NLP_aggresive_classifier_final

July 10, 2020

1 Implementation

- Purpose of this project is to analyse the Jigsaw comment dataset and clssify the data based on toxicity
- Three algorithms were tested based on the Naive Bayes, Logistic regression and support vector machine for different values of n grams and TFIDF norm values.
- Appropriate value of 'n' in n-grams is slected using the result from above algorithms also better norm for TFIDF is selected.
- Algorithms giving good performance are takena as baseline
- Finally hyperparameters of the algorithms are tuned for better results.

2 Data

Jigsaw toxic comment data is chosen from kaggle competition for training and testing purpose. Jigsaw toxic comment data [2] is a set of data containing toxic comments and binary indicators to represent whether the data is toxic or non-toxic also level of toxicity in the message is also presented.

Comments are in five categories: - severe_toxic
- obscene
- threat - insult
- identity_hate

3 Loading necessary dependecy for operation

[1]:
```
import nltk
import pickle
import re
import time
import warnings
import csv
import requests

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import itertools
import traceback
from wordcloud import WordCloud

from sklearn import preprocessing
from nltk.stem.porter import *

from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
# from performance import *
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix
from collections import Counter
```

[2]:
```
def telegram_bot_sendtext(bot_message):

    bot_token = ''
    bot_chatID = ''
    send_text = 'https://api.telegram.org/bot' + bot_token + '/sendMessage?
↪chat_id=' + bot_chatID + '&parse_mode=Markdown&text=' + bot_message

    response = requests.get(send_text)

    bot_token_alan = ''
    bot_chatID_alan = ''
    send_text_alan = 'https://api.telegram.org/bot' + bot_token_alan + '/
↪sendMessage?chat_id=' + bot_chatID_alan + '&parse_mode=Markdown&text=' +␣
↪bot_message

    response = requests.get(send_text_alan)

    return response.json()
```

[3]:
```
print(pickle.compatible_formats)
```

```
['1.0', '1.1', '1.2', '1.3', '2.0', '3.0', '4.0']
```

4 Exploratory data analysis

4.1 Read the dataset

[5]:
```
train_data = pd.read_csv("../data/train.csv",index_col=False)
test_data = pd.read_csv("../data/test.csv",index_col=False)
test_data_labels = pd.read_csv("../data/test_labels.csv",index_col=False)
test_data_labels.head()
print(type(test_data))
```

```
<class 'pandas.core.frame.DataFrame'>
```

4.2 Preprocess the testing dataset

- If we look at the test dataset it has got three categories -1,0 and 1.
- As per the guidelines from the kaggle it is stated that column value with -1 is not taken for grading.
- So we need to remove that column since we actually don't know to which categories(toxic or non toxic) the data belongs to.
- By removing the appropriate rows we can compare the performance for prediction and ground truth value.
- Since -1 value is not consistent for different columns of toxicity we need to make local copy and remove appropriate rows.

[6]:
```
test_data_toxic = test_data[['comment_text']].copy()
test_data_labels_toxic = test_data_labels['toxic'].copy().to_frame('toxic')
x = list(np.where(test_data_labels_toxic['toxic'] == -1)[0])
test_data_labels_toxic = test_data_labels_toxic.drop(test_data_labels_toxic.
 ↪index[x])
test_data_toxic = test_data_toxic.drop(test_data_toxic.index[x])
print(len(test_data_toxic),len(test_data_labels_toxic))
```

```
63978 63978
```

4.3 Dataset size

[7]:
```
print("Train data size:", train_data.shape)
print("Test data size: ", test_data.shape)
```

```
Train data size: (159571, 8)
Test data size:  (153164, 2)
```

4.4 Charctersitics of data

4.4.1 Train data

[ ]:
```
train_data.describe()
```

[ ]:
```
               toxic   severe_toxic  …         insult  identity_hate
count  159571.000000  159571.000000  …  159571.000000  159571.000000
mean        0.095844       0.009996  …       0.049364       0.008805
std         0.294379       0.099477  …       0.216627       0.093420
min         0.000000       0.000000  …       0.000000       0.000000
25%         0.000000       0.000000  …       0.000000       0.000000
50%         0.000000       0.000000  …       0.000000       0.000000
75%         0.000000       0.000000  …       0.000000       0.000000
max         1.000000       1.000000  …       1.000000       1.000000

[8 rows x 6 columns]
```

[ ]:
```
train_data.groupby('toxic').describe()
```

[ ]:
```
      severe_toxic                                  … identity_hate
             count      mean       std  min  25%  …           min  25%  50%
75%  max
toxic                                               …
0         144277.0  0.000000  0.000000  0.0  0.0  …           0.0  0.0  0.0
0.0  1.0
1          15294.0  0.104289  0.305645  0.0  0.0  …           0.0  0.0  0.0
0.0  1.0

[2 rows x 40 columns]
```

4.4.2 Test data

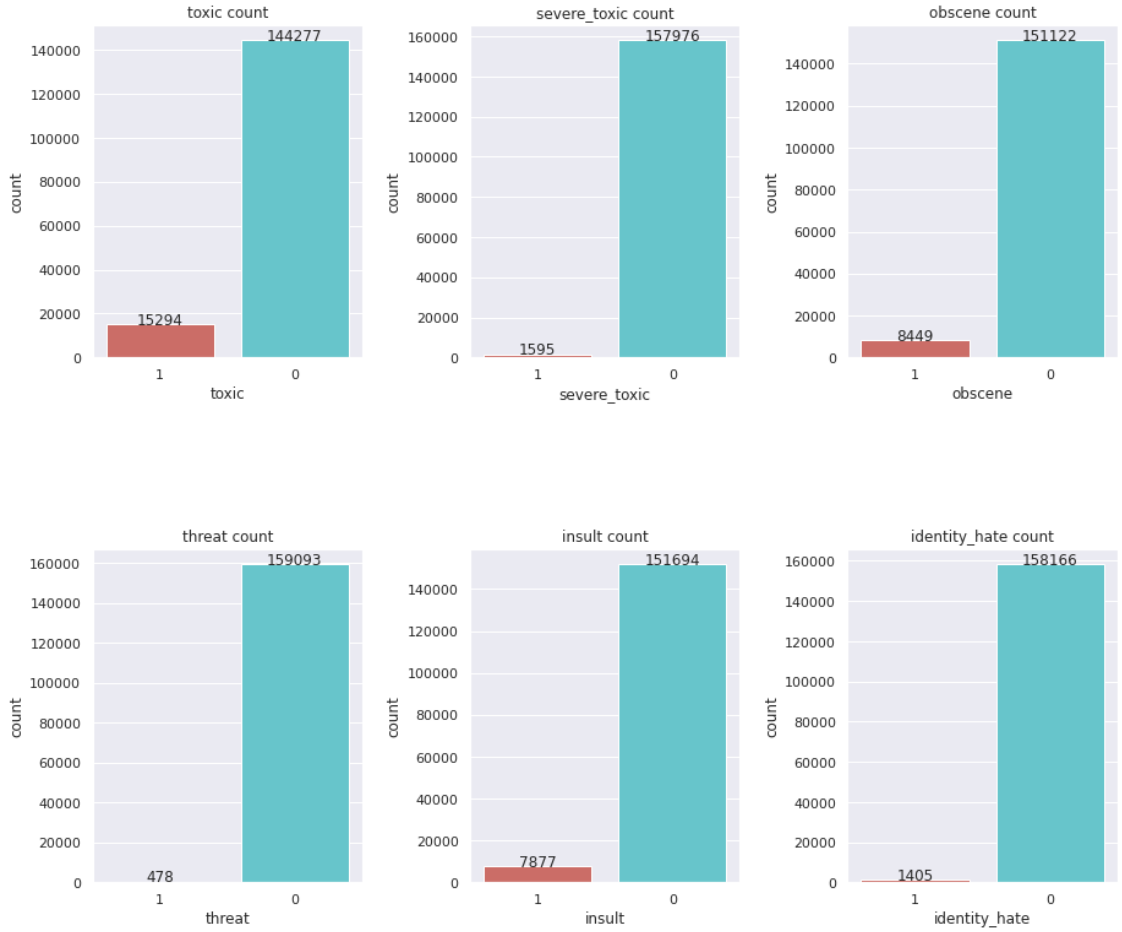
[ ]:
```
test_data.describe()
```

[ ]:
```
                     id                                       comment_text
count              153164                                             153164
unique             153164                                             153164
top     e8696ec7f26006e5  == Ham's wife == \n\n Why not bring it up on t…
freq                    1                                                1
```

