ML PROJECT

Libraries

```
Main Libraries
import pandas as pd
import numpy as np
import sklearn as sk
Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
Text Preprocessing
import neattext.functions as nfx
import nltk
from nltk import word tokenize
from nltk.stem import WordNetLemmatizer
from nltk import pos tag
#nltk.download('wordnet') -> dependencies for nltk tokenization and
lemmatization
#nltk.download('omw-1.4')
#nltk.download('punkt')
#nltk.download('averaged_perceptron tagger')
Pipelines
from sklearn.pipeline import Pipeline
Vectorizers And Transformers
from sklearn.feature extraction.text import CountVectorizer
from sklearn.feature extraction.text import TfidfTransformer
Algorithms And Ensembles
from sklearn.svm import SVC
from sklearn.linear model import SGDClassifier
from sklearn.naive bayes import MultinomialNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import StackingClassifier
Split Data
from sklearn.model_selection import train_test_split
Cross Validation
from sklearn.model selection import cross val predict,
StratifiedKFold, cross val score
```

```
Metrics
```

```
from sklearn.metrics import classification report, confusion matrix
```

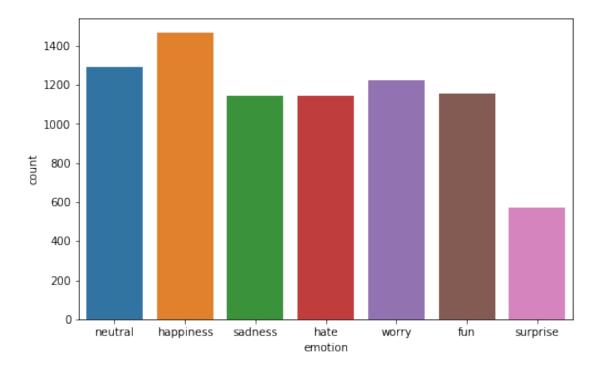
Parameter Tuning

```
from sklearn.model selection import GridSearchCV
```

```
Get Data
```

Data Exploration

```
df.shape
(8000, 2)
df.dtypes
text
           object
emotion
           object
dtype: object
df.isnull().sum()
text
emotion
           0
dtype: int64
df.groupby(['emotion']).count()
           text
emotion
           1157
fun
           1468
happiness
hate
           1143
neutral
           1292
sadness
           1146
surprise
           572
           1222
worry
plt.figure(figsize=(8,5))
sns.countplot(x='emotion', data=df)
plt.show()
```



