: 0.6790381    total: 613ms    remaining: 1m 15s
8:  learn: 0.6773534    total: 694ms    remaining: 1m 16s
9:  learn: 0.6758022    total: 764ms    remaining: 1m 15s
10: learn: 0.6744437    total: 833ms    remaining: 1m 14s
11: learn: 0.6733758    total: 904ms    remaining: 1m 14s
12: learn: 0.6717796    total: 976ms    remaining: 1m 14s
13: learn: 0.6703937    total: 1.05s    remaining: 1m 13s
14: learn: 0.6690930    total: 1.13s    remaining: 1m 13s
15: learn: 0.6671332    total: 1.2s     remaining: 1m 13s
```

……………………………

```
981:    learn: 0.3023393    total: 1m 20s   remaining: 1.47s
982:    learn: 0.3021164    total: 1m 20s   remaining: 1.39s
983:    learn: 0.3018658    total: 1m 20s   remaining: 1.31s
984:    learn: 0.3016576    total: 1m 20s   remaining: 1.23s
985:    learn: 0.3014143    total: 1m 20s   remaining: 1.15s
986:    learn: 0.3011494    total: 1m 20s   remaining: 1.07s
987:    learn: 0.3010367    total: 1m 21s   remaining: 985ms
988:    learn: 0.3008613    total: 1m 21s   remaining: 903ms
989:    learn: 0.3007253    total: 1m 21s   remaining: 822ms
990:    learn: 0.3004653    total: 1m 21s   remaining: 740ms
991:    learn: 0.3001637    total: 1m 21s   remaining: 658ms
992:    learn: 0.2999900    total: 1m 21s   remaining: 576ms
993:    learn: 0.2999283    total: 1m 21s   remaining: 494ms
994:    learn: 0.2997671    total: 1m 21s   remaining: 412ms
995:    learn: 0.2997016    total: 1m 21s   remaining: 329ms
996:    learn: 0.2995307    total: 1m 22s   remaining: 247ms
997:    learn: 0.2993978    total: 1m 22s   remaining: 165ms
998:    learn: 0.2991948    total: 1m 22s   remaining: 82.3ms
999:    learn: 0.2989196    total: 1m 22s   remaining: 0us
```

```
Learning rate set to 0.012592
0:  learn: 0.6920444    total: 88.2ms   remaining: 1m 28s
1:  learn: 0.6897713    total: 162ms    remaining: 1m 20s
2:  learn: 0.6878114    total: 239ms    remaining: 1m 19s
3:  learn: 0.6859438    total: 311ms    remaining: 1m 17s
4:  learn: 0.6846315    total: 383ms    remaining: 1m 16s
5:  learn: 0.6829251    total: 467ms    remaining: 1m 17s
6:  learn: 0.6806837    total: 538ms    remaining: 1m 16s
7:  learn: 0.6790530    total: 611ms    remaining: 1m 15s
8:  learn: 0.6768527    total: 698ms    remaining: 1m 16s
9:  learn: 0.6751366    total: 772ms    remaining: 1m 16s
10: learn: 0.6730441    total: 851ms    remaining: 1m 16s
11: learn: 0.6711665    total: 931ms    remaining: 1m 16s
12: learn: 0.6701586    total: 1s       remaining: 1m 16s
13: learn: 0.6687129    total: 1.08s    remaining: 1m 15s
14: learn: 0.6669098    total: 1.16s    remaining: 1m 15s
15: learn: 0.6655400    total: 1.23s    remaining: 1m 15s
```

……………………………

```
981:    learn: 0.3037858    total: 1m 22s   remaining: 1.5s
982:    learn: 0.3035218    total: 1m 22s   remaining: 1.42s
983:    learn: 0.3033497    total: 1m 22s   remaining: 1.34s
984:    learn: 0.3030268    total: 1m 22s   remaining: 1.25s
```

```
985:    learn: 0.3027258    total: 1m 22s   remaining: 1.17s
986:    learn: 0.3025821    total: 1m 22s   remaining: 1.09s
987:    learn: 0.3024016    total: 1m 22s   remaining: 1s
988:    learn: 0.3020612    total: 1m 22s   remaining: 919ms
989:    learn: 0.3019968    total: 1m 22s   remaining: 836ms
990:    learn: 0.3019332    total: 1m 22s   remaining: 752ms
991:    learn: 0.3017925    total: 1m 22s   remaining: 668ms
992:    learn: 0.3014855    total: 1m 22s   remaining: 585ms
993:    learn: 0.3012717    total: 1m 23s   remaining: 501ms
994:    learn: 0.3010373    total: 1m 23s   remaining: 418ms
995:    learn: 0.3008316    total: 1m 23s   remaining: 334ms
996:    learn: 0.3005971    total: 1m 23s   remaining: 251ms
997:    learn: 0.3003900    total: 1m 23s   remaining: 167ms
998:    learn: 0.3002767    total: 1m 23s   remaining: 83.5ms
999:    learn: 0.3001432    total: 1m 23s   remaining: 0us
```

```
Learning rate set to 0.012592
0:  learn: 0.6916444    total: 87.7ms   remaining: 1m 27s
1:  learn: 0.6892218    total: 160ms    remaining: 1m 19s
2:  learn: 0.6866895    total: 239ms    remaining: 1m 19s
3:  learn: 0.6849472    total: 310ms    remaining: 1m 17s
4:  learn: 0.6832336    total: 381ms    remaining: 1m 15s
5:  learn: 0.6814099    total: 464ms    remaining: 1m 16s
6:  learn: 0.6795859    total: 536ms    remaining: 1m 16s
7:  learn: 0.6779248    total: 611ms    remaining: 1m 15s
8:  learn: 0.6763097    total: 698ms    remaining: 1m 16s
9:  learn: 0.6745213    total: 769ms    remaining: 1m 16s
10: learn: 0.6729563    total: 840ms    remaining: 1m 15s
11: learn: 0.6715945    total: 919ms    remaining: 1m 15s
12: learn: 0.6701184    total: 990ms    remaining: 1m 15s
13: learn: 0.6680466    total: 1.06s    remaining: 1m 14s
14: learn: 0.6666321    total: 1.16s    remaining: 1m 16s
15: learn: 0.6653995    total: 1.23s    remaining: 1m 15s
```

…………………………