[ ]:

4.5 Visualize toxic comments counts

[ ]:
```
sns.set(style="darkgrid")

f, axes = plt.subplots(1, 3, figsize=(15, 5), sharex=True)

graph_1 = sns.countplot(x = 'toxic', data = train_data, palette = 'hls',␣
↪ax=axes[0], order = [1, 0])
graph_1.set_title('toxic count')

graph_2 = sns.countplot(x = 'severe_toxic', data = train_data, palette = 'hls',␣
↪ax=axes[1], order = [1, 0])
graph_2.set_title('severe_toxic count')

graph_3 = sns.countplot(x = 'obscene', data = train_data, palette = 'hls',␣
↪ax=axes[2], order = [1, 0])
graph_3.set_title('obscene count')

f.subplots_adjust(wspace=0.4)

f, axes = plt.subplots(1, 3, figsize=(15, 5), sharex=True)

graph_4 = sns.countplot(x = 'threat', data = train_data, palette = 'hls',␣
↪ax=axes[0], order = [1, 0])
graph_4.set_title('threat count')

graph_5 = sns.countplot(x = 'insult', data = train_data, palette = 'hls',␣
↪ax=axes[1], order = [1, 0])
graph_5.set_title('insult count')

graph_6 = sns.countplot(x = 'identity_hate', data = train_data, palette =␣
↪'hls', ax=axes[2], order = [1, 0])
graph_6.set_title('identity_hate count')

f.subplots_adjust(wspace=0.4)


for graph in [graph_1, graph_2, graph_3, graph_4, graph_5, graph_6]:
    for p in graph.patches:
        height = p.get_height()
        graph.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```



[8]:
```
text = train_data['comment_text'].tolist()
tags = train_data['toxic'].tolist()

# Generate strings for wordcloud
toxic_text = " ".join([words for words, tag in zip(text, tags) if tag == 1])
notoxic_text = " ".join([words for words, tag in zip(text, tags) if tag == 0])

toxic_wordcloud = WordCloud(height=500, width=500, background_color='white',␣
 ↪collocations=False).generate(toxic_text)
notoxic_wordcloud = WordCloud(height=500, width=500, background_color='white',␣
 ↪collocations=False).generate(notoxic_text)

# Save wordclouds
toxic_wordcloud.to_file("../images/Final/wordcloud/unigram_toxic_wordcloud.png")
notoxic_wordcloud.to_file("../images/Final/wordcloud/unigram_notoxic_wordcloud.
 ↪png")
```

[8]:
```
<wordcloud.wordcloud.WordCloud at 0x7ff7f754e780>
```

4.5.1 Words used in toxic sentences

4.5.2 Words used in no toxic sentences

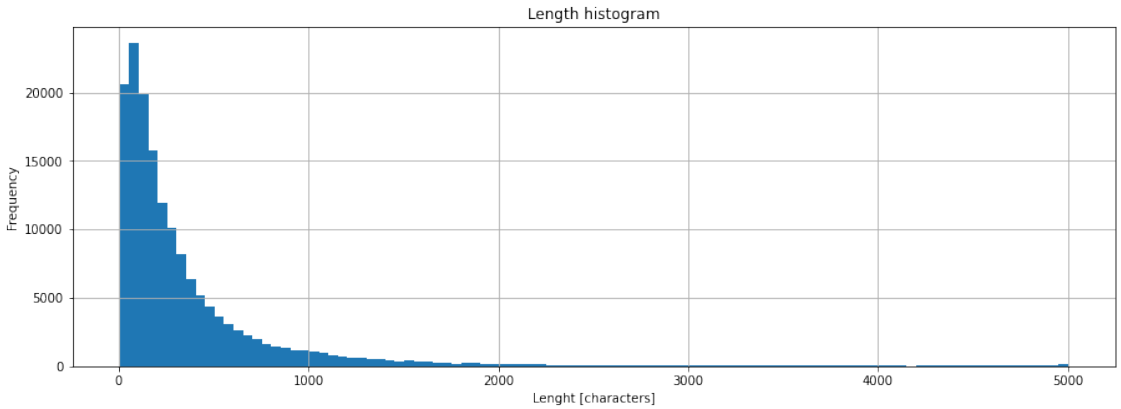
4.6 Comments length based on number of words used

[ ]:
```
train_data["length"] = train_data["comment_text"].apply(len)
train_data.head(n=10)
```

[ ]:
```
                 id  … length
0  0000997932d777bf  …    264
1  000103f0d9cfb60f  …    112
2  000113f07ec002fd  …    233
3  0001b41b1c6bb37e  …    622
4  0001d958c54c6e35  …     67
5  00025465d4725e87  …     65
6  0002bcb3da6cb337  …     44
7  00031b1e95af7921  …    115
8  00037261f536c51d  …    472
9  00040093b2687caa  …     70

[10 rows x 9 columns]
```

4.6.1 Visualize the counts using histogram

[ ]:
```
f = plt.figure(figsize=(15, 5))

plt.hist(train_data["length"], 100)
plt.title('Length histogram')
plt.xlabel('Lenght [characters]')
plt.ylabel('Frequency')

plt.grid()
plt.show()
```



4.6.2 Charctersitics of length of data

[ ]:
```
train_data.length.describe()
```

[ ]:
```
count    159571.000000
mean        394.073221
std         590.720282
min           6.000000
25%          96.000000
50%         205.000000
75%         435.000000
max        5000.000000
Name: length, dtype: float64
```

5 Performance comparison of algorithms w.r.t diferrent features

5.1 Extract the class labels

[ ]:
```
X_train = train_data['comment_text'].values
# X_train = X_train[0:300]
Y_train = train_data['toxic'].values
# Y_train = Y_train[0:300]
print(type(X_train))
X_test = test_data_toxic['comment_text'].values
Y_test = test_data_labels_toxic['toxic'].values
```

```
<class 'numpy.ndarray'>
```

5.2 Train Naive Bayes algorithm iteratively for different sets of features

[ ]:
```
def naive_bayes():
  nb_clf = Pipeline([
      ('vect', CountVectorizer()),
      ('tfidf', TfidfTransformer()),
      ('clf', MultinomialNB())
  ])

  param_grid = {'vect__ngram_range': ((1, 1), (1, 2), (1, 3)),
                'tfidf__norm': ('l1', 'l2')
               }

  kfold = KFold(n_splits=10, random_state=23)
  gs_nb_clf = GridSearchCV(nb_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')
```

```
  start = time.time()
  gs_nb_clf = gs_nb_clf.fit(X_train, Y_train)

  end = time.time()
  print("Time taken:", end - start)
  naive_bayes = "Execution of gridsearch for naive bayes is complete. Execution␣
↪time: "+str(end - start)+" sec"
  with open("../model/Grid_search_modes/gs_nb_features.pkl", 'wb') as␣
↪gs_nb_clf_file:
      pickle.dump(gs_nb_clf, gs_nb_clf_file, pickle.HIGHEST_PROTOCOL)
  return naive_bayes

var = None
naive_bayes = None
try:
  naive_bayes = naive_bayes()
except:
  var = traceback.format_exc()

if var == None:
  telegram_bot_sendtext(naive_bayes)
else:
  telegram_bot_sendtext(var)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning
/usr/local/lib/python3.6/dist-
packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning

Time taken: 2531.117730617523
```

5.3 Train Logistic regression algorithm iteratively for different sets of features

[ ]:
```
lr_clf = Pipeline([
   ( 'vect', CountVectorizer()),
    ('tfidf',TfidfTransformer()),
    ('classifier',LogisticRegression()),
])

param_grid = {'vect__ngram_range': ((1, 1), (1, 2), (1, 3)),
              'tfidf__norm': ('l1', 'l2')
```

```
            }

kfold = KFold(n_splits=10, random_state=23)
gs_lr_clf = GridSearchCV(lr_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')

start = time.time()
gs_lr_clf = gs_lr_clf.fit(X_train, Y_train)
end = time.time()
print("Time taken:", end - start)

logistic_regression = "Execution of gridsearch for logistic_regression is␣
 ↪complete. Execution time: "+str(end - start)+" sec"
telegram_bot_sendtext(logistic_regression)

with open("../model/Grid_search_modes/gs_lr_features.pkl", 'wb') as␣
 ↪gs_lr_clf_file:
    pickle.dump(gs_lr_clf, gs_lr_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning
/usr/local/lib/python3.6/dist-
packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Time taken: 4871.411665439606
```

5.4 Train Support Vector Machine iteratively for different sets of features

[ ]:
```
svm_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
```

```
    ('clf', SGDClassifier(tol=None, max_iter=100))
])

param_grid = {'vect__ngram_range': ((1, 1), (1, 2), (1, 3)),
              'tfidf__norm': ('l1', 'l2')
             }

kfold = KFold(n_splits=10, random_state=23)
gs_svm_clf = GridSearchCV(svm_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')

start = time.time()
gs_svm_clf = gs_svm_clf.fit(X_train, Y_train)
# gs_svm_clf = svm_clf.fit(X_train, Y_train)
end = time.time()
print("Time taken:", end - start)

svm = "Execution of gridsearch for support vector machine is complete.␣
 ↪Execution time: "+str(end - start)+" sec"
telegram_bot_sendtext(svm)

with open("../model/Grid_search_modes/gs_svm_features.pkl", 'wb') as␣
 ↪gs_svm_clf_file:
    pickle.dump(gs_svm_clf, gs_svm_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning
```

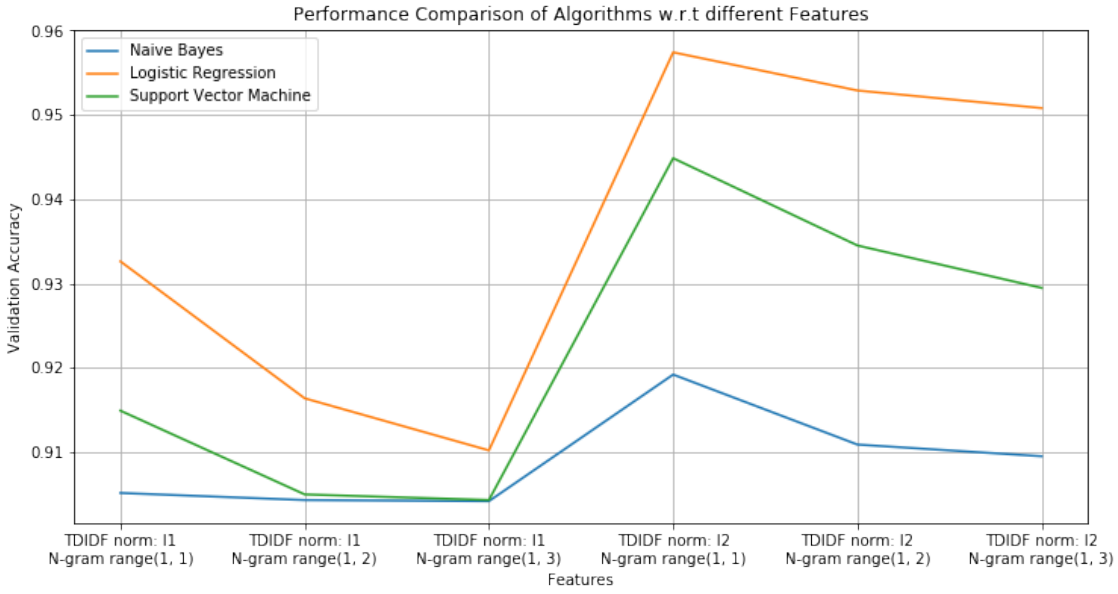
```
Time taken: 3118.977990627289
```

5.5 Plot the graph

[11]:
```
def draw_performance_comparison(x, y):
    fig = plt.figure(figsize=(12,6))
    ax1 = fig.add_subplot(111)

    ax1.plot(x, y[0], label="Naive Bayes")
    ax1.plot(x, y[1], label="Logistic Regression")
    ax1.plot(x, y[2], label="Support Vector Machine")

    plt.xlabel('Features')
    plt.ylabel('Validation Accuracy')
    plt.title('Performance Comparison of Algorithms w.r.t different Features')
    ax1.legend(loc=2)
```

```
    plt.grid(True)
    plt.savefig("../images/Final/Performance_graph/performance.png")
    plt.savefig("../images/Final/Performance_graph/performance.pdf")
    plt.show()
```

[12]:
```
with open("../model/Grid_search_modes/gs_nb_features.pkl", 'rb') as␣
↪gs_nb_clf_file:
    gs_nb_clf = pickle.load(gs_nb_clf_file)
with open("../model/Grid_search_modes/gs_lr_features.pkl", 'rb') as␣
↪gs_lr_clf_file:
    gs_lr_clf = pickle.load(gs_lr_clf_file)
with open("../model/Grid_search_modes/gs_svm_features.pkl", 'rb') as␣
↪gs_svm_clf_file:
    gs_svm_clf = pickle.load(gs_svm_clf_file)

x = list()
params = gs_nb_clf.cv_results_['params']
for param in params:
    norm = "TDIDF norm: " + str(param['tfidf__norm'])
    ngram_range = "N-gram range" + str(param['vect__ngram_range'])
    x.append(norm + "\n" + ngram_range)

nb_means = gs_nb_clf.cv_results_['mean_test_score']
lr_means = gs_lr_clf.cv_results_['mean_test_score']
svm_means = gs_svm_clf.cv_results_['mean_test_score']
y = [nb_means, lr_means, svm_means]
performance = [["Naive_bayes:",y[0]],["Logistic regression:",y[1]],["Support␣
↪vector machine:",y[2]]]
telegram_bot_sendtext(str(performance))
draw_performance_comparison(x,y)
```



6 Hyperparamter Tuning

6.0.1 Tune hyperparameters of Naive Bayes

[ ]:
```
estimators_nb = [
    ('vect', CountVectorizer(ngram_range=(1,1))),
    ('tfidf', TfidfTransformer(norm='l2')),
    ('clf', MultinomialNB())
]
nb_clf = Pipeline(estimators_nb)

param_grid = dict(
    clf__alpha = [0.001, 0.01, 0.1, 1, 10],
)
kfold = KFold(n_splits=10, random_state=23)
gs_nb_clf = GridSearchCV(nb_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')

start = time.time()
gs_nb_clf = gs_nb_clf.fit(X_train, Y_train)
end = time.time()
print("Time taken:", end - start)

naive_bayes_tuned = "Execution of hyperparameter tuning for naive bayes is␣
 ↪complete. Execution time: "+str(end - start)+" sec"
telegram_bot_sendtext(naive_bayes_tuned)
```