Encode Label

```
def define class(emotion):
    if emotion == 'sadness':
        return 0
    elif emotion == 'neutral':
        return 1
    elif emotion == 'worry':
        return 2
    elif emotion == 'happiness':
        return 3
    elif emotion == 'fun':
        return 4
    elif emotion == 'surprise':
        return 5
    elif emotion == 'hate':
        return 6
    else:
        raise Exception('Not a valid emotion')
df['emotion_class'] = df['emotion'].apply(define_class)
df.head()
                                                text
                                                        emotion
emotion class
  Yeah for coupons! Found this place randomly ...
                                                        neutral
1
   i still love to feel a gentle breeze and hear ... happiness
```

```
Normal, fama devido ao programa de tv. A sobre...
                                                         neutral
1
3
                      im feeling a little less jaded
                                                         sadness
0
4
   i have never shaken the feeling of ferocious p...
                                                           hate
df = df[['text','emotion class']]
df.head()
                                                      emotion class
                                                text
  Yeah for coupons! Found this place randomly ...
   i still love to feel a gentle breeze and hear ...
                                                                   3
                                                                   1
  Normal, fama devido ao programa de tv. A sobre...
3
                      im feeling a little less jaded
                                                                   0
4
                                                                   6
  i have never shaken the feeling of ferocious p...
Preprocess Text
df['new text'] = df['text'].apply(lambda x: x.lower())
df['new_text'] = df['new_text'].apply(nfx.remove_multiple_spaces)
df['new text'] = df['new text'].apply(nfx.remove punctuations)
df['new_text'] = df['new_text'].apply(nfx.remove_puncts)
df['new text'] = df['new text'].apply(nfx.remove stopwords)
df['new_text'] = df['new_text'].apply(nfx.remove_emojis)
df['new text'] = df['new text'].apply(nfx.remove special characters)
df['new text'] = df['new text'].apply(nfx.remove bad quotes)
df['new_text'] = df['new_text'].apply(nfx.remove_non_ascii)
df['new text'] = df['new text'].apply(nfx.remove accents)
df['new_text'] = df['new_text'].apply(nfx.remove_urls)
df['new text'] = df['new text'].apply(nfx.remove html tags)
df['new text'] = df['new text'].apply(nfx.remove userhandles)
df['new_text'] = df['new_text'].apply(nfx.remove_hashtags)
df['new text'] = df['new text'].apply(nfx.remove phone numbers)
df.head()
                                                      emotion class
                                                text
  Yeah for coupons!
                       Found this place randomly ...
                                                                   1
                                                                   3
   i still love to feel a gentle breeze and hear ...
                                                                   1
  Normal, fama devido ao programa de tv. A sobre...
                      im feeling a little less jaded
3
                                                                   0
   i have never shaken the feeling of ferocious p...
                                                                   6
   yeah coupons found place randomly like lot dif...
   love feel gentle breeze hear peal soft wind ch...
1
   normal fama devido ao programa de tv sobremesa...
3
                             im feeling little jaded
4
      shaken feeling ferocious protectiveness mother
```

```
df = df[['new text','emotion class']]
df.head()
                                             new text emotion class
   yeah coupons found place randomly like lot dif...
  love feel gentle breeze hear peal soft wind ch...
                                                                   3
  normal fama devido ao programa de tv sobremesa...
                                                                   1
                              im feeling little jaded
                                                                   0
4
      shaken feeling ferocious protectiveness mother
Setup Tokenization And Lemmatization Parameter
# Lemmatization is slower than stemming but it is generally more
effective. It gives more context and meaning to words. It preserves
them
class LemmaTokenizer:
    def init (self):
        self.lemma = WordNetLemmatizer()
    def call (self, text):
        tokenized = word tokenize(text)
        lemmatized=[]
        for token, tag in pos tag(tokenized):
            pos=tag[0].lower()
            if pos not in ['a', 'r', 'n', 'v']:
                pos='n'
            lemmatized.append(self.lemma.lemmatize(token,pos))
        return lemmatized
Setup Folds With StratifiedKFold
folds = StratifiedKFold(n splits = 10, shuffle = True, random state =
1000) # since dataset is somewhat imbalanced (1 class)
Base Models With Estimators And Transformers
target names =
['sadness', 'neutral', 'worry', 'happiness', 'fun', 'surprise', 'hate']
SVM
svm = Pipeline([
    ('vect', CountVectorizer(tokenizer=LemmaTokenizer())),
    ('tfidf', TfidfTransformer()),
    ('svm', SVC(decision function shape='ovr')),
1)
svm prediction = cross val predict(svm, df['new text'],
df['emotion_class'], cv = \overline{folds})
svm score = cross val score(svm, df['new text'], df['emotion class'],
```

```
cv = folds)
svm_report = classification_report(df['emotion_class'],
svm_prediction, target_names=target_names, output_dict=True)
plt.figure(figsize=(8, 5))
plt.title('SVM Metrics', fontsize=15)
sns.heatmap(pd.DataFrame(svm_report).T, annot=True, cmap='Blues',
fmt='g')
```

<AxesSubplot:title={'center':'SVM Metrics'}>

weighted avg -

0.856878

precision

8000 sadness -0.832281 0.80541 0.818625 1146 7000 0.981424 0.987924 1292 neutral -0.99451 0.833333 0.842881 0.83808 1222 worry -6000 1468 happiness -0.827213 0.935967 0.878236 - 5000 0.791134 0.802074 0.796567 fun -1157 - 4000 0.68007 surprise -0.813808 0.740952 572 - 3000 hate -0.877341 0.819773 0.84758 1143 accuracy -0.856125 0.856125 0.856125 0.856125 - 2000 macro avg -0.852803 0.838228 0.843995 8000