```
981:    learn: 0.3041048    total: 1m 21s   remaining: 1.49s
982:    learn: 0.3039346    total: 1m 21s   remaining: 1.41s
983:    learn: 0.3038754    total: 1m 21s   remaining: 1.33s
984:    learn: 0.3037389    total: 1m 21s   remaining: 1.24s
985:    learn: 0.3035018    total: 1m 21s   remaining: 1.16s
986:    learn: 0.3032521    total: 1m 21s   remaining: 1.08s
987:    learn: 0.3029439    total: 1m 22s   remaining: 997ms
988:    learn: 0.3027430    total: 1m 22s   remaining: 914ms
```

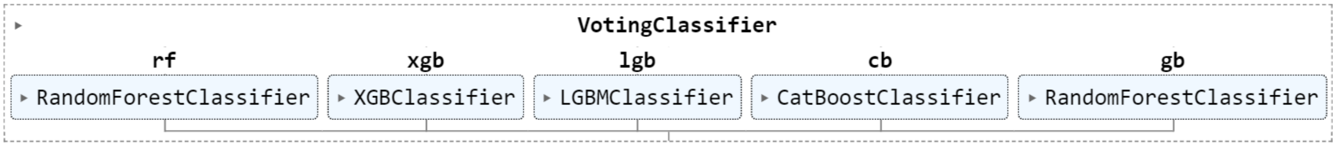
```
989:    learn: 0.3026220    total: 1m 22s   remaining: 832ms
990:    learn: 0.3024990    total: 1m 22s   remaining: 748ms
991:    learn: 0.3022544    total: 1m 22s   remaining: 665ms
992:    learn: 0.3021512    total: 1m 22s   remaining: 582ms
993:    learn: 0.3020348    total: 1m 22s   remaining: 499ms
994:    learn: 0.3018712    total: 1m 22s   remaining: 416ms
995:    learn: 0.3017743    total: 1m 22s   remaining: 332ms
996:    learn: 0.3016407    total: 1m 22s   remaining: 249ms
997:    learn: 0.3015196    total: 1m 22s   remaining: 166ms
998:    learn: 0.3014478    total: 1m 23s   remaining: 83.1ms
999:    learn: 0.3013050    total: 1m 23s   remaining: 0us
```

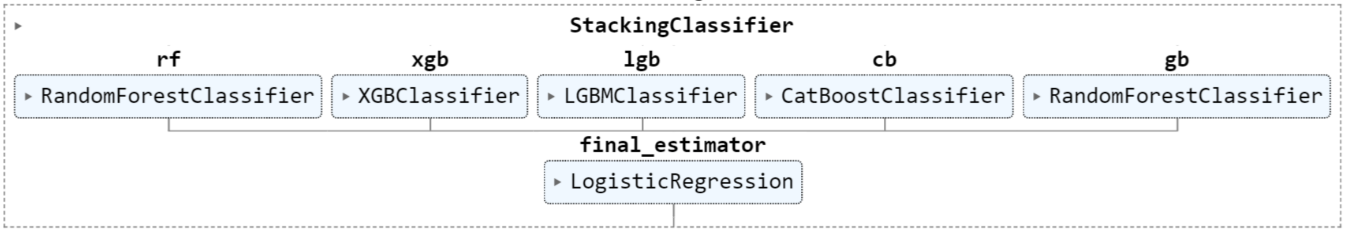
```
Learning rate set to 0.012592
0:  learn: 0.6911581    total: 92ms remaining: 1m 31s
1:  learn: 0.6892107    total: 179ms    remaining: 1m 29s
2:  learn: 0.6873435    total: 252ms    remaining: 1m 23s
3:  learn: 0.6850827    total: 323ms    remaining: 1m 20s
4:  learn: 0.6833879    total: 401ms    remaining: 1m 19s
5:  learn: 0.6817052    total: 474ms    remaining: 1m 18s
6:  learn: 0.6801662    total: 543ms    remaining: 1m 17s
7:  learn: 0.6786194    total: 622ms    remaining: 1m 17s
8:  learn: 0.6769754    total: 695ms    remaining: 1m 16s
9:  learn: 0.6751120    total: 766ms    remaining: 1m 15s
10: learn: 0.6733614    total: 843ms    remaining: 1m 15s
11: learn: 0.6723055    total: 920ms    remaining: 1m 15s
12: learn: 0.6706668    total: 992ms    remaining: 1m 15s
13: learn: 0.6689574    total: 1.07s    remaining: 1m 15s
14: learn: 0.6674405    total: 1.15s    remaining: 1m 15s
15: learn: 0.6659634    total: 1.23s    remaining: 1m 15s
```

....

```
981:    learn: 0.3011695    total: 1m 19s   remaining: 1.46s
982:    learn: 0.3009254    total: 1m 19s   remaining: 1.38s
983:    learn: 0.3007502    total: 1m 19s   remaining: 1.3s
984:    learn: 0.3006521    total: 1m 20s   remaining: 1.22s
985:    learn: 0.3004708    total: 1m 20s   remaining: 1.14s
986:    learn: 0.3003749    total: 1m 20s   remaining: 1.06s
987:    learn: 0.3001303    total: 1m 20s   remaining: 975ms
988:    learn: 0.2999314    total: 1m 20s   remaining: 894ms
989:    learn: 0.2996229    total: 1m 20s   remaining: 813ms
990:    learn: 0.2993063    total: 1m 20s   remaining: 732ms
991:    learn: 0.2991958    total: 1m 20s   remaining: 651ms
992:    learn: 0.2989733    total: 1m 20s   remaining: 570ms
993:    learn: 0.2987368    total: 1m 20s   remaining: 489ms
```

```
994:	learn: 0.2985793	total: 1m 21s	remaining: 408ms
995:	learn: 0.2983046	total: 1m 21s	remaining: 326ms
996:	learn: 0.2981743	total: 1m 21s	remaining: 245ms
997:	learn: 0.2980621	total: 1m 21s	remaining: 163ms
998:	learn: 0.2979987	total: 1m 21s	remaining: 81.7ms
999:	learn: 0.2979361	total: 1m 21s	remaining: 0us
```





In [16]:

```python
# Make predictions using the ensemble classifiers
y_pred_voting = voting_clf.predict(X_test_scaled)
y_pred_stacking = stacking_clf.predict(X_test_scaled)
```

In [17]:

```python
# Get the accuracy of the ensemble classifiers
voting_acc = accuracy_score(y_test, y_pred_voting)
stacking_acc = accuracy_score(y_test, y_pred_stacking)
```

In [18]:

```python
from sklearn.metrics import accuracy_score
# Get the accuracy of each classifier
rf_acc = accuracy_score(y_test, rf_best.predict(X_test_scaled))
xgb_acc = accuracy_score(y_test, xgb_best.predict(X_test_scaled))
lgb_acc = accuracy_score(y_test, lgb_best.predict(X_test_scaled))
cb_acc = accuracy_score(y_test, cb_best.predict(X_test_scaled))
gb_acc = accuracy_score(y_test, gb_best.predict(X_test_scaled))

# Print the accuracies
print('Random Forest Accuracy: %.2f%%' % (rf_acc * 100))
print('XGBoost Accuracy: %.2f%%' % (xgb_acc * 100))
print('LightGBM Accuracy: %.2f%%' % (lgb_acc * 100))
print('CatBoost Accuracy: %.2f%%' % (cb_acc * 100))
print('Gradient Boosting Accuracy: %.2f%%' % (gb_acc * 100))
print("\n")
# Print the accuracy of the ensemble classifiers
print('Voting Classifier Accuracy: %.2f%%' % (voting_acc * 100))
print('Stacking Classifier Accuracy: %.2f%%' % (stacking_acc * 100))
```

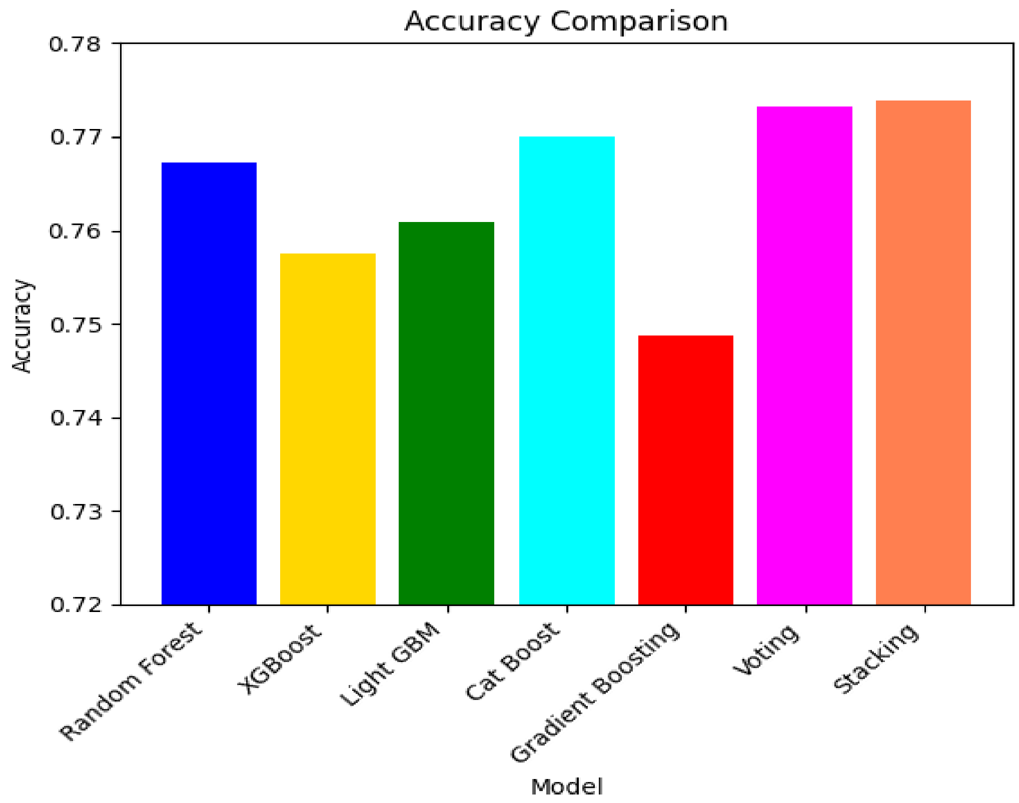
Out [18]:

```
Random Forest Accuracy: 76.72%
XGBoost Accuracy: 75.75%
LightGBM Accuracy: 76.08%
CatBoost Accuracy: 77.00%
Gradient Boosting Accuracy: 74.87%


Voting Classifier Accuracy: 77.33%
Stacking Classifier Accuracy: 77.39%
```

In [19]:

```python
import matplotlib.pyplot as plt
import numpy as np

models = ['Random Forest', 'XGBoost', 'Light GBM', 'Cat Boost',
'Gradient Boosting', 'Voting', 'Stacking']
accuracies = [rf_acc, xgb_acc, lgb_acc, cb_acc, gb_acc, voting_acc,
stacking_acc]

# create bar plot
plt.bar(models, accuracies, color=['blue', 'gold', 'green', 'cyan',
'red', 'magenta', 'coral' ])
plt.xticks(np.arange(len(models)), models, rotation=45, ha='right')

plt.title('Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.72, 0.78)
plt.show()
```

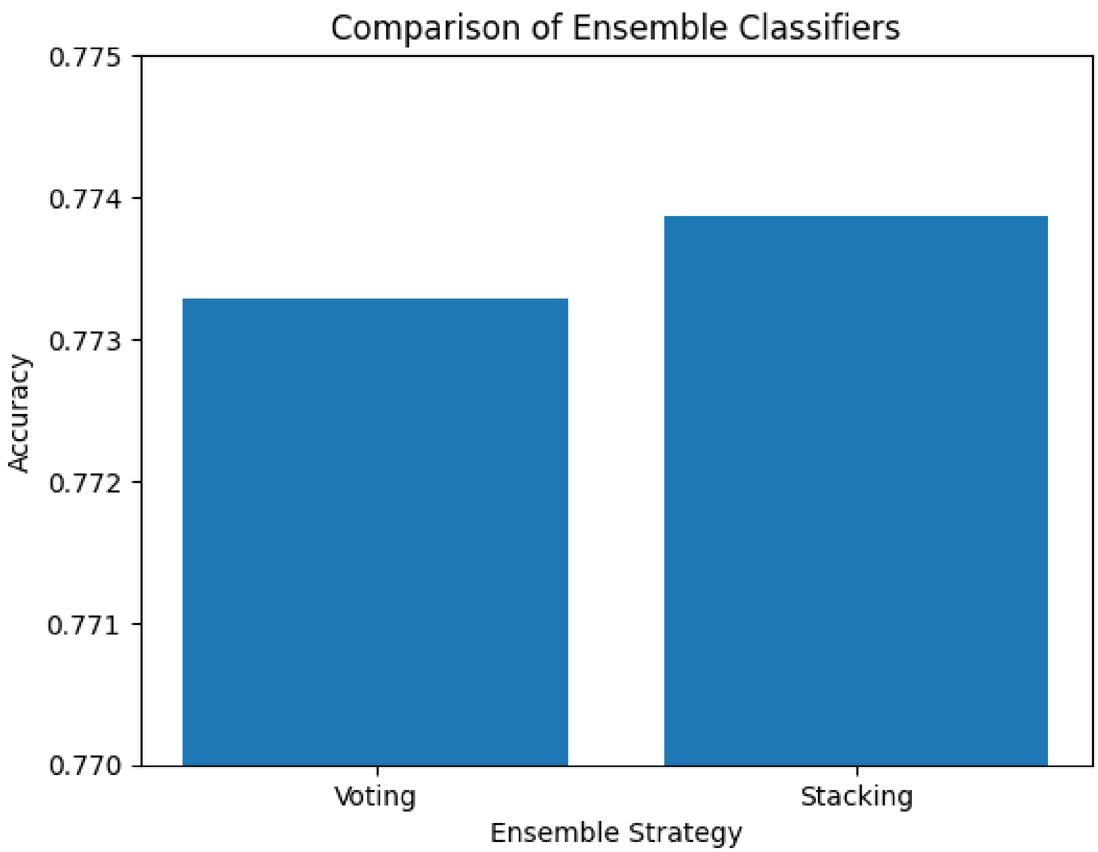
Out [19]:



In [20]:

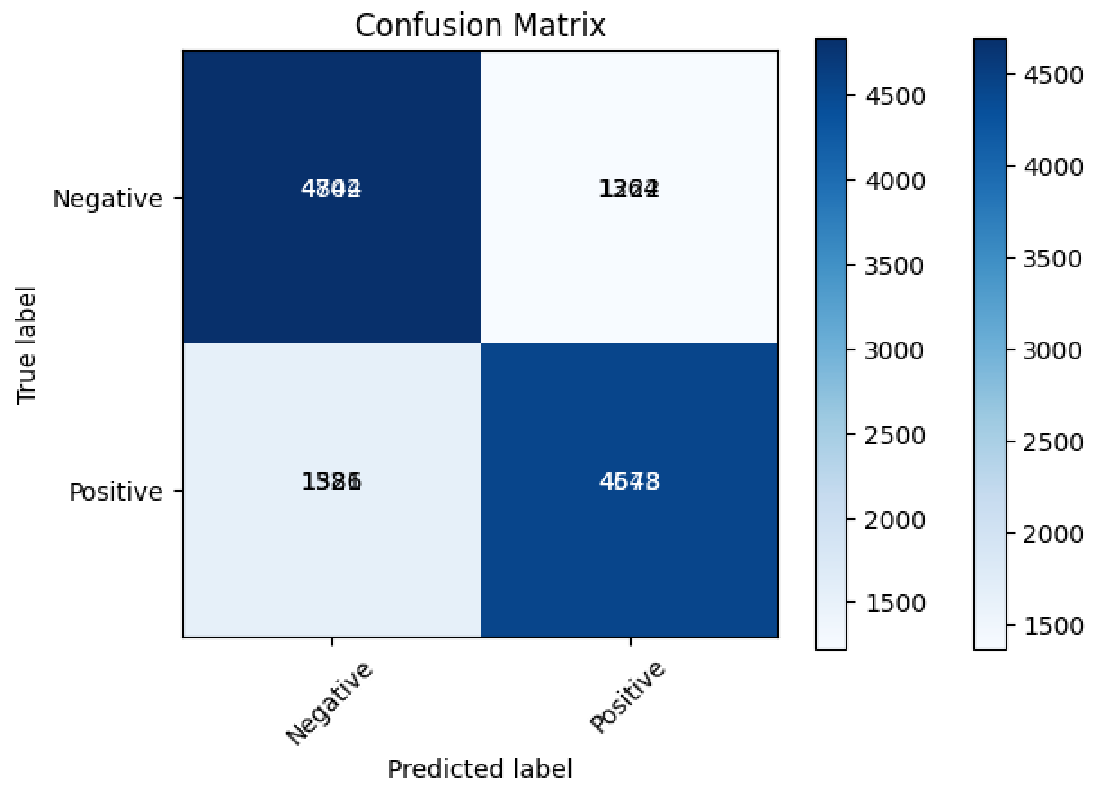
```
import numpy as np
import matplotlib.pyplot as plt

# Create a bar chart to compare the accuracies of the two ensemble classifiers
models = ['Voting', 'Stacking']
accuracies = [voting_acc, stacking_acc]

x_pos = np.arange(len(models))
plt.bar(x_pos, accuracies)
plt.ylim(0.77, 0.775)
plt.title('Comparison of Ensemble Classifiers')
plt.xlabel('Ensemble Strategy')
plt.ylabel('Accuracy')
plt.xticks(x_pos, models)
plt.show()
```

Out [20]:



```
In [21]:
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

# Define a function to plot the confusion matrix
def plot_confusion_matrix(cm, classes, normalize=False,
cmap=plt.cm.Blues):
    """
    This function plots the confusion matrix.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.0
    for i, j in np.ndindex(cm.shape):
        plt.text(j, i, format(cm[i, j], '.2f' if normalize else 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Calculate the confusion matrix for the voting classifier
conf_matrix_voting = confusion_matrix(y_test, y_pred_voting)
print('Confusion Matrix (voting):\n', conf_matrix_voting)
plot_confusion_matrix(conf_matrix_voting, classes=['Negative',
'Positive'])
print("\n")
# Calculate the confusion matrix for the stacking classifier
conf_matrix_stacking = confusion_matrix(y_test, y_pred_stacking)
print('Confusion Matrix (stacking):\n', conf_matrix_stacking)
plot_confusion_matrix(conf_matrix_stacking, classes=['Negative',
'Positive'])
print("\n")
```

```
Out [21]:
Confusion Matrix (voting):
 [[4702 1364]
 [1386 4678]]


Confusion Matrix (stacking):
 [[4844 1222]
 [1521 4543]]
```



In [22]:

```python
# Get the TP, FP, TN, and FN rates for the voting classifier
tn_voting, fp_voting, fn_voting, tp_voting = conf_matrix_voting.ravel(
)
tpr_voting = tp_voting / (tp_voting + fn_voting)
fpr_voting = fp_voting / (fp_voting + tn_voting)
tnr_voting = tn_voting / (tn_voting + fp_voting)
fnr_voting = fn_voting / (tp_voting + fn_voting)

# Print the rates for the voting classifier
print("TP rate (voting):", tpr_voting)
print("FP rate (voting):", fpr_voting)
print("TN rate (voting):", tnr_voting)
print("FN rate (voting):", fnr_voting)

# Get the TP, FP, TN, and FN rates for the stacking classifier
tn_stacking, fp_stacking, fn_stacking, tp_stacking = conf_matrix_stack
ing.ravel()
tpr_stacking = tp_stacking / (tp_stacking + fn_stacking)
fpr_stacking = fp_stacking / (fp_stacking + tn_stacking)
tnr_stacking = tn_stacking / (tn_stacking + fp_stacking)
fnr_stacking = fn_stacking / (tp_stacking + fn_stacking)
```

```
# Print the rates for the stacking classifier
print("TP rate (stacking):", tpr_stacking)
print("FP rate (stacking):", fpr_stacking)
print("TN rate (stacking):", tnr_stacking)
print("FN rate (stacking):", fnr_stacking)
```

Out [22]:

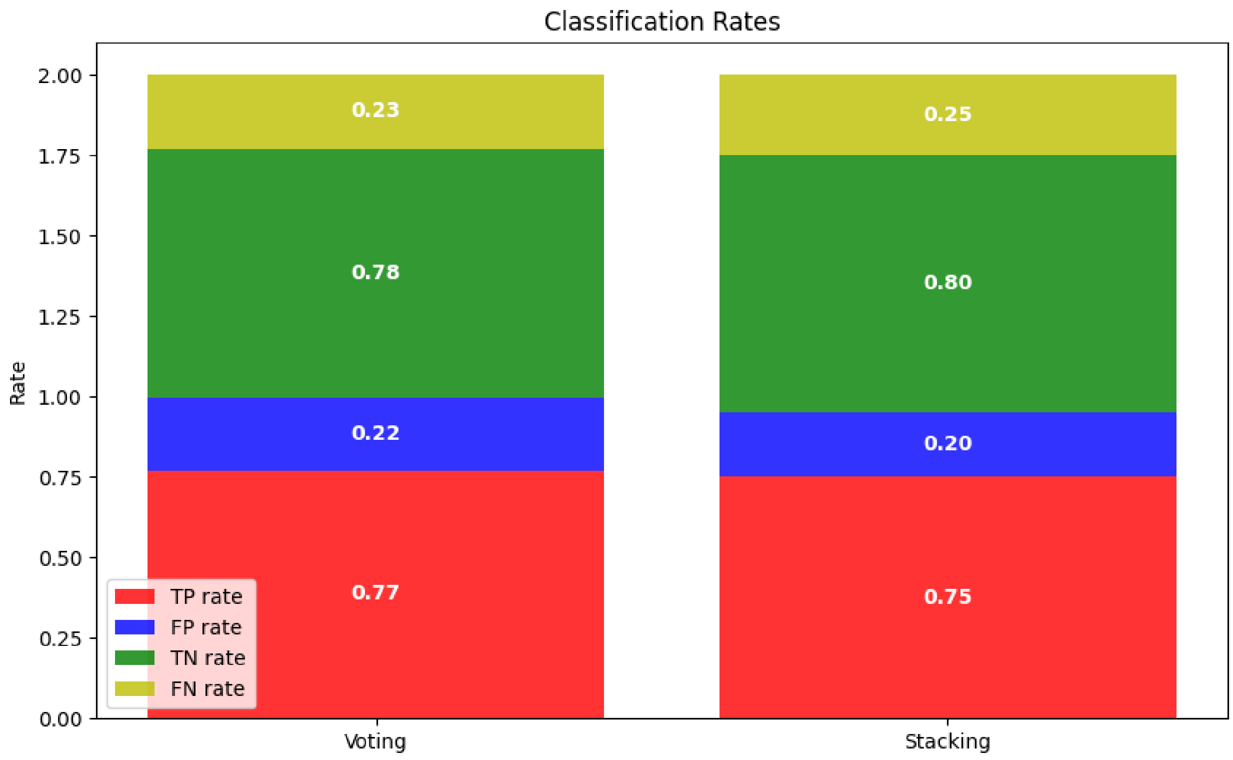
```
TP rate (voting): 0.7714379947229552
FP rate (voting): 0.22485987471150676
TN rate (voting): 0.7751401252884932
FN rate (voting): 0.22856200527704484
TP rate (stacking): 0.7491754617414248
FP rate (stacking): 0.2014507088691065
TN rate (stacking): 0.7985492911308935
FN rate (stacking): 0.2508245382585752
```

In [23]:

```
import matplotlib.pyplot as plt

# Create lists of the rates for the voting and stacking classifiers
tpr_list = [tpr_voting, tpr_stacking]
fpr_list = [fpr_voting, fpr_stacking]
tnr_list = [tnr_voting, tnr_stacking]
fnr_list = [fnr_voting, fnr_stacking]

# Create a list of the classifier names
classifiers = ['Voting', 'Stacking']

# Create a bar plot
fig, ax = plt.subplots(figsize=(10,6))
ax.bar(classifiers, tpr_list, color='r', alpha=opacity, label='TP rate
')
ax.bar(classifiers, fpr_list, color='b', alpha=opacity, label='FP rate
', bottom=tpr_list)
ax.bar(classifiers, tnr_list, color='g', alpha=opacity, label='TN rate
', bottom=[sum(x) for x in zip(tpr_list, fpr_list)])
ax.bar(classifiers, fnr_list, color='y', alpha=opacity, label='FN rate
', bottom=[sum(x) for x in zip(tpr_list, fpr_list, tnr_list)])

# Add value labels to the bars
for i, v in enumerate(tpr_list):
    ax.text(i, v/2, "{:.2f}".format(v), color='white', fontweight='bol
d', ha='center', va='center')
    ax.text(i, v+fpr_list[i]/2, "{:.2f}".format(fpr_list[i]), color='w
hite', fontweight='bold', ha='center', va='center')
```

```
    ax.text(i, v+fpr_list[i]+tnr_list[i]/2, "{:.2f}".format(tnr_list[i
]), color='white', fontweight='bold', ha='center', va='center')
    ax.text(i, v+fpr_list[i]+tnr_list[i]+fnr_list[i]/2, "{:.2f}".forma
t(fnr_list[i]), color='white', fontweight='bold', ha='center', va='cen
ter')

# Add labels and legend
ax.set_ylabel('Rate')
ax.set_title('Classification Rates')
ax.legend()

# Show the plot
plt.show()
```

Out [23]:



```
In [24]:
# Calculate classification report for voting & Stacking classifier
print('Classification Report (voting):\n',
classification_report(y_test, y_pred_voting))

print('Classification Report (stacking):\n',
classification_report(y_test, y_pred_stacking))

# Calculate F1 score, recall, precision, and accuracy for voting
classifier
voting_f1 = classification_report(y_test, y_pred_voting,
output_dict=True)['weighted avg']['f1-score']
voting_recall = classification_report(y_test, y_pred_voting,
output_dict=True)['weighted avg']['recall']
voting_precision = classification_report(y_test, y_pred_voting,
output_dict=True)['weighted avg']['precision']
voting_acc = accuracy_score(y_test, y_pred_voting)

# Calculate F1 score, recall, precision, and accuracy for stacking
classifier
stacking_f1 = classification_report(y_test, y_pred_stacking,
output_dict=True)['weighted avg']['f1-score']
stacking_recall = classification_report(y_test, y_pred_stacking,
output_dict=True)['weighted avg']['recall']
stacking_precision = classification_report(y_test, y_pred_stacking,
output_dict=True)['weighted avg']['precision']
stacking_acc = accuracy_score(y_test, y_pred_stacking)

# Print F1 score, recall, precision, and accuracy for voting and
stacking classifiers
print('F1 Score (voting): %.2f%%' % (voting_f1 * 100))
print('Recall (voting): %.2f%%' % (voting_recall * 100))
print('Precision (voting): %.2f%%' % (voting_precision * 100))
print('Accuracy (voting): %.2f%%' % (voting_acc * 100))
print("\n")
print('F1 Score (stacking): %.2f%%' % (stacking_f1 * 100))
print('Recall (stacking): %.2f%%' % (stacking_recall * 100))
print('Precision (stacking): %.2f%%' % (stacking_precision * 100))
print('Accuracy (stacking): %.2f%%' % (stacking_acc * 100))
```

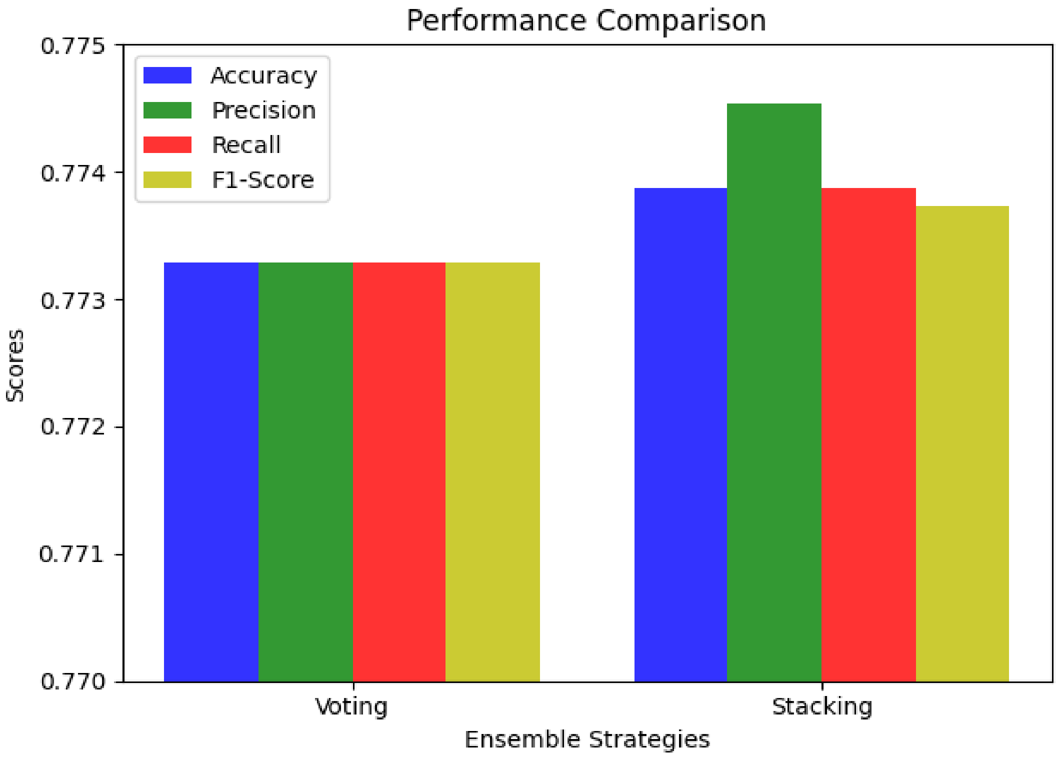
```
Out [24]:
Classification Report (voting):
               precision    recall  f1-score   support

           0       0.77      0.78      0.77      6066
           1       0.77      0.77      0.77      6064

    accuracy                           0.77     12130
   macro avg       0.77      0.77      0.77     12130
weighted avg       0.77      0.77      0.77     12130

Classification Report (stacking):
```

```
              precision    recall  f1-score   support

           0       0.76      0.80      0.78      6066
           1       0.79      0.75      0.77      6064

    accuracy                           0.77     12130
   macro avg       0.77      0.77      0.77     12130
weighted avg       0.77      0.77      0.77     12130

F1 Score (voting): 77.33%
Recall (voting): 77.33%
Precision (voting): 77.33%
Accuracy (voting): 77.33%


F1 Score (stacking): 77.37%
Recall (stacking): 77.39%
Precision (stacking): 77.45%
Accuracy (stacking): 77.39%
```

In [25]:

```python
import matplotlib.pyplot as plt
classifiers = ['Voting', 'Stacking']

accuracy_scores =  [voting_acc, stacking_acc]
precision_scores = [voting_precision, stacking_precision]
recall_scores = [voting_recall, stacking_recall]
f1_scores = [voting_f1, stacking_f1]

fig, ax = plt.subplots()
index = list(range(len(classifiers)))
bar_width = 0.2
opacity = 0.8
plt.ylim(0.77, 0.775)
rects1 = plt.bar(index, accuracy_scores, bar_width,
alpha=opacity,
color='b',
label='Accuracy')

rects2 = plt.bar([i + bar_width for i in index], precision_scores,
bar_width,
alpha=opacity,
color='g',
label='Precision')

rects3 = plt.bar([i + bar_width*2 for i in index], recall_scores,
bar_width,
alpha=opacity,
color='r',
label='Recall')

rects4 = plt.bar([i + bar_width*3 for i in index], f1_scores,
bar_width,
alpha=opacity,
color='y',
label='F1-Score')

plt.xlabel('Ensemble Strategies')
plt.ylabel('Scores')
plt.title('Performance Comparison')
plt.xticks([i + bar_width*1.5 for i in index], classifiers)
plt.legend()

plt.tight_layout()
plt.show()
```

Out [25]:

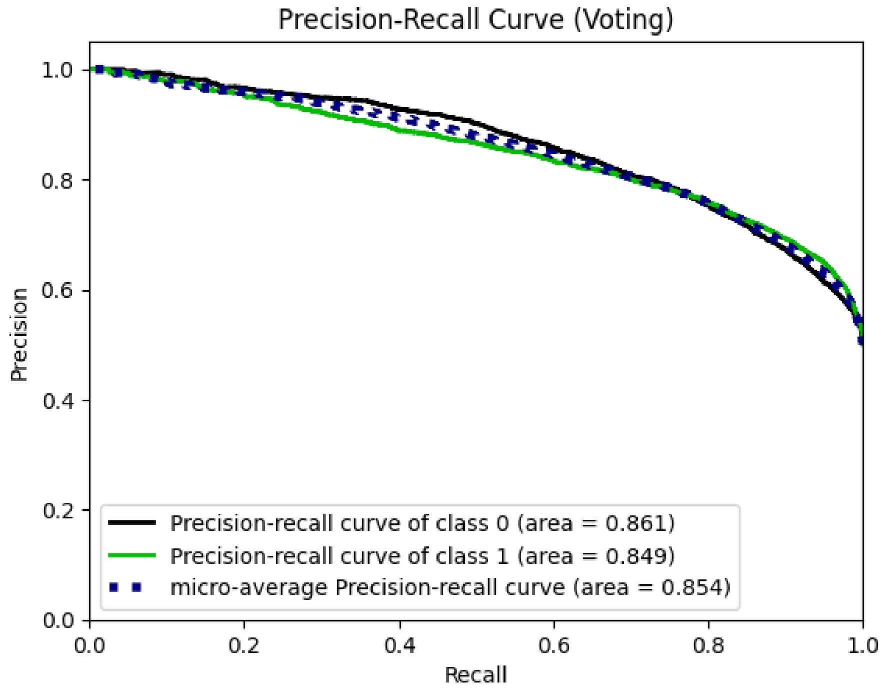


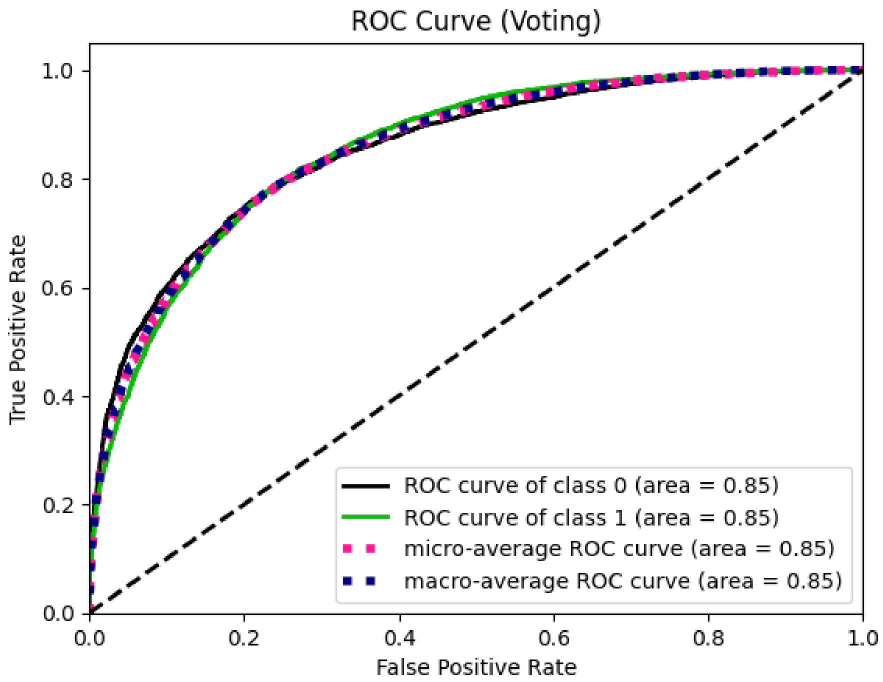
In [26]:

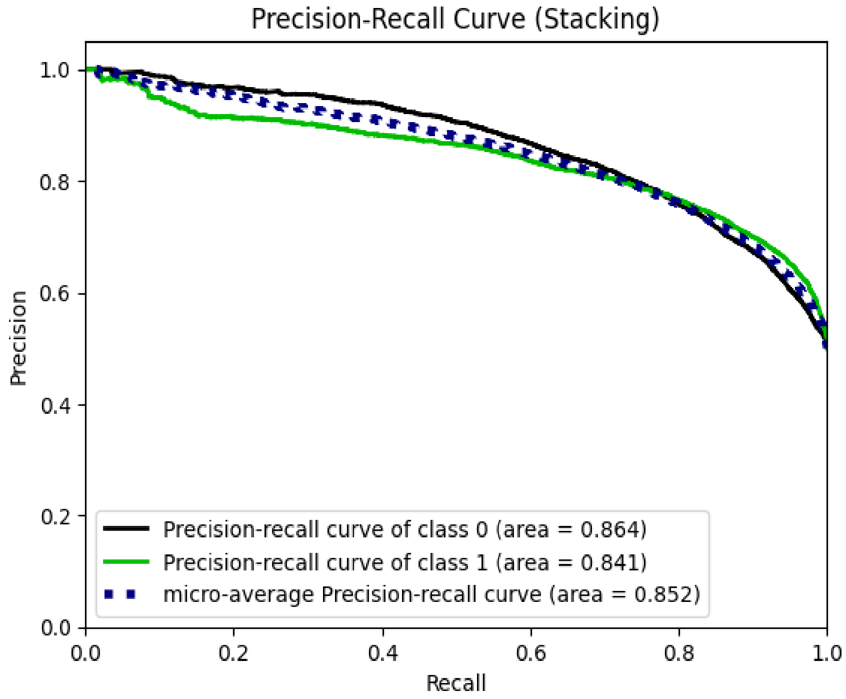
```
import scikitplot as skplt
```

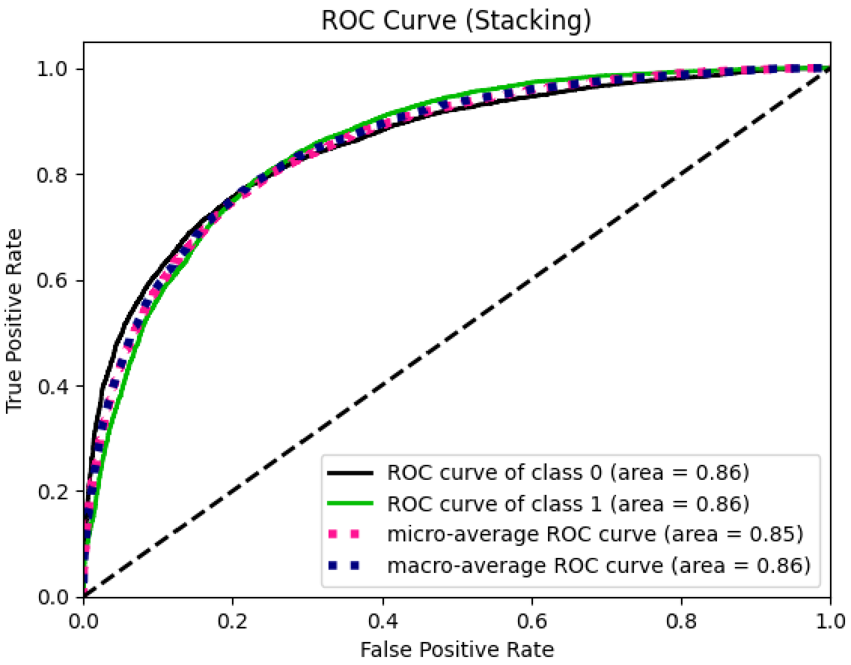
```
# Generate precision-recall curve for the voting classifier
skplt.metrics.plot_precision_recall(y_test, voting_clf.predict_proba(X_test_scaled))
plt.title('Precision-Recall Curve (Voting)')

# Generate ROC curve for the voting classifier
skplt.metrics.plot_roc(y_test, voting_clf.predict_proba(X_test_scaled))
plt.title('ROC Curve (Voting)')

# Generate precision-recall curve for the stacking classifier
skplt.metrics.plot_precision_recall(y_test, stacking_clf.predict_proba(X_test_scaled))
plt.title('Precision-Recall Curve (Stacking)')

# Generate ROC curve for the stacking classifier
skplt.metrics.plot_roc(y_test, stacking_clf.predict_proba(X_test_scaled))
plt.title('ROC Curve (Stacking)')
```

```
Out [26]: Text(0.5, 1.0, 'ROC Curve (Stacking)')
```




ROC Curve (Voting)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.85)
ROC curve of class 1 (area = 0.85)
micro-average ROC curve (area = 0.85)
macro-average ROC curve (area = 0.85)


Precision-Recall Curve (Stacking)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class 0 (area = 0.864)
Precision-recall curve of class 1 (area = 0.841)
micro-average Precision-recall curve (area = 0.852)


ROC Curve (Stacking)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.86)
ROC curve of class 1 (area = 0.86)
micro-average ROC curve (area = 0.85)
macro-average ROC curve (area = 0.86)