```
with open("../model/Grid_search_modes/gs_nb_tuned.pkl", 'wb') as gs_nb_clf_file:
    pickle.dump(gs_nb_clf, gs_nb_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning
```

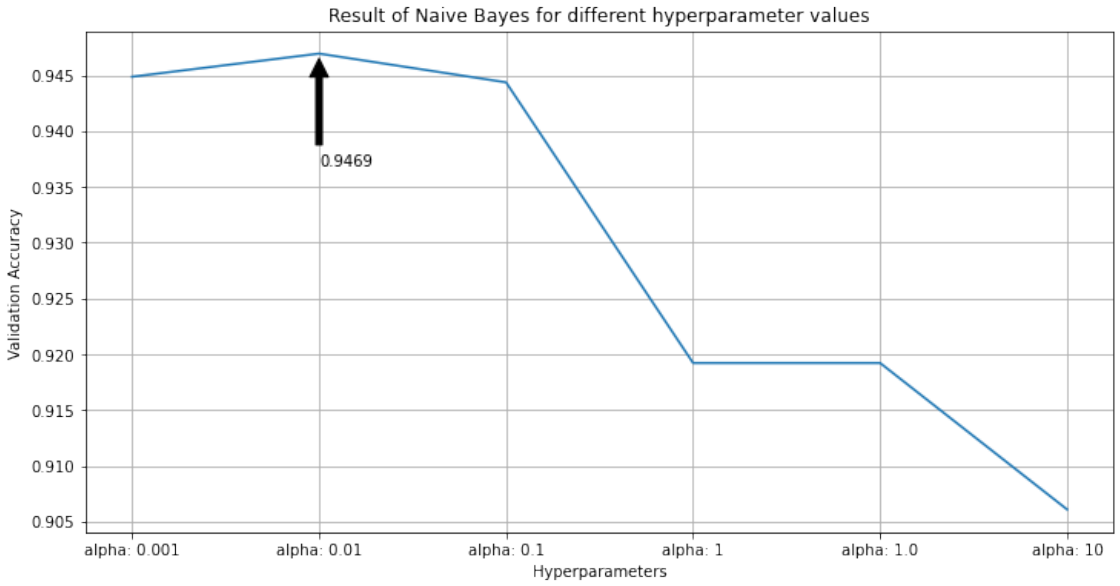
```
Time taken: 415.47970819473267
```

6.1 Performance

[ ]:
```
def draw_hp_performance_nb(x, y):
    fig = plt.figure(figsize=(12,6))
    ax1 = fig.add_subplot(111)

    ax1.plot(x, y, label="Naive Bayes")
    ax1.annotate(str(round(max(y), 4)), xy=(x[y.index(max(y))], y[y.
 ↪index(max(y))]),
                     xytext=(x[y.index(max(y))], max(y) - 0.01),
                     arrowprops=dict(facecolor='black', shrink=0.05))
    plt.xlabel('Hyperparameters')
    plt.ylabel('Validation Accuracy')
    plt.title('Result of Naive Bayes for different hyperparameter values')
    plt.grid(True)
    plt.savefig("../images/Final/naive_bayes_hyperparameter_tuning/nb.
 ↪png")
    plt.savefig("../images/Final/naive_bayes_hyperparameter_tuning/nb.pdf")
    plt.show()
```

[ ]:
```
# NB Results
with open("../model/Grid_search_modes/gs_nb_tuned.pkl", 'rb') as gs_nb_clf_file:
    gs_nb_clf = pickle.load(gs_nb_clf_file)

with open("../model/Grid_search_modes/gs_nb_features.pkl", 'rb') as␣
 ↪gs_nb_features_file:
    gs_nb_features = pickle.load(gs_nb_features_file)

x = list()
params = gs_nb_clf.cv_results_['params']
for param in params:
    alpha = "alpha: " + str(param['clf__alpha'])
    x.append(alpha)

nb_means = gs_nb_clf.cv_results_['mean_test_score']
y = nb_means
```

```
draw_hp_performance_nb(x, y)
naive_bayes_tuned = str("Best validation accuracy for naive bayes: %f using %s"␣
 ↪% (max(y), x[y.index(max(y))]))

telegram_bot_sendtext(naive_bayes_tuned)
print("Best: %f using %s" % (max(y), x[y.index(max(y))]))
```




```
Best: 0.946939 using alpha: 0.01
```

6.2 Tune hyperparameters of Logistic regressor

[ ]:
```
estimators_lr = [
    ('vect', CountVectorizer(ngram_range=(1,1))),
    ('tfidf', TfidfTransformer(norm='l2')),
    ('clf', LogisticRegression(class_weight='balanced'))
]
lr_clf = Pipeline(estimators_lr)

param_grid = dict(
    clf__C = [0.01, 0.1, 1, 10, 100],
    clf__solver = ['newton-cg', 'liblinear', 'lbfgs']
)
kfold = KFold(n_splits=10, random_state=23)
gs_lr_clf = GridSearchCV(lr_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')
```

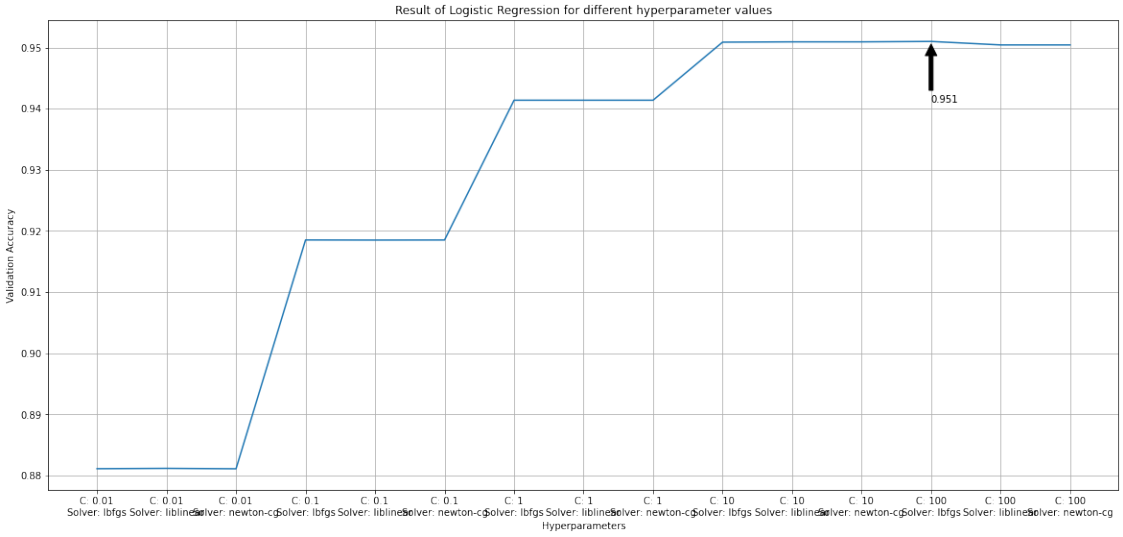
```
start = time.time()
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    gs_lr_clf = gs_lr_clf.fit(X_train, Y_train)
end = time.time()
print("Time taken:", end - start)

lr_tuned = "Execution of hyperparameter tuning for logistic regression is␣
 ↪complete. Execution time: "+str(end - start)+" sec"
telegram_bot_sendtext(lr_tuned)

with open("../model/Grid_search_modes/gs_lr_tuned.pkl", 'wb') as gs_lr_clf_file:
    pickle.dump(gs_lr_clf, gs_lr_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning

Time taken: 2114.2145347595215
```

6.3 Performance

[ ]:
```
def draw_hp_performance_lr(x, y):
    fig = plt.figure(figsize=(20,9))
    ax1 = fig.add_subplot(111)

    x, y = zip(*sorted(zip(x, y)))

    ax1.plot(x, y, label="Logistic Regression")
    ax1.annotate(str(round(max(y), 4)), xy=(x[y.index(max(y))], y[y.
 ↪index(max(y))]),
                    xytext=(x[y.index(max(y))], max(y) - 0.01),
                    arrowprops=dict(facecolor='black', shrink=0.05))
    plt.xlabel('Hyperparameters')
    plt.ylabel('Validation Accuracy')
    plt.title('Result of Logistic Regression for different hyperparameter␣
 ↪values')
    plt.grid(True)
    plt.savefig("../images/Final/logistic_regression_hyperparameter_tuning/lr.
 ↪png")
    plt.savefig("../images/Final/logistic_regression_hyperparameter_tuning/lr.
 ↪pdf")
    plt.show()
```

```
[ ]: with open("../model/Grid_search_modes/gs_lr_tuned.pkl", 'rb') as gs_lr_clf_file:
         gs_lr_clf = pickle.load(gs_lr_clf_file)

     with open("../model/Grid_search_modes/gs_lr_features.pkl", 'rb') as␣
      ↪gs_lr_features_file:
         gs_lr_features = pickle.load(gs_lr_features_file)

     x = list()
     params = gs_lr_clf.cv_results_['params']
     for param in params:
         C = "C: " + str(param['clf__C'])
         solver = "Solver: " + str(param['clf__solver'])
         x.append(C + "\n" + solver)

     lr_means = gs_lr_clf.cv_results_['mean_test_score']
     y = lr_means

     draw_hp_performance_lr(x, y)
     lr_tuned_value = str("Best validation accuracy for logistic regression: %f␣
      ↪using %s" % (max(y), x[y.index(max(y))]))

     telegram_bot_sendtext(lr_tuned_value)
     print("Best: %f using %s" % (max(y), x[y.index(max(y))]))
```




```
Best: 0.951012 using C: 100
Solver: lbfgs
```

6.4 Tune hyperparameters of support vector machine

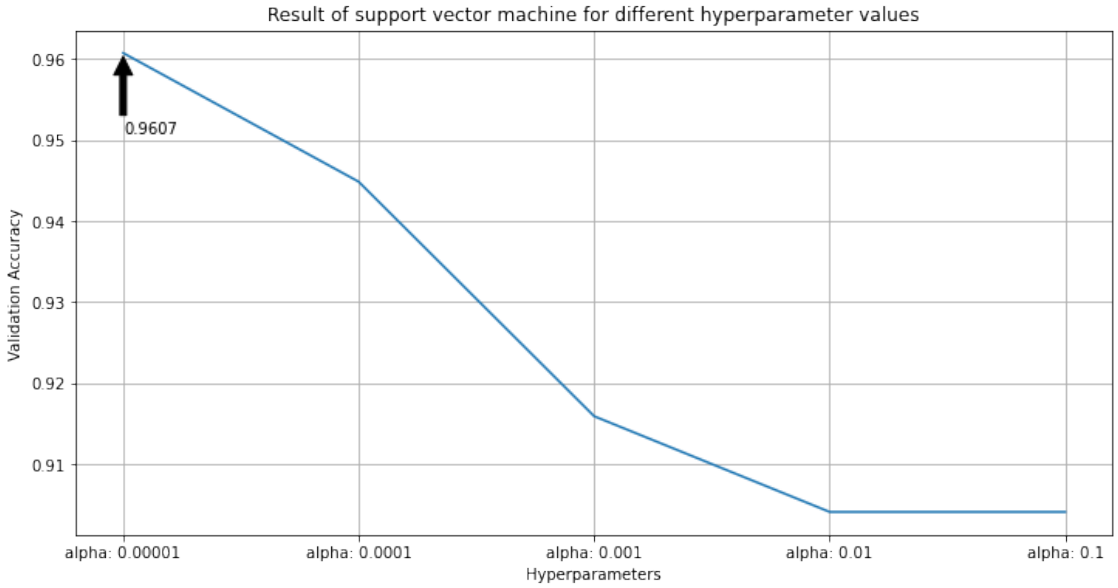
[ ]:
```
estimators_svm = [
    ('vect', CountVectorizer(ngram_range=(1,1))),
    ('tfidf', TfidfTransformer(norm='l2')),
    ('clf', SGDClassifier(tol=None, max_iter=100))
]
svm_clf = Pipeline(estimators_svm)

param_grid = dict(
    clf__alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1],
)
# param_grid = {'clf__learning_rate': [ 0.1, 1, 10, 100],
#               'clf__loss':['log']}

kfold = KFold(n_splits=10, random_state=23)
gs_svm_clf = GridSearchCV(svm_clf, param_grid=param_grid, n_jobs=-1, cv=kfold,␣
 ↪scoring='accuracy')

start = time.time()
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    gs_svm_clf = gs_svm_clf.fit(X_train, Y_train)
end = time.time()
print("Time taken:", end - start)

svm_tuned = "Execution of hyperparameter tuning for svm is complete. Execution␣
 ↪time: "+str(end - start)+" sec"
telegram_bot_sendtext(svm_tuned)

with open("../model/Grid_search_modes/gs_svm_tuned.pkl", 'wb') as␣
 ↪gs_svm_clf_file:
    pickle.dump(gs_svm_clf, gs_svm_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default
(None), or set shuffle=True.
  FutureWarning

Time taken: 613.0611274242401
```

[ ]:
```
def draw_hp_performance_svm(x, y):
    fig = plt.figure(figsize=(12,6))
    ax1 = fig.add_subplot(111)

    x, y = zip(*sorted(zip(x, y)))
```

```
    ax1.plot(x, y, label="Support vector machine")
    ax1.annotate(str(round(max(y), 4)), xy=(x[y.index(max(y))], y[y.
 ↪index(max(y))]),
                     xytext=(x[y.index(max(y))], max(y) - 0.01),
                     arrowprops=dict(facecolor='black', shrink=0.05))
    plt.xlabel('Hyperparameters')
    plt.ylabel('Validation Accuracy')
    plt.title('Result of support vector machine for different hyperparameter␣
 ↪values')
    plt.grid(True)
    plt.savefig("../images/Final/SVM_hyperparameter_tuning/svm.png")
    plt.savefig("../images/Final/SVM_hyperparameter_tuning/svm.pdf")

    plt.show()
```

[ ]:
```
with open("../model/Grid_search_modes/gs_svm_tuned.pkl", 'rb') as␣
 ↪gs_svm_clf_file:
    gs_svm_clf = pickle.load(gs_svm_clf_file)

with open("../model/Grid_search_modes/gs_svm_features.pkl", 'rb') as␣
 ↪gs_svm_features_file:
    gs_svm_features = pickle.load(gs_svm_features_file)

x = list()
params = gs_svm_clf.cv_results_['params']
for param in params:
    alpha = "alpha: " + str(np.format_float_positional(param['clf__alpha'],␣
 ↪trim='-'))
    x.append(alpha)

svm_means = gs_svm_clf.cv_results_['mean_test_score']
y = svm_means

draw_hp_performance_svm(x, y)
svm_tuned_value = str("Best accuracy for svm: %f using %s" % (max(y), x[y.
 ↪index(max(y))]))

telegram_bot_sendtext(svm_tuned_value)
print("Best: %f using %s" % (max(y), x[y.index(max(y))]))
```