0.85527

fl-score

8000

support

- 1000

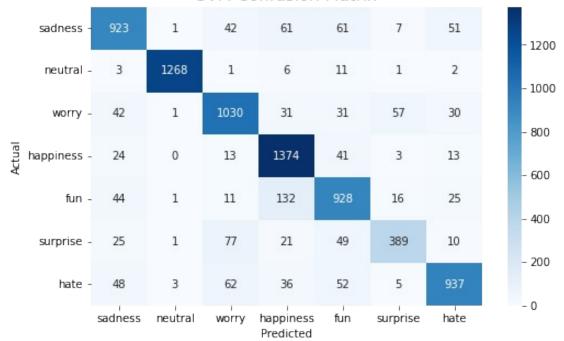
SVM Metrics

```
svm_matrix = confusion_matrix(df['emotion_class'], svm_prediction)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(svm_matrix, target_names, target_names),
annot=True, cmap='Blues', fmt='g')
plt.title('SVM Confusion Matrix', fontsize=15)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

0.856125

recall

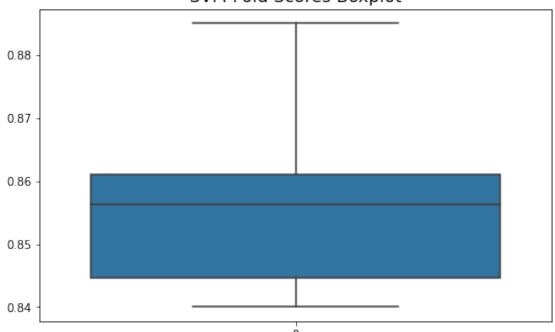
SVM Confusion Matrix



plt.figure(figsize=(8, 5))
plt.title('SVM Fold Scores Boxplot', fontsize=15)
sns.boxplot(data=svm_score)

<AxesSubplot:title={'center':'SVM Fold Scores Boxplot'}>

SVM Fold Scores Boxplot



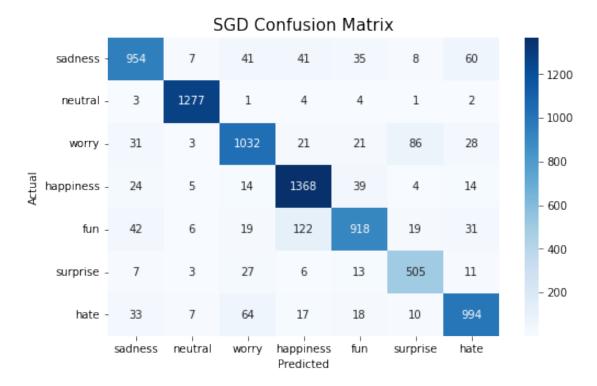
```
SGD
sgd = Pipeline([
    ('vect', CountVectorizer(tokenizer=LemmaTokenizer())),
    ('tfidf', TfidfTransformer()),
    ('sgd', OneVsRestClassifier(SGDClassifier())),
1)
sqd prediction = cross val predict(sqd, df['new text'],
df['emotion class'], cv = folds)
sgd score = cross val score(sgd, df['new text'], df['emotion class'],
cv = folds)
sgd report = classification report(df['emotion class'],
sgd prediction, target names=target names, output dict=True)
plt.figure(figsize=(8, 5))
plt.title('SGD Metrics', fontsize=15)
sns.heatmap(pd.DataFrame(sqd report).T, annot=True, cmap='Blues',
fmt='q')
```

<AxesSubplot:title={'center':'SGD Metrics'}>

SGD Metrics 8000 0.872029 0.832461 0.851786 1146 sadness -7000 0.98839 0.982308 neutral -0.9763 1292 worry -0.861436 0.844517 0.852893 1222 6000 0.93188 happiness -0.866371 0.897932 1468 - 5000 fun -0.875954 0.793431 0.832653 1157 - 4000 0.882867 surprise -0.797788 0.838174 572 - 3000 0.87193 0.869641 0.870784 1143 hate accuracy -0.881 0.881 0.881 0.881 - 2000 macro avg -0.874544 0.877598 0.875219 8000 - 1000 weighted avg -0.881458 0.881 0.880476 8000 recall fl-score precision support