```
Best: 0.960713 using alpha: 0.00001
```

6.4.1 It is evident from the above result that support vector machine outperforme naive bayes and logistic regression for alpha = 0.00001 (Learning rate), l2 norm and (1,1) n grams. So 6 different levels of toxicity is trained with svm classifier with above mentioned hyperparameter to achive good result.

6.4.2 SVM classifier for 6 different classes

[7]:
```
class SVM():
    def __init__(self, train_data, test_data, comment_type):
        if(comment_type not in train_data):
            raise Exception(comment_type + ' is not found in the training data')

        self.X_train = train_data['comment_text'].values
        self.Y_train = train_data[comment_type].values
        self.X_test = test_data['comment_text'].values
        self.Y_test = test_data[comment_type].values
        self.comment_type = comment_type


    def train(self):
        estimators_svm = [('vect', CountVectorizer(ngram_range=(1,1))),
                          ('tfidf', TfidfTransformer(norm='l2')),
                          ('clf', SGDClassifier(tol=None,␣
↪max_iter=100,alpha=0.00001))]

        self.svm_clf = Pipeline(estimators_svm)
```

```
        print("start")
        start = time.time()
        with warnings.catch_warnings():
            warnings.filterwarnings("ignore")
            self.svm_clf = self.svm_clf.fit(self.X_train, self.Y_train)
        end = time.time()
        print("Time taken:", end - start)

        svm_tuned_new = str(self.comment_type)+": Training complete. Execution␣
↪time: " + str(end - start) + " sec"
        telegram_bot_sendtext(svm_tuned_new)

        with open("../model/svm_trained_model/svm_" + str(self.comment_type) +␣
↪".pkl", 'wb') as svm_clf_file:
            pickle.dump(self.svm_clf, svm_clf_file, pickle.HIGHEST_PROTOCOL)

    def load_model(self, file):
        self.svm_clf = pickle.load(open(file, 'rb'))

    def performance_meaure(self):
        self.prediction = self.svm_clf.predict(self.X_test)
        print(classification_report(self.prediction, self.Y_test))
        print("-"*20)
        print("confusion matrix")
        print(self.confusion_matrix())

    def confusion_matrix(self):
        data = {'y_Actual': self.Y_test,
            'y_Predicted': self.prediction}

        df = pd.DataFrame(data, columns=['y_Actual','y_Predicted'])
        confusion_matrix = pd.crosstab(df['y_Actual'], df['y_Predicted'],␣
↪rownames=['Actual'], colnames=['Predicted'])
        sns.heatmap(confusion_matrix, annot=True, fmt='d')
        plt.savefig("../images/Final/Confusion_matrix/" + str(self.
↪comment_type) + ".pdf")
        plt.show()
```

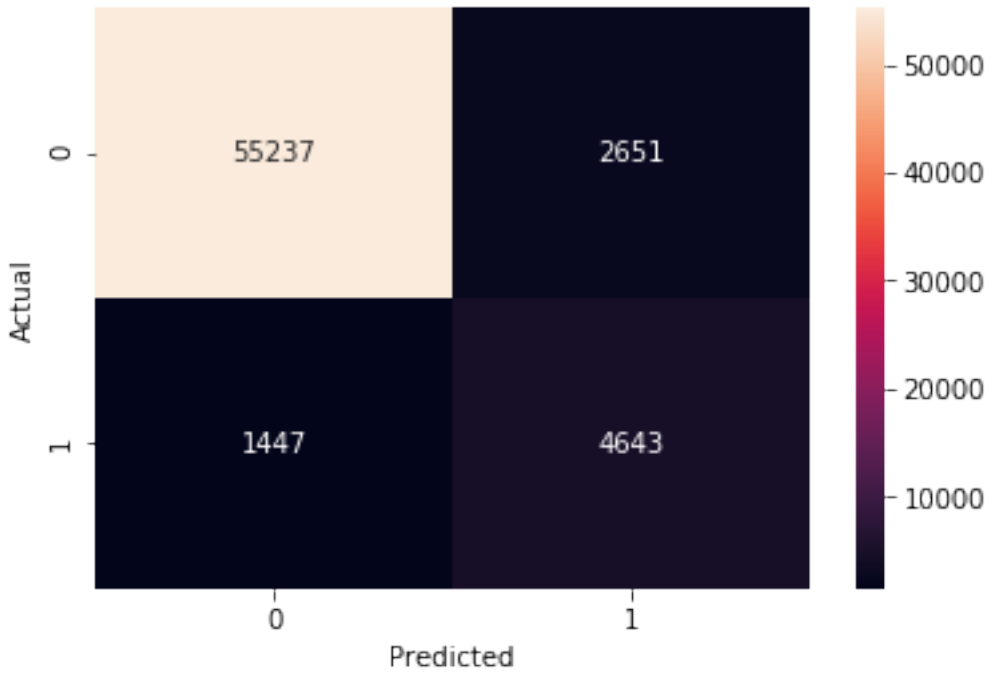
6.4.3 Load data

[8]:
```
train_data = pd.read_csv("../data/train.csv",index_col=False)

test_data_comments = pd.read_csv("../data/test.csv",index_col=False)
test_data_labels = pd.read_csv("../data/test_labels.csv",index_col=False)
```

```
test_data = pd.concat([test_data_comments, test_data_labels.drop(['id'],␣
 ↪axis=1)], axis=1, sort=False)
test_data = test_data[test_data['toxic'] != -1]
```

[9]:
```
svm_toxic = SVM(train_data, test_data, 'toxic')
svm_toxic.train()
svm_toxic.performance_meaure()
```

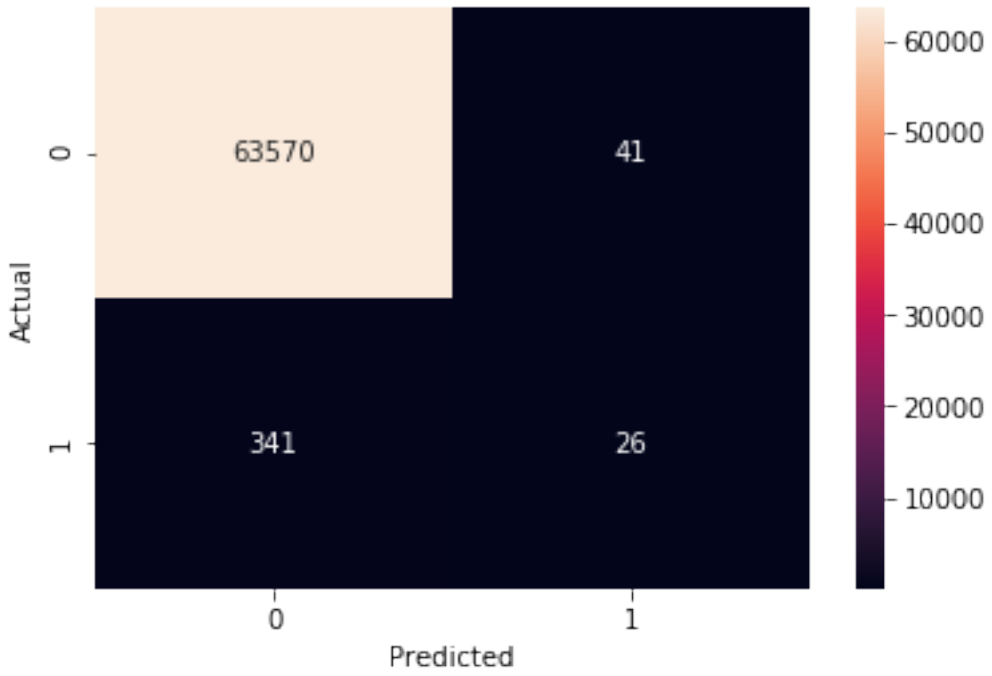
```
start
Time taken: 10.101313591003418
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     56684
           1       0.76      0.64      0.69      7294

    accuracy                           0.94     63978
   macro avg       0.86      0.81      0.83     63978
weighted avg       0.93      0.94      0.93     63978

--------------------
confusion matrix
```




```
None
```

[10]:
```
svm_severe_toxic = SVM(train_data, test_data, 'severe_toxic')
svm_severe_toxic.train()
svm_severe_toxic.performance_meaure()
```

```
start
Time taken: 10.526662111282349
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     63911
           1       0.07      0.39      0.12        67

    accuracy                           0.99     63978
   macro avg       0.54      0.69      0.56     63978
weighted avg       1.00      0.99      1.00     63978