```
sgd_matrix = confusion_matrix(df['emotion_class'], sgd_prediction)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(sgd_matrix, target_names, target_names),
annot=True, cmap='Blues', fmt='g')
plt.title('SGD Confusion Matrix', fontsize=15)
plt.ylabel('Actual')
```

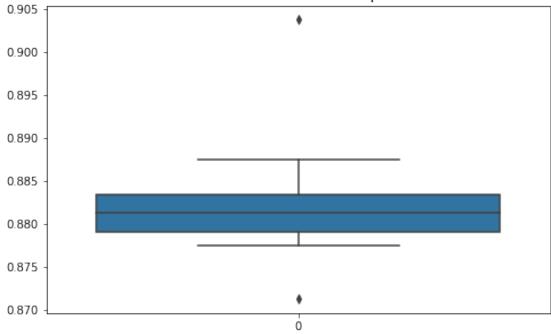
plt.xlabel('Predicted') plt.show()



```
plt.figure(figsize=(8, 5))
plt.title('SGD Fold Scores Boxplot', fontsize=15)
sns.boxplot(data=sgd_score)
```

<AxesSubplot:title={'center':'SGD Fold Scores Boxplot'}>





```
Naive Bayes (Multinomial)
bayes = Pipeline([
          ('vect', CountVectorizer(tokenizer=LemmaTokenizer())),
          ('tfidf', TfidfTransformer()),
          ('bayes', OneVsRestClassifier(MultinomialNB())),
])

bayes_prediction = cross_val_predict(bayes, df['new_text'],
df['emotion_class'], cv = folds)
bayes_score = cross_val_score(bayes, df['new_text'],
df['emotion_class'], cv = folds)

bayes_report = classification_report(df['emotion_class'],
bayes_prediction,target_names=target_names, output_dict=True)
plt.figure(figsize=(8, 5))
plt.title('MultinomialNB Metrics', fontsize=15)
sns.heatmap(pd.DataFrame(bayes_report).T, annot=True, cmap='Blues', fmt='g')
```

<AxesSubplot:title={'center':'MultinomialNB Metrics'}>

MultinomialNB Metrics 8000 0.779661 0.875952 0.702443 1146 sadness -- 7000 0.782635 0.997678 0.877169 1292 neutral worry -0.758327 0.875614 0.812761 1222 6000 happiness -0.681818 0.940054 0.790378 1468 - 5000 0.870486 0.604149 fun -0.713265 1157 - 4000 surprise -0.936416 0.283217 0.434899 572 - 3000 hate -0.8739 0.782152 0.825485 1143 accuracy -0.787375 0.787375 0.787375 0.787375 - 2000 0.825648 0.740758 0.74766 8000 macro avg -- 1000 0.787375 0.774725 8000 weighted avg -0.81053

```
bayes_matrix = confusion_matrix(df['emotion_class'], bayes_prediction)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(bayes_matrix, target_names, target_names),
annot=True, cmap='Blues', fmt='g')
plt.title('MultinomialNB Confusion Matrix', fontsize=15)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

fl-score

support

recall

precision

MultinomialNB Confusion Matrix sadness -- 1200 neutral -- 1000 worry -- 800 happiness -- 600 fun -- 400 surprise -- 200

plt.figure(figsize=(8, 5))
plt.title('MultinomialNB Fold Scores Boxplot', fontsize=15)
sns.boxplot(data=bayes_score)

worry

sadness

neutral

hate -

<AxesSubplot:title={'center':'MultinomialNB Fold Scores Boxplot'}>

happiness

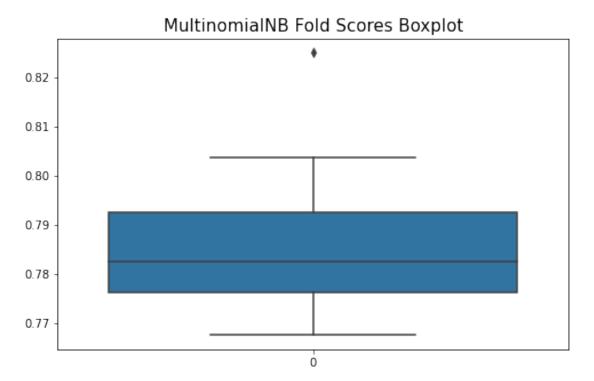
Predicted

fun

hate

- 0

surprise



Parameter Tuning (GridSearchCV)

```
Train Test Split
x_train, x_test, y_train, y_test =
train test split(df['new text'],df['emotion class'],shuffle=True,strat
ify=df['emotion class'],test size=0.2,random state=20)
Pipelines
pipe_svm = Pipeline([('vect',
CountVectorizer(tokenizer=LemmaTokenizer())),('transformer',
TfidfTransformer()),('svm', SVC(decision_function_shape='ovr'))])
pipe sgd = Pipeline([('vect',
CountVectorizer(tokenizer=LemmaTokenizer())),('transformer',
TfidfTransformer()),('sgd', OneVsRestClassifier(SGDClassifier()))])
pipe bayes = Pipeline([('vect',
CountVectorizer(tokenizer=LemmaTokenizer())),('transformer',
TfidfTransformer()),('mnb', OneVsRestClassifier(MultinomialNB()))])
Parameters
# Support Vector Machine parameters to test
svm params = { 'svm C' : [0.1, 1, 10, 100, 1000], }
              'svm__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'svm kernel': ('linear','rbf'),
              'vect ngram range': [(1, 1), (1, 2)]
             }
# SGD Classifier parameters to test
sqd params = {
    "sgd loss" : ["hinge", "log loss", "squared hinge",
"modified huber"],
    "sgd__alpha" : [0.0001, 0.001, 0.01, 0.1],
    "sgd__penalty" : ["l2", "l1", "none"],
    "sgd learning rate":
["constant", "optimal", "invscaling", "adaptive"],
    "sgd class_weight": [None, 'balanced'],
    'vect ngram range': [(1, 1), (1, 2)],
# Bayes Classifier parameters to test
bayes params = {
    "mnb alpha" : [0.01, 0.1, 0.5, 1.0, 10.0],
    "mnb fit prior" : (True, False),
    "vect ngram range": [(1,1),(1,2),(1,6)]
               }
Process
# Setting up grids
grid svm = GridSearchCV(pipe svm, parameters, refit = True, verbose =
3)
grid sgd = GridSearchCV(pipe sgd, parameters, refit = True, verbose =
```

```
3)
grid bayes = GridSearchCV(pipe bayes, parameters, refit = True,
verbose = 3)
# Fitting the grid
grid.fit(x train, y train)
grid.fit(x train, y train)
grid.fit(x train, y train)
Results
# Support Vector Machine
{'svc__C': 1, 'svc__gamma': 1, 'svc__kernel': 'linear',
'vect ngram range': (1, 2)}
{'svc C': 1,
 'svc__gamma': 1,
 'svc kernel': 'linear',
 'vect ngram range': (1, 2)}
# SGD Classifier
{'sgd_loss': 'hinge', 'sgd_alpha': '0.0001', 'sgd penalty': 'l1',
'sgd_learning_rate': 'optimal', 'sqd class weight': None.
'vect ngram range': (1,2)}
{'sqd loss': 'hinge',
 'sgd<u>    </u>alpha': '0.0001',
 'sqd penalty': 'l1',
 'sgd learning rate': 'optimal',
 'sgd__class_weight': None,
 'vect ngram range': (1, 2)}
# Baves Classifier
{'mb alpha': 0.1, 'mb fit prior': False, 'vect ngram range': (1,6)}
{'mb_alpha': 0.1, 'mb_fit_prior': False, 'vect_ngram_range': (1,
6)}
After Tuning
SVM
svm after = Pipeline([
    ('vect'.
CountVectorizer(tokenizer=LemmaTokenizer(),ngram range=(1,2))),
    ('tfidf', TfidfTransformer()),
SVC(C=1,gamma=1,kernel='linear',decision function shape='ovr')),
])
svm prediction after = cross val predict(svm after, df['new text'],
df['emotion class'], cv = folds)
svm score after = cross val score(svm after, df['new text'],
```

```
df['emotion_class'], cv = folds)

svm_report_after = classification_report(df['emotion_class'],
    svm_prediction_after, target_names=target_names, output_dict=True)

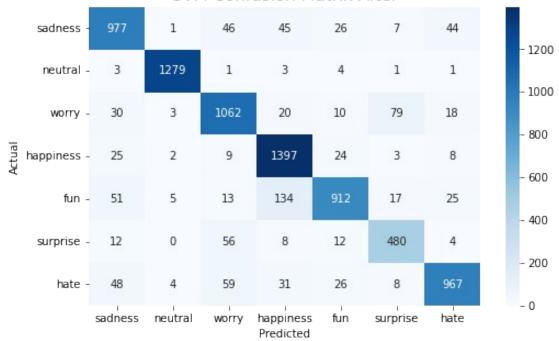
plt.figure(figsize=(8, 5))
    plt.title('SVM Metrics After', fontsize=15)
    sns.heatmap(pd.DataFrame(svm_report_after).T, annot=True,
    cmap='Blues', fmt='g')
```

<AxesSubplot:title={'center':'SVM Metrics After'}>

SVM Metrics After 8000 0.852531 0.852531 0.852531 1146 sadness -7000 0.989938 0.989172 1292 neutral -0.988408 0.869067 0.860616 1222 worry -0.852327 6000 1468 happiness -0.852869 0.951635 0.899549 - 5000 0.899408 0.788245 0.840166 fun -1157 - 4000 surprise -0.806723 0.839161 0.822622 572 - 3000 0.846019 hate -0.906279 0.875113 1143 accuracy -0.88425 0.88425 0.88425 0.88425 - 2000 0.87711 macro avg -0.879792 0.876657 8000 - 1000 weighted avg -0.88569 0.88425 0.883761 8000 precision recall fl-score support

```
svm_matrix_after = confusion_matrix(df['emotion_class'],
svm_prediction_after)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(svm_matrix_after, target_names,
target_names), annot=True, cmap='Blues', fmt='g')
plt.title('SVM Confusion Matrix After', fontsize=15)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

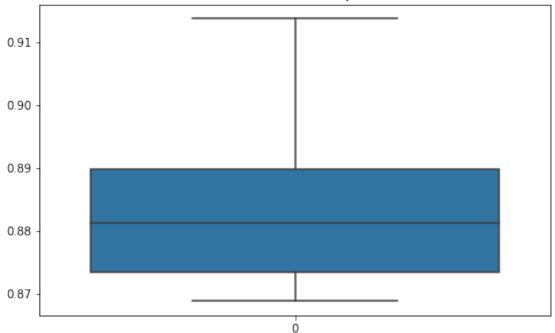
SVM Confusion Matrix After



plt.figure(figsize=(8, 5))
plt.title('SVM Fold Scores Boxplot After', fontsize=15)
sns.boxplot(data=svm_score_after)

<AxesSubplot:title={'center':'SVM Fold Scores Boxplot After'}>

SVM Fold Scores Boxplot After



```
SGD
sgd after = Pipeline([
    ('vect',
CountVectorizer(tokenizer=LemmaTokenizer(),ngram range=(1,2))),
    ('tfidf', TfidfTransformer()),
OneVsRestClassifier(SGDClassifier(loss='hinge',alpha=0.0001,penalty='l
1',learning_rate='optimal',class_weight=None))),
1)
sgd prediction after = cross val predict(sgd after, df['new text'],
df['emotion_class'], cv = folds)
sqd score after = cross val score(sqd after, df['new text'],
df['emotion class'], cv = folds)
sgd_report_after = classification_report(df['emotion_class'],
sqd prediction after, target names=target names, output dict=True)
plt.figure(figsize=(8, 5))
plt.title('SGD Metrics After', fontsize=15)
sns.heatmap(pd.DataFrame(sqd report after).T, annot=True,
cmap='Blues', fmt='g')
<AxesSubplot:title={'center':'SGD Metrics After'}>
```

SGD Metrics After 8000 0.877495 0.843805 0.86032 1146 sadness -7000 neutral -0.947329 0.98839 0.967424 1292 0.868654 0.88216 0.875355 1222 worry -6000 happiness -0.88537 0.952316 0.917624 1468 - 5000 0.894789 0.786517 0.837167 1157 fun -- 4000 0.891608 surprise -0.82391 0.856423 572 - 3000 0.91042 0.871391 0.890478 hate -1143 accuracy -0.892 0.892 0.892 0.892 - 2000 0.888027 0.886399 8000 macro avg -0.886853 - 1000 weighted avg -0.892242 0.892 0.891111 8000 recall precision f1-score support

```
sgd_matrix_after = confusion_matrix(df['emotion_class'],
sgd_prediction_after)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(sgd matrix after, target names,
```

```
target_names), annot=True, cmap='Blues', fmt='g')
plt.title('SGD Confusion Matrix After', fontsize=15)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

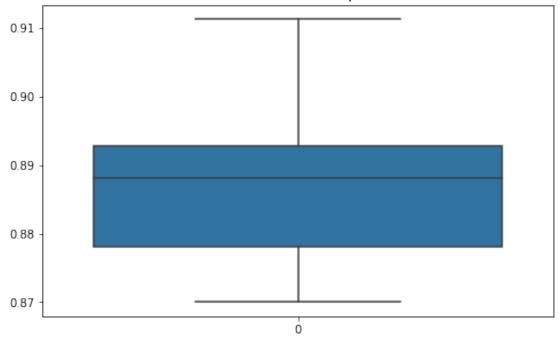
SGD Confusion Matrix After



```
plt.figure(figsize=(8, 5))
plt.title('SGD Fold Scores Boxplot After', fontsize=15)
sns.boxplot(data=sgd_score_after)
```

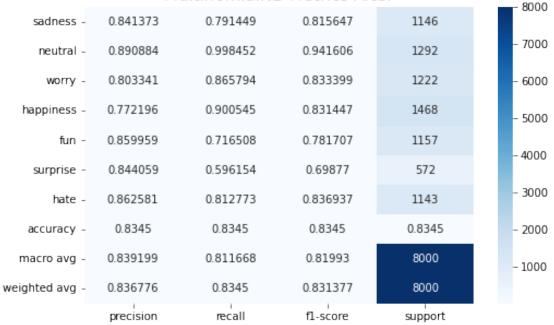
<AxesSubplot:title={'center':'SGD Fold Scores Boxplot After'}>

SGD Fold Scores Boxplot After

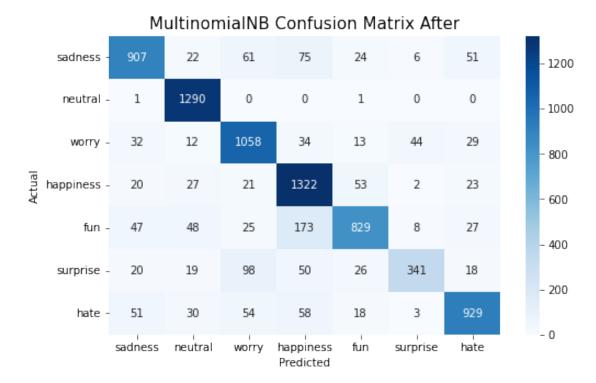


```
Naive Bayes (Multinomial)
mnb after = Pipeline([
    ('vect',
CountVectorizer(tokenizer=LemmaTokenizer(),ngram range=(1,6))),
    ('tfidf', TfidfTransformer()),
    ('mnb',
OneVsRestClassifier(MultinomialNB(alpha=0.1,fit prior=False))),
1)
mnb prediction after = cross val predict(mnb after, df['new text'],
df['emotion_class'], cv = folds)
mnb score after = cross val score(mnb after, df['new text'],
df['emotion class'], cv = folds)
mnb report after = classification report(df['emotion class'],
mnb prediction after, target names=target names, output dict=True)
plt.figure(figsize=(8, 5))
plt.title('MultinomialNB Metrics After', fontsize=15)
sns.heatmap(pd.DataFrame(mnb report after).T, annot=True,
cmap='Blues', fmt='g')
<AxesSubplot:title={'center':'MultinomialNB Metrics After'}>
```

MultinomialNB Metrics After

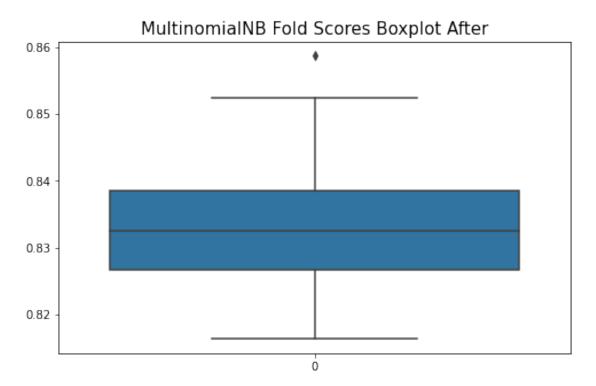


```
mnb_matrix_after = confusion_matrix(df['emotion_class'],
mnb_prediction_after)
plt.figure(figsize=(8, 5))
sns.heatmap(pd.DataFrame(mnb_matrix_after, target_names,
target_names), annot=True, cmap='Blues', fmt='g')
plt.title('MultinomialNB Confusion Matrix After', fontsize=15)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



plt.figure(figsize=(8, 5))
plt.title('MultinomialNB Fold Scores Boxplot After', fontsize=15)
sns.boxplot(data=mnb_score_after)

<AxesSubplot:title={'center':'MultinomialNB Fold Scores Boxplot
After'}>



Winner

And the winner is...SGDClassifier

```
Stacking
estimators =
[('svc',SVC(C=1,gamma=1,kernel='linear',decision_function_shape='ovr')
final estimator =
OneVsRestClassifier(SGDClassifier(loss='hinge',alpha=0.0001,penalty='l
1',learning rate='optimal',class weight=None))
stack = StackingClassifier(estimators=estimators,
final estimator=final estimator,cv=10)
stacking pipe = Pipeline([('vect',
CountVectorizer(tokenizer=LemmaTokenizer(),ngram range=(1,2))),
('tfidf', TfidfTransformer()),('stack', stack)])
stacking pipe.fit(x train, y train)
Pipeline(steps=[('vect',
                 CountVectorizer(ngram range=(1, 2),
                                 tokenizer=<__main__.LemmaTokenizer</pre>
object at 0x00000206A7911708>)).
                ('tfidf', TfidfTransformer()),
                ('stack',
                 StackingClassifier(cv=10,
                                     estimators=[('svc',
                                                  SVC(C=1, gamma=1,
kernel='linear'))],
final estimator=OneVsRestClassifier(estimator=SGDClassifier(penalty='l
1'))))])
stacking pipe.score(x test,y test)
0.88875
stacking pipe.score(x train,y train)
0.99
```

Results

SGDClassifier alone still wins