--------------------
confusion matrix
```

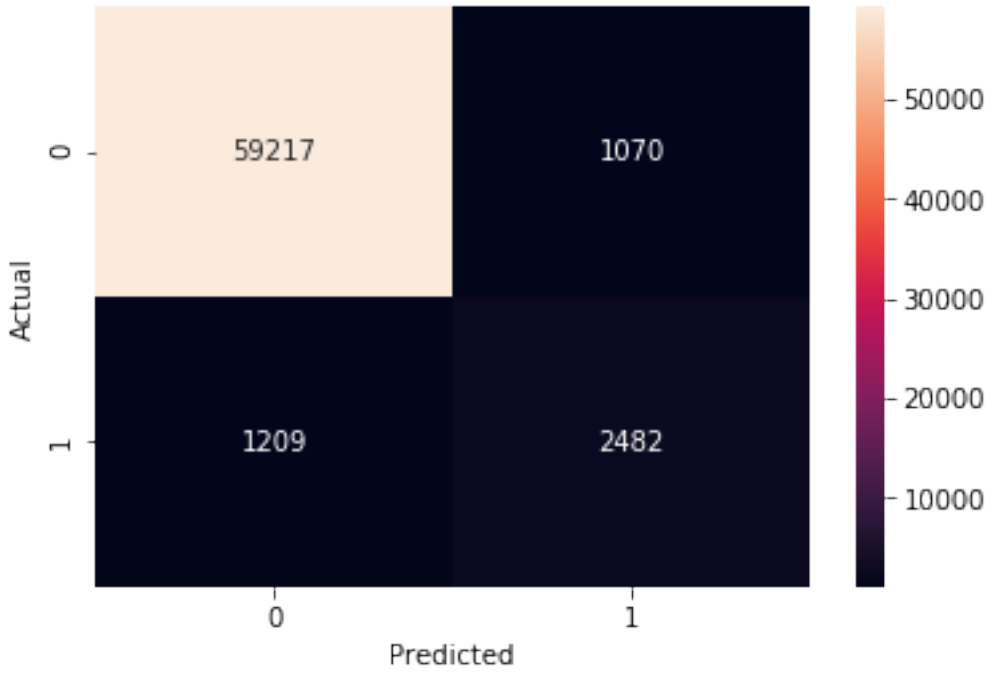



```
None
```

[11]:
```
svm_obscene = SVM(train_data, test_data, 'obscene')
svm_obscene.train()
svm_obscene.performance_meaure()
```

```
start
```

```
Time taken: 12.23163652420044
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     60426
           1       0.67      0.70      0.69      3552

    accuracy                           0.96     63978
   macro avg       0.83      0.84      0.83     63978
weighted avg       0.97      0.96      0.96     63978

--------------------
confusion matrix
```

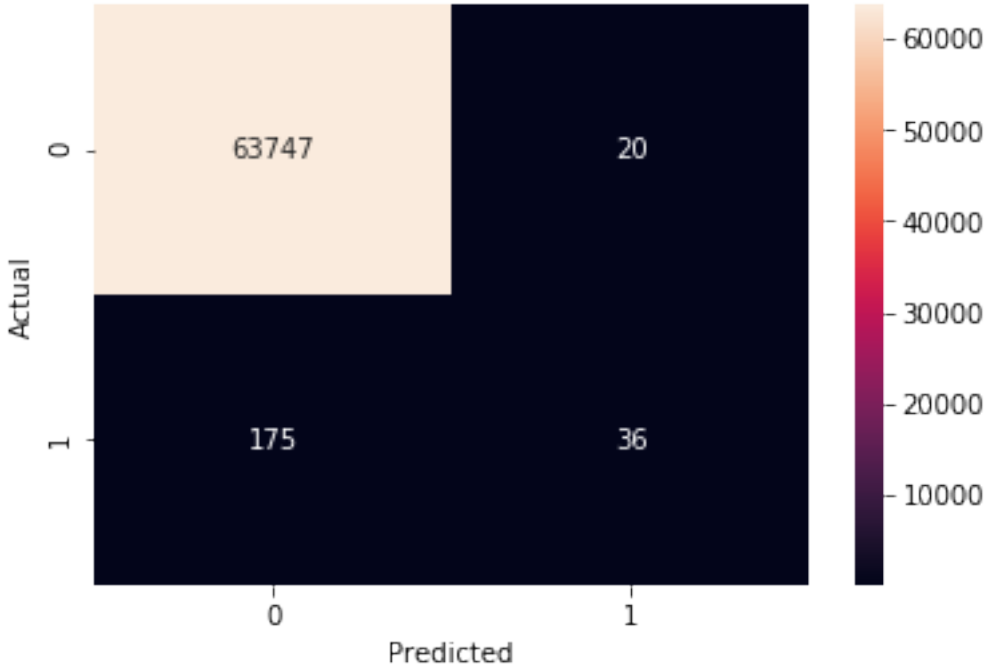



```
None
```

[12]:
```
svm_threat = SVM(train_data, test_data, 'threat')
svm_threat.train()
svm_threat.performance_meaure()
```

```
start
Time taken: 12.167128324508667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     63922
           1       0.17      0.64      0.27        56
```

```
    accuracy                           1.00     63978
   macro avg       0.59      0.82      0.63     63978
weighted avg       1.00      1.00      1.00     63978

--------------------
confusion matrix
```

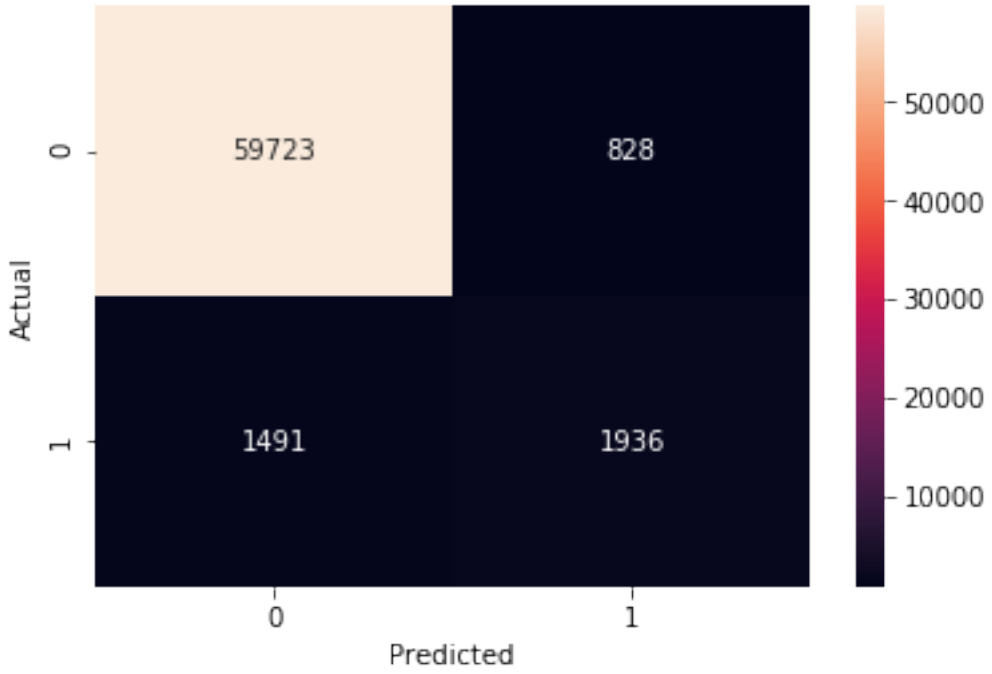



```
None
```

[13]:
```
svm_insult = SVM(train_data, test_data, 'insult')
svm_insult.train()
svm_insult.performance_meaure()
```

```
start
Time taken: 12.20309829711914
              precision    recall  f1-score   support

           0       0.99      0.98      0.98     61214
           1       0.56      0.70      0.63      2764

    accuracy                           0.96     63978
   macro avg       0.78      0.84      0.80     63978
weighted avg       0.97      0.96      0.97     63978
```

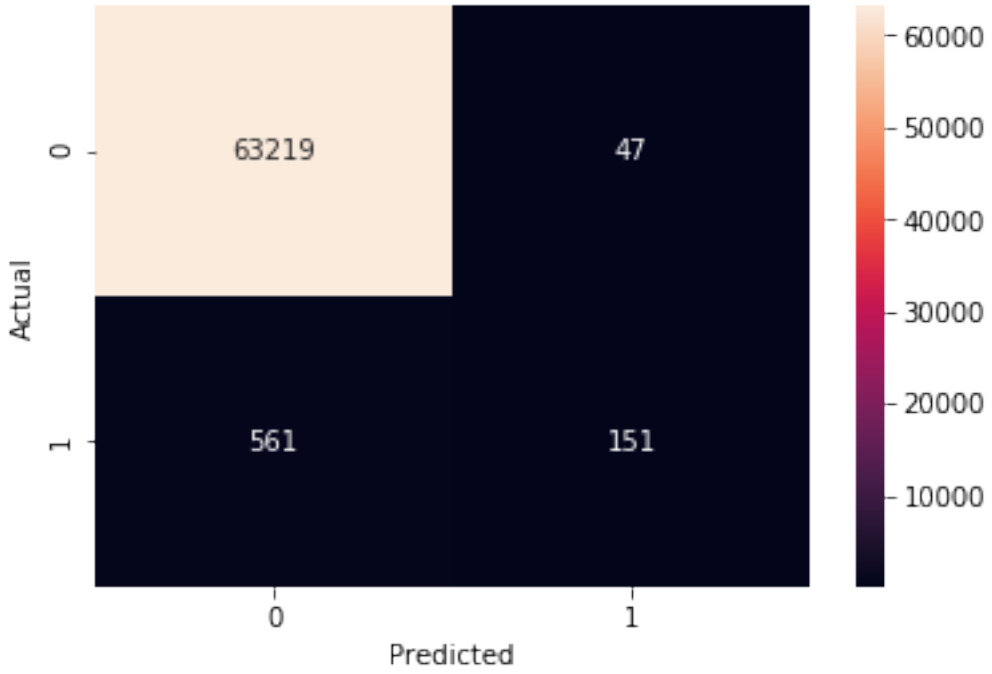
```
--------------------
confusion matrix
```




```
None
```

[14]:
```
svm_identity_hate = SVM(train_data, test_data, 'identity_hate')
svm_identity_hate.train()
svm_identity_hate.performance_meaure()
```

```
start
Time taken: 11.810347080230713
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     63780
           1       0.21      0.76      0.33       198

    accuracy                           0.99     63978
   macro avg       0.61      0.88      0.66     63978
weighted avg       1.00      0.99      0.99     63978

--------------------
confusion matrix
```



None

6.5 Test and play

6.5.1 Load trained models

[34]:
```
class Aggressive_classifier():

    def __init__(self, categories, models):
        if(len(categories) != len(models)):
            raise Exception("The number of categories is not the same as the␣
↪number of models")
        self.categories = categories
        self.models = models

    def predict(self, sentence):
        for category, model in zip(self.categories, self.models):
            prediction = model.predict( [sentence] )

            if(prediction == 0):
                result = "no"
            else:
                result = "yes"

            print(category + " : " + result)
```

[35]:
```
with open("../model/svm_trained_model/svm_toxic.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_toxic = pickle.load(gs_nb_features_file)

with open("../model/svm_trained_model/svm_severe_toxic.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_severe_toxic = pickle.load(gs_nb_features_file)

with open("../model/svm_trained_model/svm_obscene.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_obscene = pickle.load(gs_nb_features_file)

with open("../model/svm_trained_model/svm_threat.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_threat = pickle.load(gs_nb_features_file)

with open("../model/svm_trained_model/svm_insult.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_insult = pickle.load(gs_nb_features_file)

with open("../model/svm_trained_model/svm_identity_hate.pkl", 'rb') as␣
↪gs_nb_features_file:
    svm_identity_hate = pickle.load(gs_nb_features_file)

my_classifier = Aggressive_classifier(["toxic", "severe_toxic", "obscene",␣
↪"threat", "insult", "identity_hate"],
                                      [svm_toxic, svm_severe_toxic, svm_obscene,␣
↪svm_threat, svm_insult, svm_identity_hate])
```

6.5.2 Test models with some tweets

Test with sentences found in internet

[70]:
```
sentence = "Thats not what you mum said when I was fucking her from behind last␣
↪night like the bitch she is."
print(sentence)
my_classifier.predict(sentence)
print()

sentence = "Oooh look at a little cry baby, aren't you just the sorriest lil'␣
↪bitch that ever breathed."
print(sentence)
my_classifier.predict(sentence)
print()

sentence = "Why don't you go eat a nigger you fat bastard."
print(sentence)
```

```
my_classifier.predict(sentence)
print()

sentence = "I have scraped shit off my shoe that looks prettier than you."
print(sentence)
my_classifier.predict(sentence)
print()

sentence = "Why don't you just fuck off and die, dick-bag."
print(sentence)
my_classifier.predict(sentence)
print()

sentence = "You're face looks like you were drop kicked as a kid into a food␣
 ↪blender."
print(sentence)
my_classifier.predict(sentence)
print()

sentence = "Yo mumma so fat that is took 3 moutnainteering teams to find your␣
 ↪ugly mug when you were born."
print(sentence)
my_classifier.predict(sentence)
print()
```

```
Thats not what you mum said when I was fucking her from behind last night like
the bitch she is.
toxic : yes
severe_toxic : no
obscene : yes
threat : no
insult : yes
identity_hate : no

Oooh look at a little cry baby, aren't you just the sorriest lil' bitch that
ever breathed.
toxic : yes
severe_toxic : no
obscene : yes
threat : no
insult : yes
identity_hate : no

Why don't you go eat a nigger you fat bastard.
toxic : yes
severe_toxic : no
obscene : yes
```

```
threat : no
insult : yes
identity_hate : yes

I have scraped shit off my shoe that looks prettier than you.
toxic : yes
severe_toxic : no
obscene : yes
threat : no
insult : no
identity_hate : no

Why don't you just fuck off and die, dick-bag.
toxic : yes
severe_toxic : no
obscene : yes
threat : no
insult : yes
identity_hate : no

You're face looks like you were drop kicked as a kid into a food blender.
toxic : no
severe_toxic : no
obscene : no
threat : no
insult : no
identity_hate : no

Yo mumma so fat that is took 3 moutnainteering teams to find your ugly mug when
you were born.
toxic : yes
severe_toxic : no
obscene : no
threat : no
insult : yes
identity_hate : no
```

Implicit threats are not detected

[64]:
```
implicit_threat = "Any difficulty and we will assume control but, when the␣
 ↪looting starts, the shooting starts. Thank you!"

print(implicit_threat)
my_classifier.predict(implicit_threat)
```

```
Any difficulty and we will assume control but, when the looting starts, the
shooting starts. Thank you!
```

```
toxic : no
severe_toxic : no
obscene : no
threat : no
insult : no
identity_hate : no
```

Verb conjugation not detected

[65]:
```
verb_infinity = "I kill people"
print(verb_infinity)
my_classifier.predict(verb_infinity)

print()

verb_gerundio = "I am killing people"
print(verb_gerundio)
my_classifier.predict(verb_gerundio)
```

```
I kill people
toxic : yes
severe_toxic : no
obscene : no
threat : yes
insult : no
identity_hate : no

I am killing people
toxic : no
severe_toxic : no
obscene : no
threat : no
insult : no
identity_hate : no
```

Reference - https://www.kaggle.com/c/jigsaw-multilingual-toxic-comment-classification/data

[ ]: