Jigsaw_Analysis

August 22, 2019

[2]:
```
%load_ext autoreload
%autoreload 2
```

[3]:
```
import time
import tensorflow as tf
import pandas as pd
import gensim


import p8_util
import p8_util_config
import p9_util
```

```
Using TensorFlow backend.
```

1 1. Data loading

[5]:
```
import pandas as pd
df_train = pd.read_csv('./data/train.csv.zip', compression='zip', header=0,\
     sep=',', quotechar='"')
print(df_train.shape)
```

```
(1804874, 45)
```

[93]:
```
df_train.columns
```

[93]:
```
Index(['id', 'target', 'comment_text', 'severe_toxicity', 'obscene',
       'identity_attack', 'insult', 'threat', 'asian', 'atheist', 'bisexual',
       'black', 'buddhist', 'christian', 'female', 'heterosexual', 'hindu',
       'homosexual_gay_or_lesbian', 'intellectual_or_learning_disability',
       'jewish', 'latino', 'male', 'muslim', 'other_disability',
       'other_gender', 'other_race_or_ethnicity', 'other_religion',
       'other_sexual_orientation', 'physical_disability',
       'psychiatric_or_mental_illness', 'transgender', 'white', 'created_date',
       'publication_id', 'parent_id', 'article_id', 'rating', 'funny', 'wow',
       'sad', 'likes', 'disagree', 'sexual_explicit',
```

```
 'identity_annotator_count', 'toxicity_annotator_count'],
dtype='object')
```

2 2. Data pre-processing

2.1 2.1. Check NAN rate into columns

[236]:
```
ser_train_isna = df_train.isna().sum()
ser_train_isna
```

[236]:
```
id                                         0
target                                     0
comment_text                               0
severe_toxicity                            0
obscene                                    0
identity_attack                            0
insult                                     0
threat                                     0
asian                                1399744
atheist                              1399744
bisexual                             1399744
black                                1399744
buddhist                             1399744
christian                            1399744
female                               1399744
heterosexual                         1399744
hindu                                1399744
homosexual_gay_or_lesbian            1399744
intellectual_or_learning_disability  1399744
jewish                               1399744
latino                               1399744
male                                 1399744
muslim                               1399744
other_disability                     1399744
other_gender                         1399744
other_race_or_ethnicity              1399744
other_religion                       1399744
other_sexual_orientation             1399744
physical_disability                  1399744
psychiatric_or_mental_illness        1399744
transgender                          1399744
white                                1399744
created_date                               0
publication_id                             0
parent_id                             778646
article_id                                 0
rating                                     0
```

```
funny                                    0
wow                                      0
sad                                      0
likes                                    0
disagree                                 0
sexual_explicit                          0
identity_annotator_count                 0
toxicity_annotator_count                 0
dtype: int64
```

2.1.1 Data description

[ ]:
```

```

[ ]:
```

```

2.2 2.2. Filter features without Nan values

[95]:
```
list_colname = [colname for colname, value in ser_train_isna.items() if value␣
 ↪== 0]
print("Number of valid columns= {}".format(len(list_colname)))
```

```
Number of valid columns= 20
```

2.2.1 Columns are filtered from dataframe

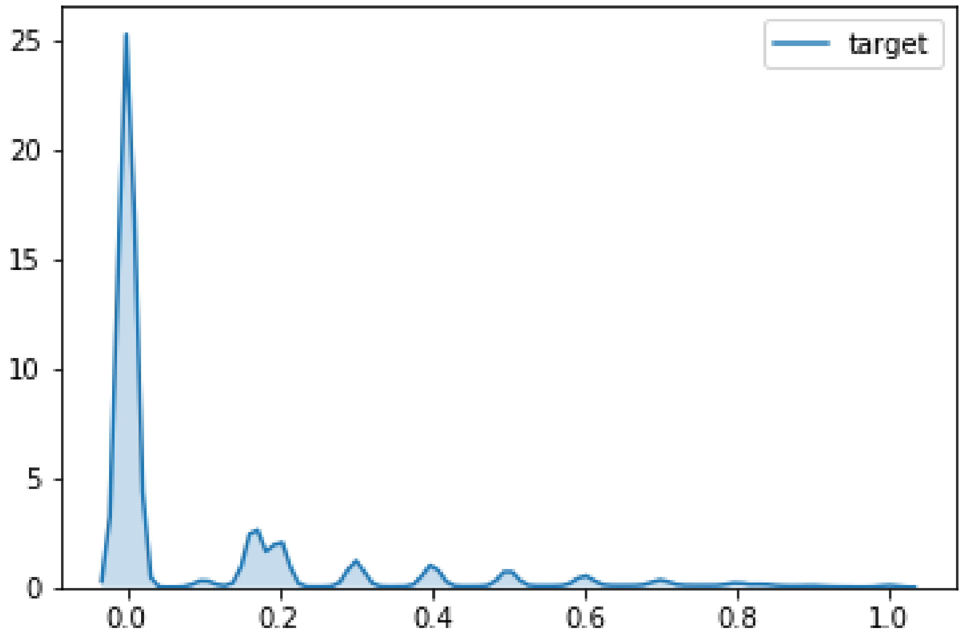
Columns with Nan values are droped : this leads to having 20 features.

[96]:
```
df_train_filtered = df_train[list_colname]

df_train_filtered.shape, df_train.shape
```

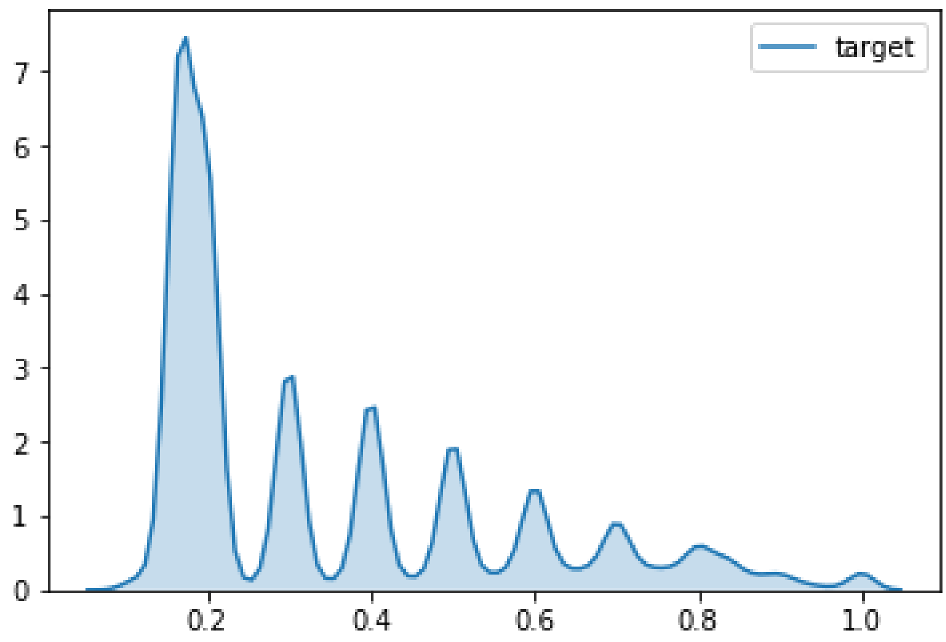
[96]:
```
((1804874, 20), (1804874, 45))
```

2.3 2.3. Target (toxicity) distribution

[126]:
```
import seaborn as sns
for col in ['target']:
    sns.kdeplot(df_train_filtered[col], shade=True)
```



[189]:
```
_=sns.kdeplot(ser_text_10pcent, shade=True)
```



[226]:
```
import p9_util
```

```
total = df_train_filtered.shape[0]
dict_percent = dict()
verbose = True
for val in range(1,10,1) :
    threshold = val/10
    percent = p9_util.print_col_stat(df_train_filtered, col,threshold,␣
 ↪verbose=verbose)
    if verbose :
        print()
    dict_percent[val] = percent
```

```
Number of texts where toxicity > 0.1 : 524054
Percentage of texts where toxicity > 0.1 : 0.290355

Number of texts where toxicity > 0.2 : 270194
Percentage of texts where toxicity > 0.2 : 0.149702

Number of texts where toxicity > 0.3 : 206991
Percentage of texts where toxicity > 0.3 : 0.114684

Number of texts where toxicity > 0.4 : 151463
Percentage of texts where toxicity > 0.4 : 0.083919

Number of texts where toxicity > 0.5 : 106438
Percentage of texts where toxicity > 0.5 : 0.058973

Number of texts where toxicity > 0.6 : 72235
Percentage of texts where toxicity > 0.6 : 0.040022

Number of texts where toxicity > 0.7 : 45451
Percentage of texts where toxicity > 0.7 : 0.025182

Number of texts where toxicity > 0.8 : 23802
Percentage of texts where toxicity > 0.8 : 0.013188

Number of texts where toxicity > 0.9 : 7517
Percentage of texts where toxicity > 0.9 : 0.004165
```

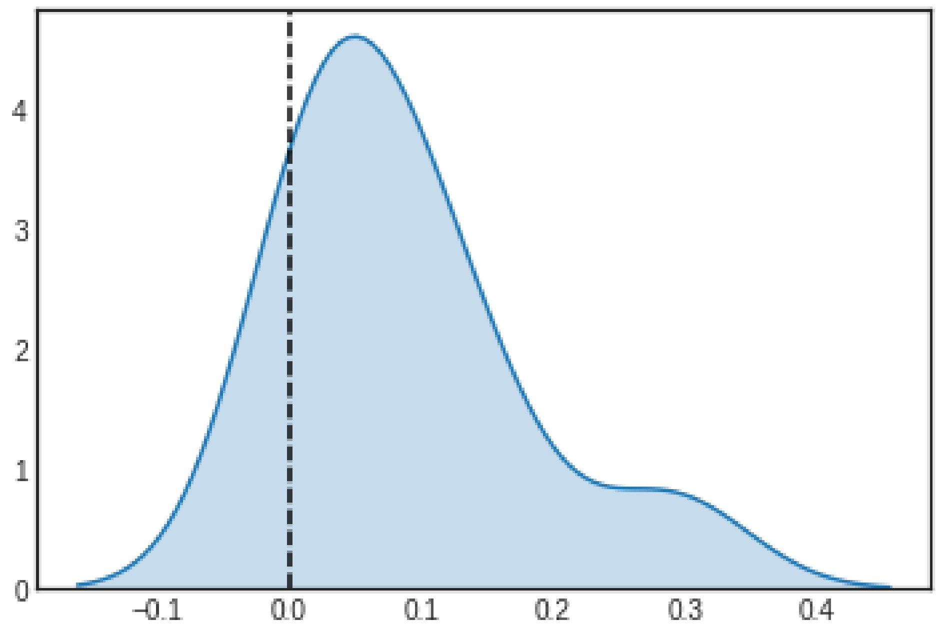
[227]:

```
pd.Series(dict_percent).min()
```

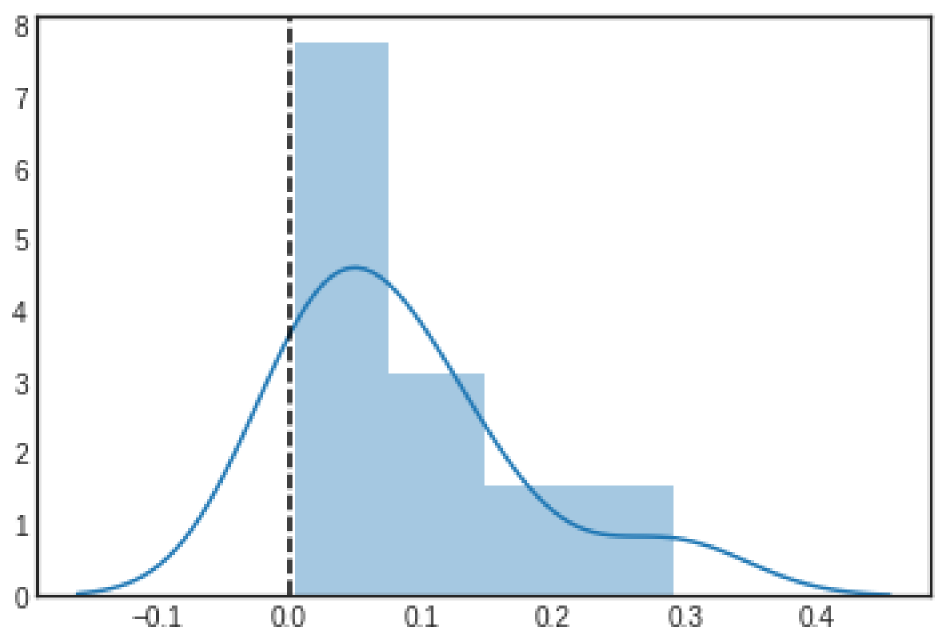
[227]: 0.004164833667059307

[228]:

```
import pandas as pd

_=sns.kdeplot(pd.Series(dict_percent), shade=True)
plt.axvline(0, color="k", linestyle="--");
```



[229]:
```
sns.distplot(pd.Series(dict_percent), kde=True);
plt.axvline(0, color="k", linestyle="--");
```



2.4 2.4. Extract a test dataset from train file

- Dataset with excluded Nan values results in df_train_filtered dataframe.
- Features are studied over a fraction of the train dataset (1%) resulting in df_train_sample.
- A test dataset is extracted from df_train_sample using indexes excluded from df_train_filtered. Result is df_test_filtered dataframe.

[237]:
```
df_train_sample = df_train_filtered.sample(frac=0.01,random_state=0)
print(df_train_sample.shape)
```

```
(18049, 20)
```

```
Get test dataset from indexes into df_train_filtered that do not belongs to train dataset
```

[238]:
```
df_test_filtered = df_train_filtered.query("index!="+str(list(df_train_sample.
 ↪index)))
```

```
Check consistency after building test dataset and train dataset operations from df_train_filte
It is expected to have equality : Size of train dataset + size of test dataset = size of sample
```

[239]:
```
df_train_filtered.shape[0],df_test_filtered.shape[0] + df_train_sample.shape[0]
```

[239]:
```
(1804874, 1804874)
```

[240]:
```
print(df_train_sample.shape)
print(df_test_filtered.shape)
```

```
(18049, 20)
(1786825, 20)
```

- df_train_filtered holds data with excluded Nan features values.
- 1% of df_train_filtered is used for analysis.
- 30% from df_test_filtered is used leading to df_test_sample.

[241]:
```
# 1% of df_train_filtered is used for analysis.
frac_train = df_train_filtered.shape[0]//100
print("\nNumber rows for analysis train dataset= {}".format(frac_train))

# 30% issued from 1% of df_train_filtered is used for test part analysis.
frac_test = frac_train//3
print("Number rows for analysis test part of dataset= {}".format(frac_test))
```

```

Number rows for analysis train dataset= 18048
Number rows for analysis test part of dataset= 6016
```

[242]:
```
df_test_sample = df_test_filtered.sample(n=frac_test,random_state=0)
print(df_test_sample.shape)
```

```
(6016, 20)
```

3 3. Toxicity contributors analysis

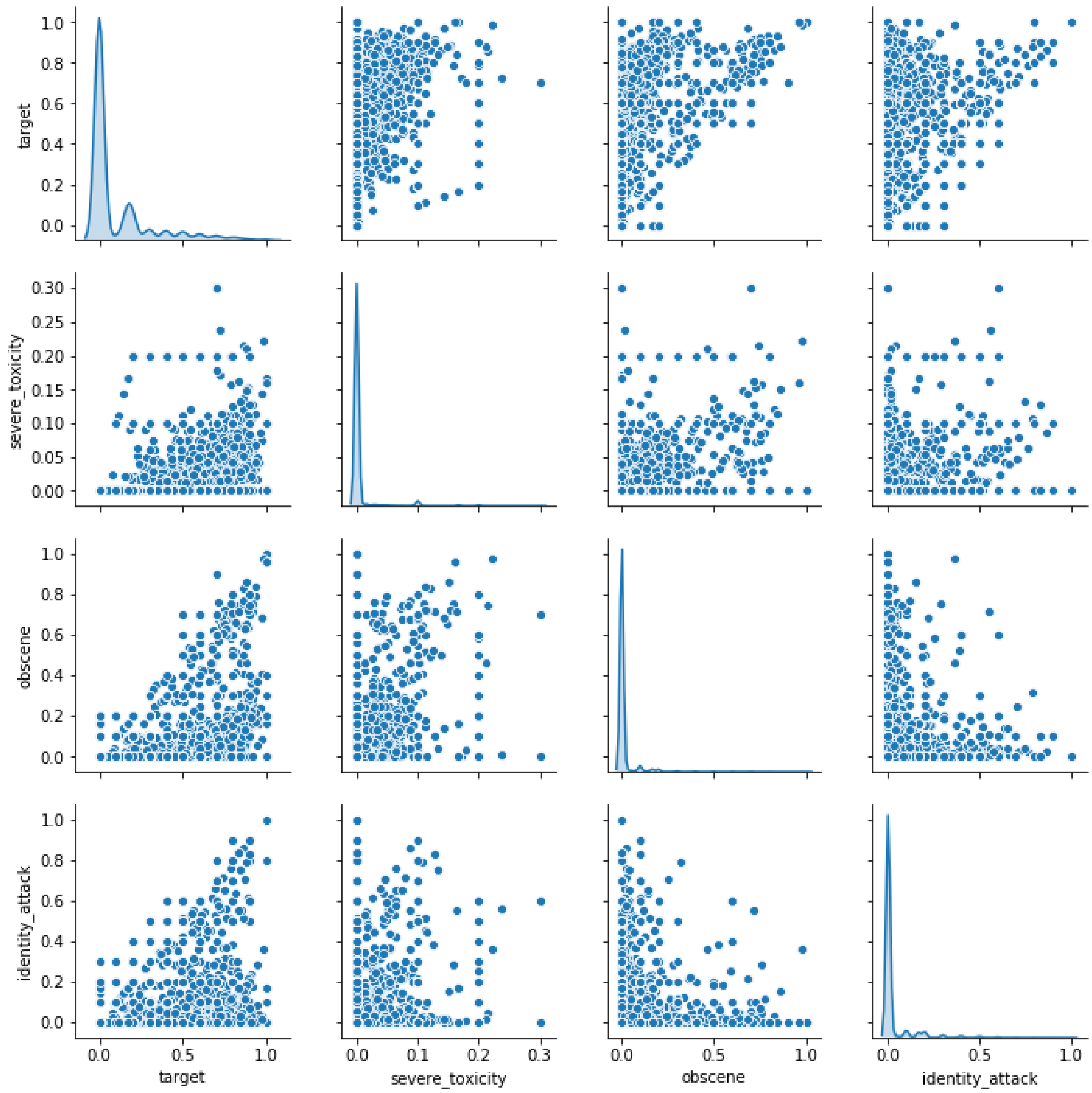
- df_train_part_1 and df_train_part_2 contain toxicity contributors for analysis.

[243]:
```
df_train_part_1 =␣
↪df_train_sample[['target','severe_toxicity','obscene','identity_attack']]
df_train_part_2 =␣
↪df_train_sample[['target','insult','threat','sexual_explicit']]

df_train_part_3 = df_train_sample[['target','funny','wow','sad']]
df_train_part_4 = df_train_sample[['target','likes','disagree']]
```

[13]:
```
import seaborn as sns
sns.pairplot(df_train_part_1, diag_kind="kde")
```

```
/home/bangui/.local/lib/python3.6/site-packages/matplotlib/__init__.py:886:
MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative
to the 'datapath' directory.
  "found relative to the 'datapath' directory.".format(key))
```

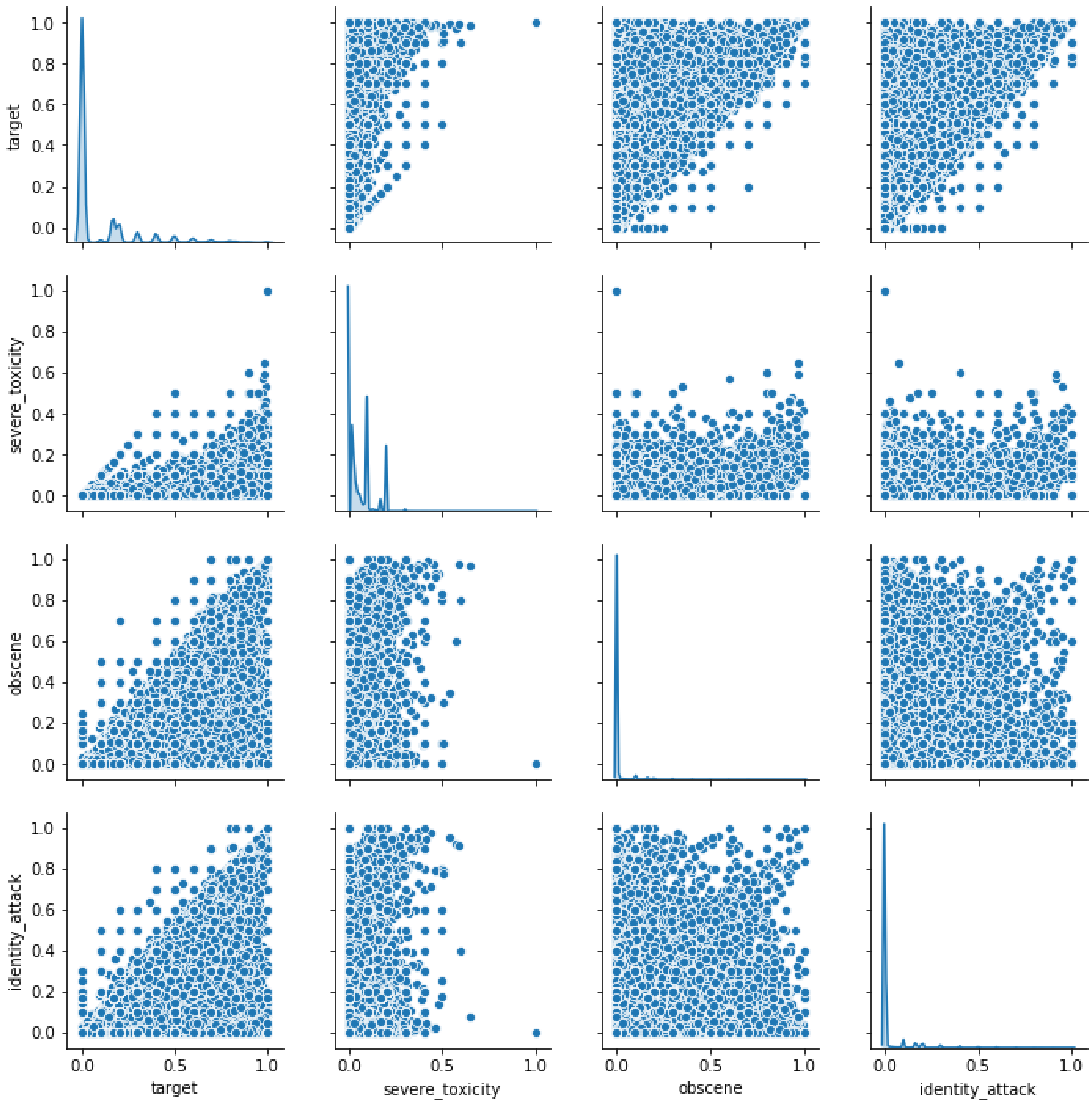
[13]:
```
<seaborn.axisgrid.PairGrid at 0x7fe4bf356b70>
```



[46]:
```
import seaborn as sns
sns.pairplot(df_train_part_1, diag_kind="kde")
```

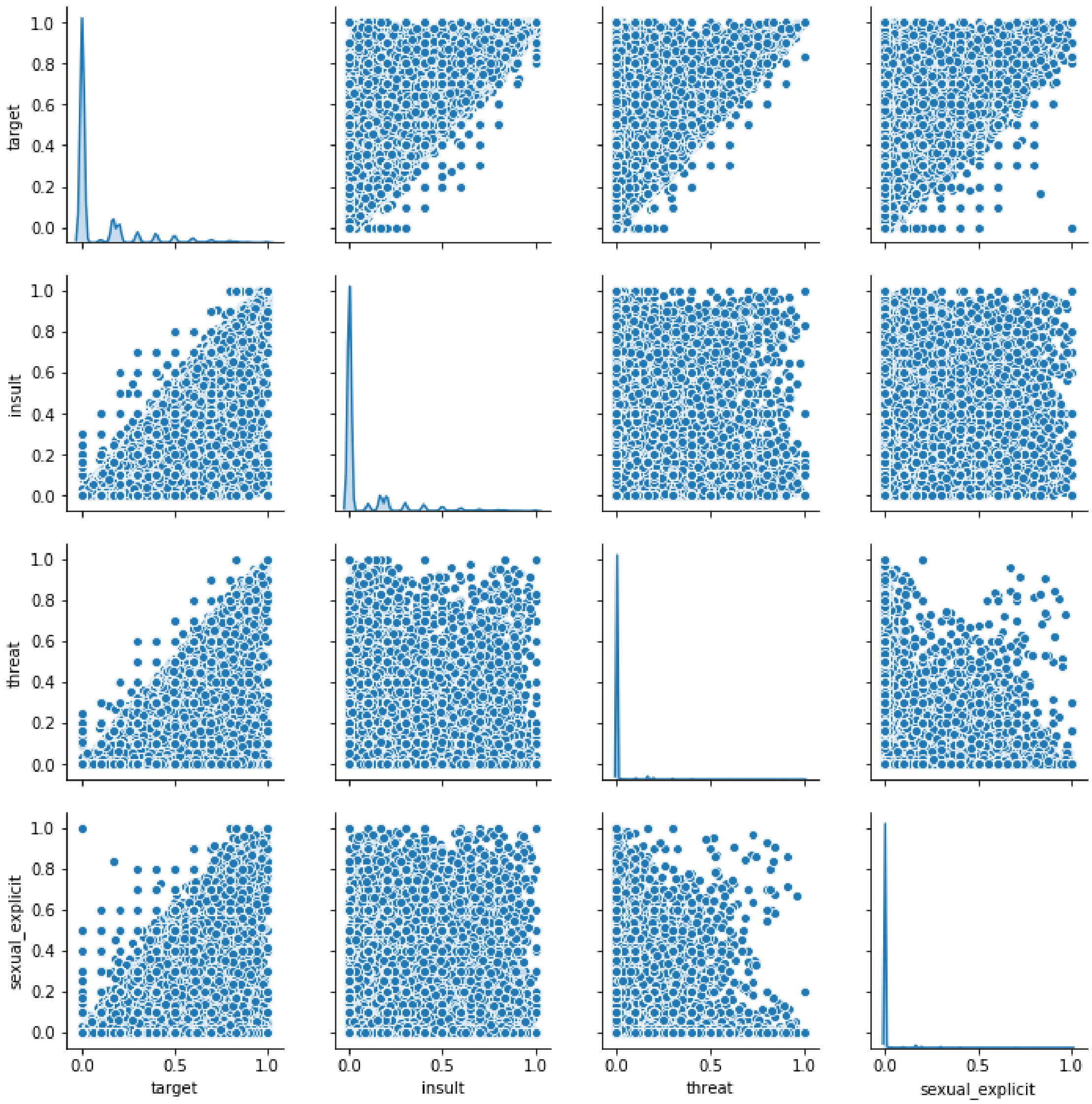
```
/home/bangui/.local/lib/python3.6/site-packages/matplotlib/__init__.py:886:
MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative
to the 'datapath' directory.
  "found relative to the 'datapath' directory.".format(key))
```

[46]: <seaborn.axisgrid.PairGrid at 0x7f399aada860>



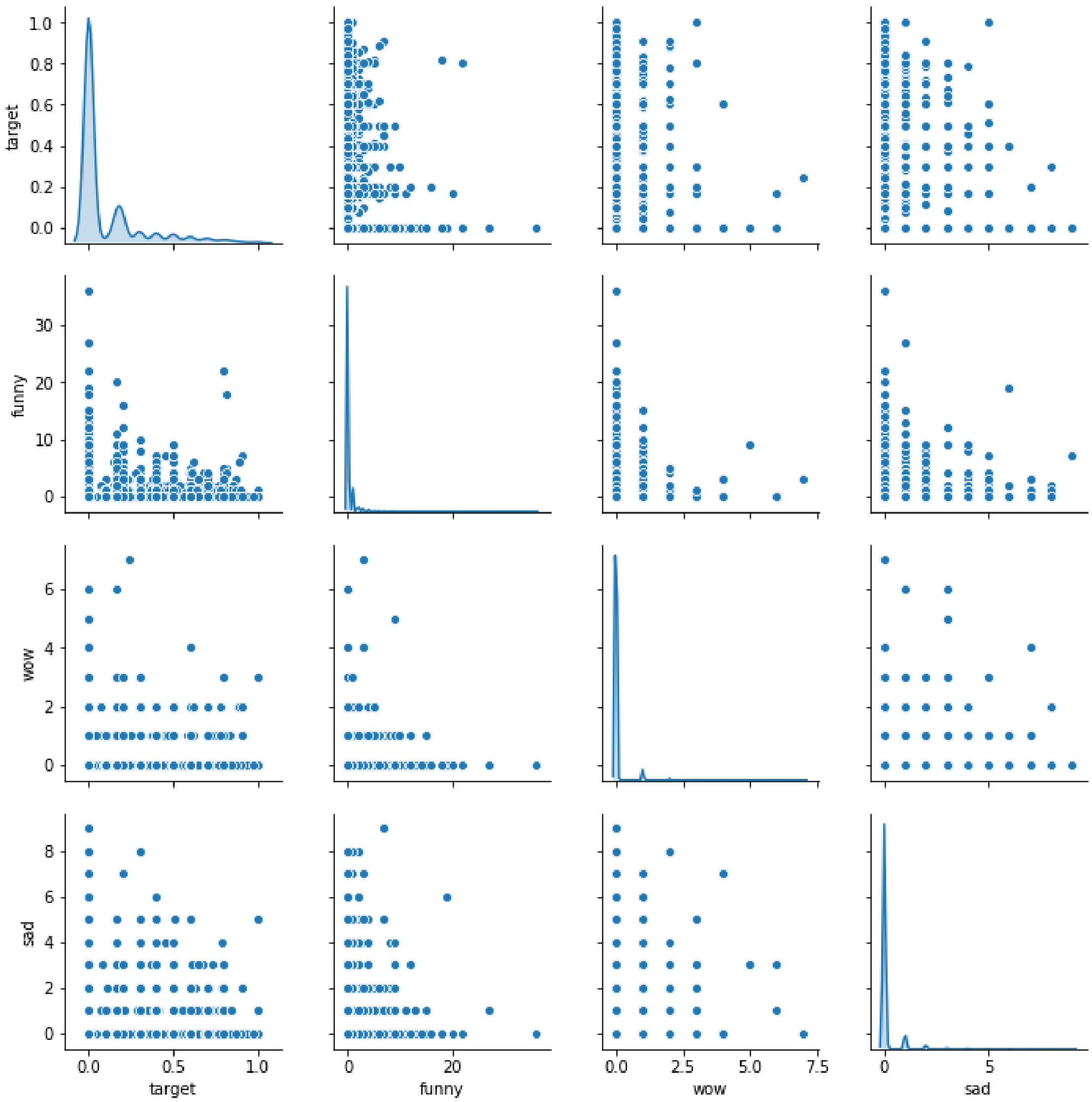
[47]:
```
import seaborn as sns
sns.pairplot(df_train_part_2, diag_kind="kde")
```

[47]: <seaborn.axisgrid.PairGrid at 0x7f3957043048>



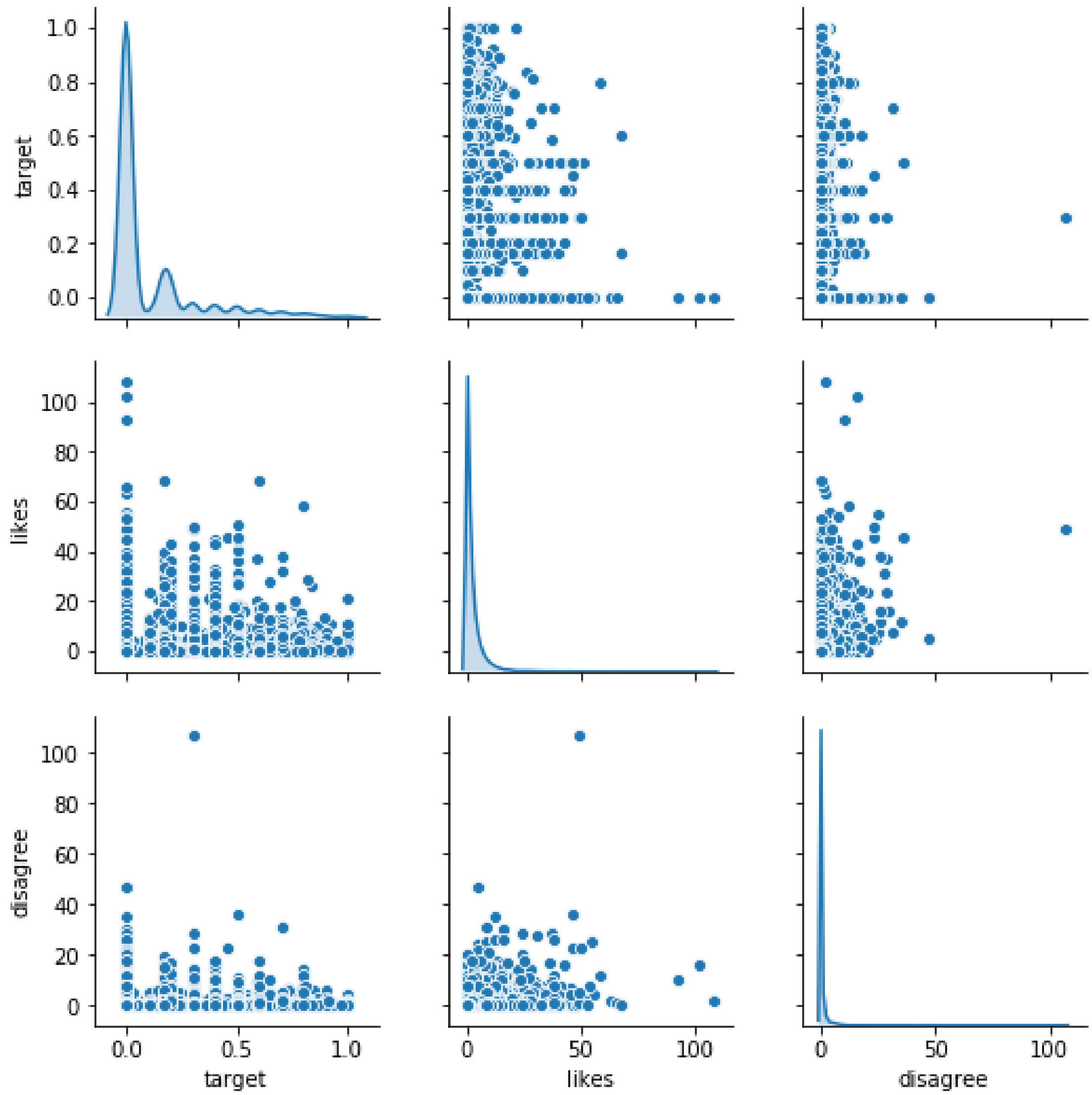
[256]:
```
import seaborn as sns
sns.pairplot(df_train_part_3, diag_kind="kde")
```

[256]: <seaborn.axisgrid.PairGrid at 0x7f8411b52d30>



[257]:
```
import seaborn as sns
sns.pairplot(df_train_part_4, diag_kind="kde")
```

[257]: `<seaborn.axisgrid.PairGrid at 0x7f8404e6ccc0>`



3.1 3.1. Feature engineering : `new_feature`

Combinaison of features that contribute to toxicity detection and forming `new_feature` shows that rectangle-triangle is replaced with a shape in which linear correlation between `target` and `new_feature` is more obvious.

All features does not contribute with the same weight to the toxicity.

`new_feature` is then formed from weights issued from correlations matrix.

3.1.1 Features to toxicity contribution are filtered

- Result lies on dataframe `df_train_sample`

[103]:
```
list_feature_excluded =␣
↪['comment_text','created_date','publication_id','article_id','identity_annotator_count','to
list_feature_excluded +=['rating','funny','wow','sad','likes','disagree']
list_feature_excluded
```

[103]:
```
['comment_text',
 'created_date',
 'publication_id',
 'article_id',
 'identity_annotator_count',
 'toxicity_annotator_count',
 'rating',
 'funny',
 'wow',
 'sad',
 'likes',
 'disagree']
```

[104]:
```
list_feature = df_train_sample.columns
list_feature = list_feature[1:-1]
list_feature = [feature for feature in list_feature if feature not in␣
↪list_feature_excluded ]
list_feature
```

[104]:
```
['target',
 'severe_toxicity',
 'obscene',
 'identity_attack',
 'insult',
 'threat',
 'sexual_explicit']
```

[271]:
```
df = df_train_sample[list_feature]
res = df.query('target>0.3').sample(2).head()
print(df_train_sample['comment_text'].loc[res.index[0]])
print()
print(df_train_sample['comment_text'].loc[res.index[1]])
res
```

```
So why not worry about the scandal of heterosexual priests?  Why not act against
the "straight" sexually active priests and hierarchs with families or
girlfriends on the side?  Why not act against the "straight" priests and
hierarchs who abuse or hide abusers?  There's been a helluva lot more scandal
from that, despite some people's efforts to portray it as a gay issue.  The
claim of protecting the Church by keeping gay men out of the seminaries is,
bluntly, hypocrisy.

Pathetic. Is that the best the Globe can do?
A drop of (so far) less than 1% is somehow relevant to judge Trump?
```

```
How many 1% drops did we have during the Obama years?

When the market goes down big time, AND IT WILL, it will have little to do with
Trump, but with China, Fed policies of the past decade, and a war somewhere that
probably is mostly due to Obama's weak leadership in foreign affairs.

By spewing this kind of nonsense, the media enfeeble themselves  and take
themselves out of the conversation of serious people discussing serious issues.
```

[271]:

```
          target  severe_toxicity   obscene  identity_attack    insult  \
357023  0.400000              0.1  0.100000              0.2  0.300000
555712  0.833333              0.0  0.166667              0.0  0.833333

        threat  sexual_explicit
357023     0.0              0.0
555712     0.0              0.0
```

[267]:

```

```

```
The sore losers will be out heavy on this one. The truth drives them batty.
Say... come on now king kong... er i mean waveloser. Lets hear your brilliant
analysis. Wootwoot!!!!

I only got the email notification which said something about Smug. I believe in
full disclosure - just so all our presuppositions are on the table. Yes I am a
white, Christian, conservative male - just the type you most likely spent most
of your years persecuting. You use a clearly isolated incident with the EPD, and
ignore the hundreds of thousands of good things they do. Pretty pathetic if you
ask me, and the taxpayers are paying you for this as well. SAD
```

[272]:

```
len("I only got the email notification which said something about Smug. I␣
 ↪believe in full disclosure - just so all our presuppositions are on the␣
 ↪table. Yes I am a white, Christian, conservative male - just the type you␣
 ↪most likely spent most of your years persecuting. You use a clearly isolated␣
 ↪incident with the EPD, and ignore the hundreds of thousands of good things␣
 ↪they do. Pretty pathetic if you ask me, and the taxpayers are paying you for␣
 ↪this as well. SAD")
```

[272]: 461

[105]:

```
df_train_sample = df_train_sample[list_feature]
```

3.1.2 Feature engineering : building `new_feature`

- Features such as :
  - `threat`, `insult`, `sexual_explicit`, `identity_attack`, `obscene`, `severe_toxicity` are combined all-together forming to a new feature.

- Those features do contribute to toxicity text detection.
- Those features draw a rectangle-triangle compared with `target` column.
- A linear correlation relation is suspected between those features and `target`.

In a first step, all weights are fixed to 1. It is supposed that all those features have the same weight considering target value.

Weights are handled into `ser_weight_unit` Series.

[106]:
```
import numpy as np
import pandas as pd

df_train_sample.shape
arr_unit = np.array([1. for value in list_feature])
ser_weight_unit =pd.Series( arr_unit, index=list_feature)
print(ser_weight_unit)
```

```
target              1.0
severe_toxicity     1.0
obscene             1.0
identity_attack     1.0
insult              1.0
threat              1.0
sexual_explicit     1.0
dtype: float64
```

- `df_train_sample` is added with a new column named `new_feature`. This new feature is the combination of all other columns with weight values to 1.

[112]:
```
import p9_util

df_train_sample = p9_util.df_weight_newFeature(df_train_sample,␣
 ↪ser_weight_unit, list_feature, 'new_feature')
print()
print(df_train_sample.columns)
print(df_train_sample.shape)
```

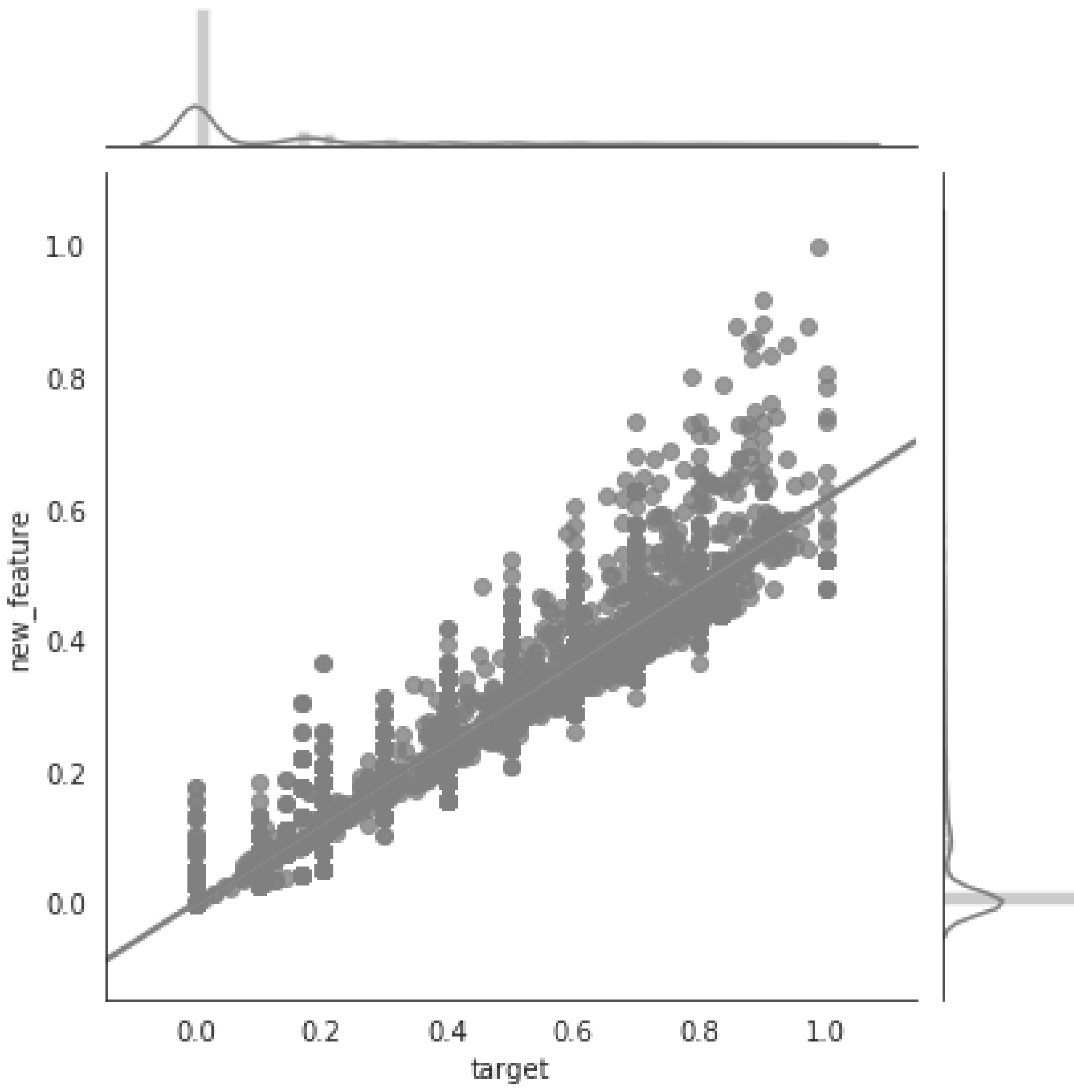
```
target 1.0
severe_toxicity 1.0
obscene 1.0
identity_attack 1.0
insult 1.0
threat 1.0
sexual_explicit 1.0

Index(['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult',
       'threat', 'sexual_explicit', 'new_feature'],
      dtype='object')
(18049, 8)
```

[108]:
```
df_train_sample['new_feature'].max()
```

[108]:
```
1.0
```

[109]:
```
import scipy

import p3_util_plot

var1 = 'target'
var2 = 'new_feature'

pearson = scipy.stats.pearsonr(df_train_sample[var1],df_train_sample[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
↪pearson[0]))

#p3_util_plot.df_sns_joint_plot(df_train_sample, var1, var2,␣
↪parameter_kind='reg', parameter_color='grey')
p3_util_plot.df_sns_joint_plot(df_train_sample, var1, var2,␣
↪parameter_color='grey')
```

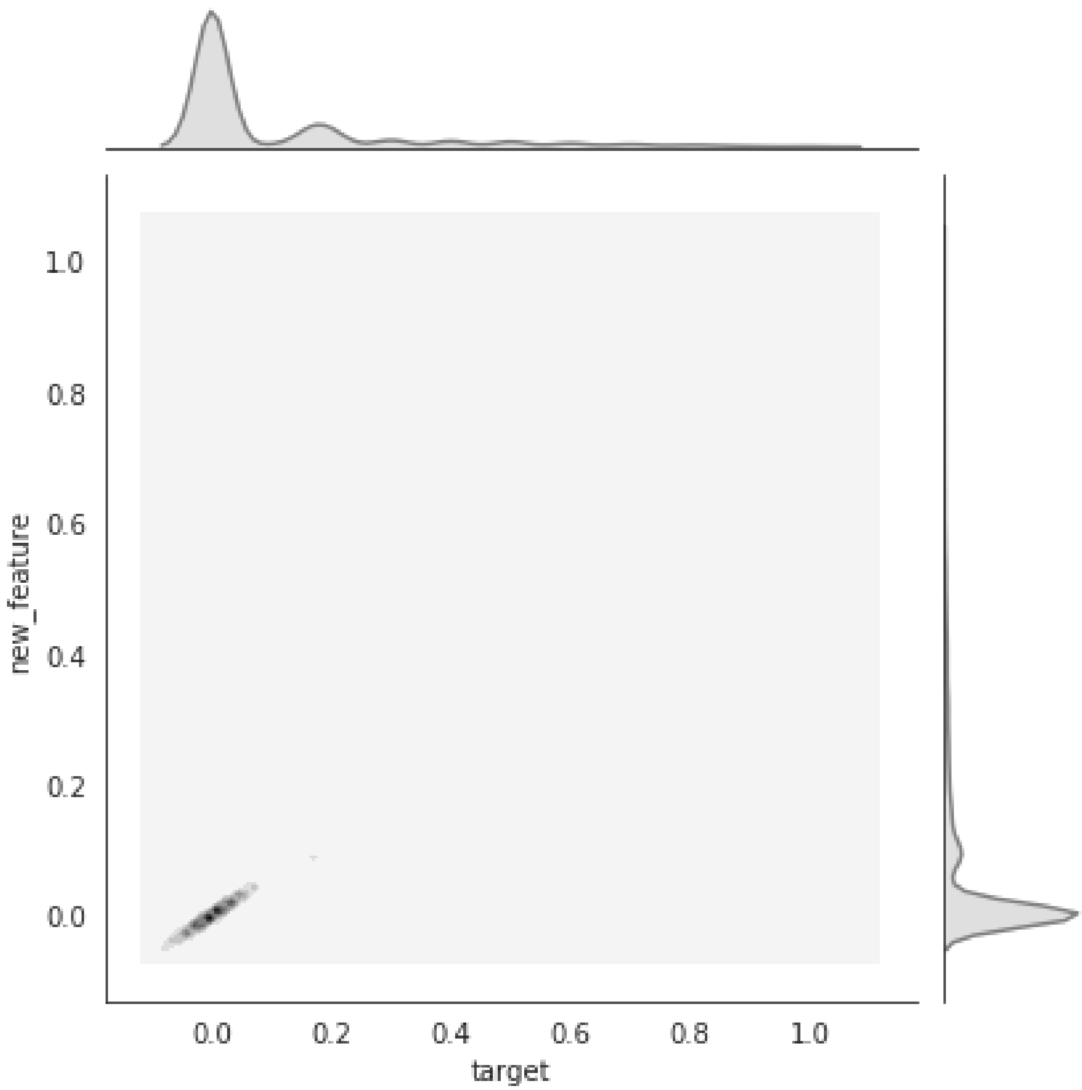
```
Pearson coefficient between (target,new_feature) = 0.9772645985025511

/home/bangui/.local/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Displaying density distributions give additional interpretation.
Features are correlated strongly in interval `[0.0 , 0.2]`

[26]:
```
import scipy
import p3_util_plot
var1 = 'target'
var2 = 'new_feature'

pearson = scipy.stats.pearsonr(df_train_sample[var1],df_train_sample[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
 ↪pearson[0]))

p3_util_plot.df_sns_joint_plot(df_train_sample, var1, var2,␣
 ↪parameter_kind='kde', parameter_color='grey')
```

```
Pearson coefficient between (target,new_feature) = 0.9772645985025511
```



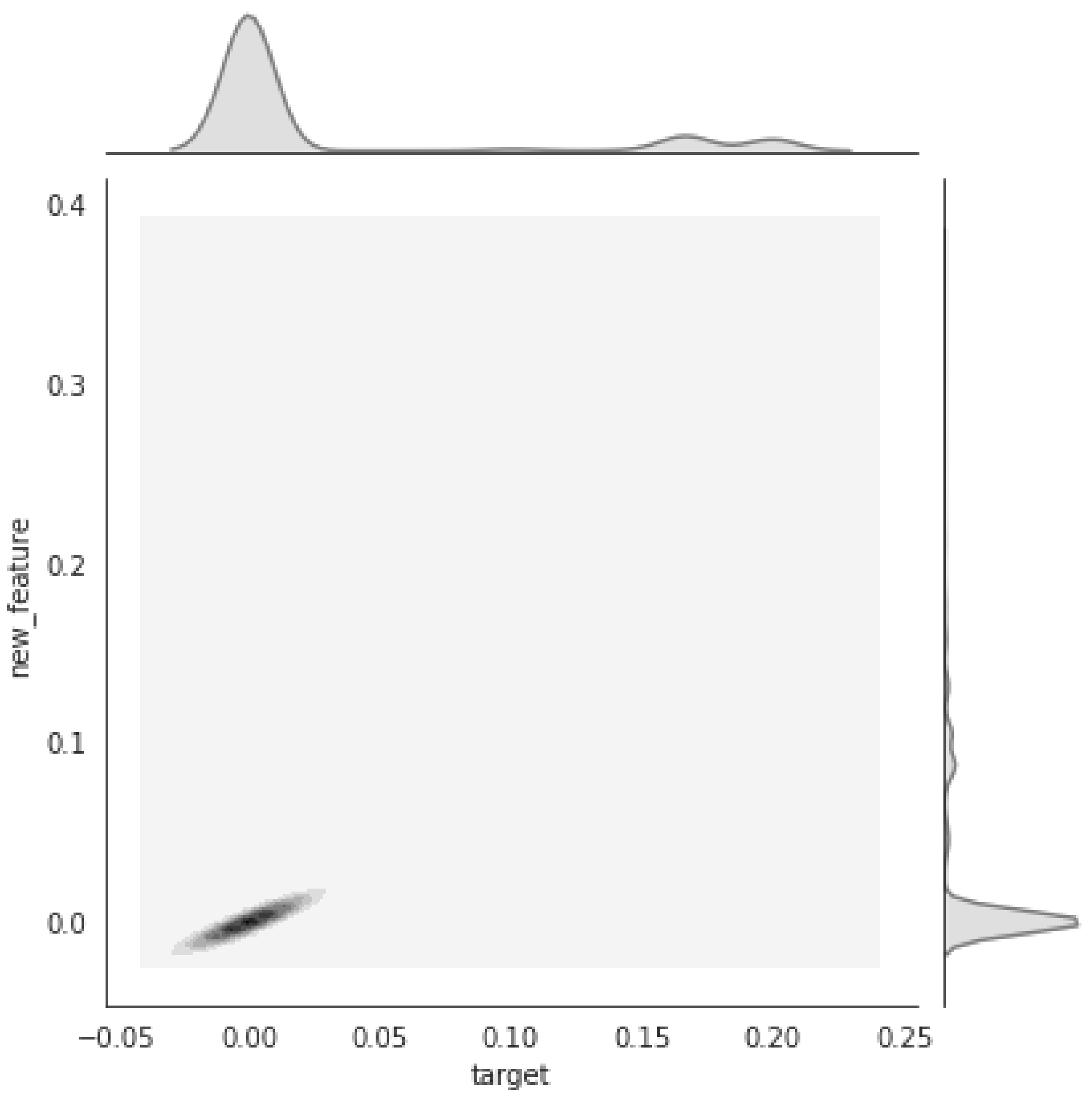
Then let's plot target values >0.2 and target values <= 0.2
This is achieved applying query to `df_train_sample` dataframe.

[29]:
```
select="target <= 0.2"
df_train_sample_select_0 = df_train_sample.query(select, inplace=False)

select="target > 0.2"
df_train_sample_select_1 = df_train_sample.query(select, inplace=False)
```

[30]:
```
import scipy
import p3_util
```

```
pearson = scipy.stats.
 ↪pearsonr(df_train_sample_select_0[var1],df_train_sample_select_0[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
 ↪pearson[0]))
p3_util_plot.df_sns_joint_plot(df_train_sample_select_0, var1, var2,␣
 ↪parameter_kind='kde', parameter_color='grey')
```

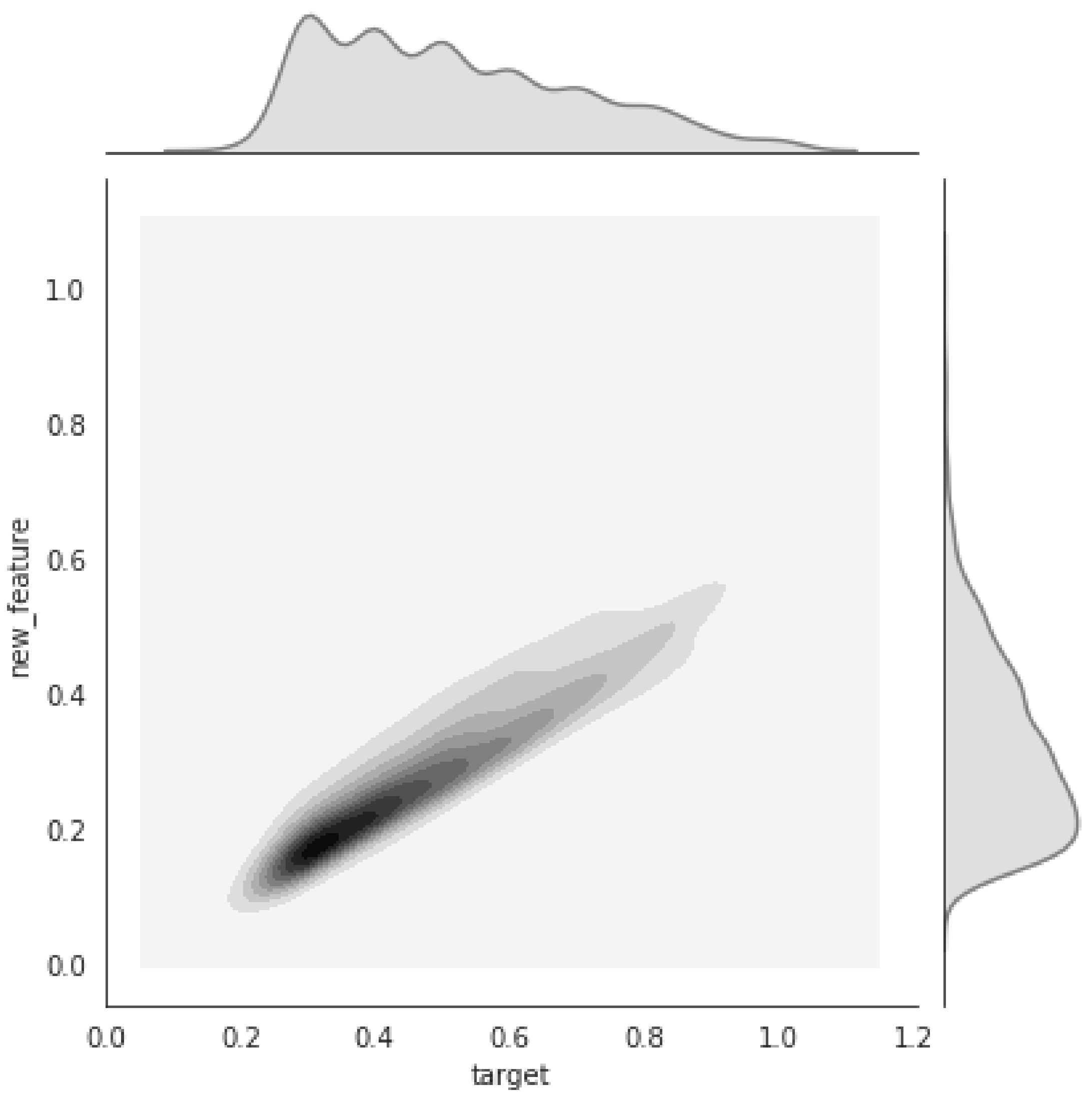
```
Pearson coefficient between (target,new_feature) = 0.929778393032554
```



[31]:
```
import scipy
import p3_util
```

```
pearson = scipy.stats.
 ↪pearsonr(df_train_sample_select_1[var1],df_train_sample_select_1[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
 ↪pearson[0]))
p3_util_plot.df_sns_joint_plot(df_train_sample_select_1, var1, var2,␣
 ↪parameter_kind='kde', parameter_color='grey')
```

```
Pearson coefficient between (target,new_feature) = 0.9035771413019859
```



3.2 Weighting features with matrix correlations

[32]:
```
df_train_sample.max()
```

[32]:
```
target              1.0
severe_toxicity     0.3
obscene             1.0
identity_attack     1.0
insult              1.0
threat              1.0
sexual_explicit     1.0
new_feature         1.0
dtype: float64
```

[33]:
```
if 'new_feature' in df_train_sample :
    del(df_train_sample['new_feature'])
```

[34]:
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
print(std_scaler.fit(df_train_sample))

X_std = std_scaler.transform(df_train_sample)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

[35]:
```
df_train_sample.columns
```

[35]:
```
Index(['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult',
       'threat', 'sexual_explicit'],
      dtype='object')
```

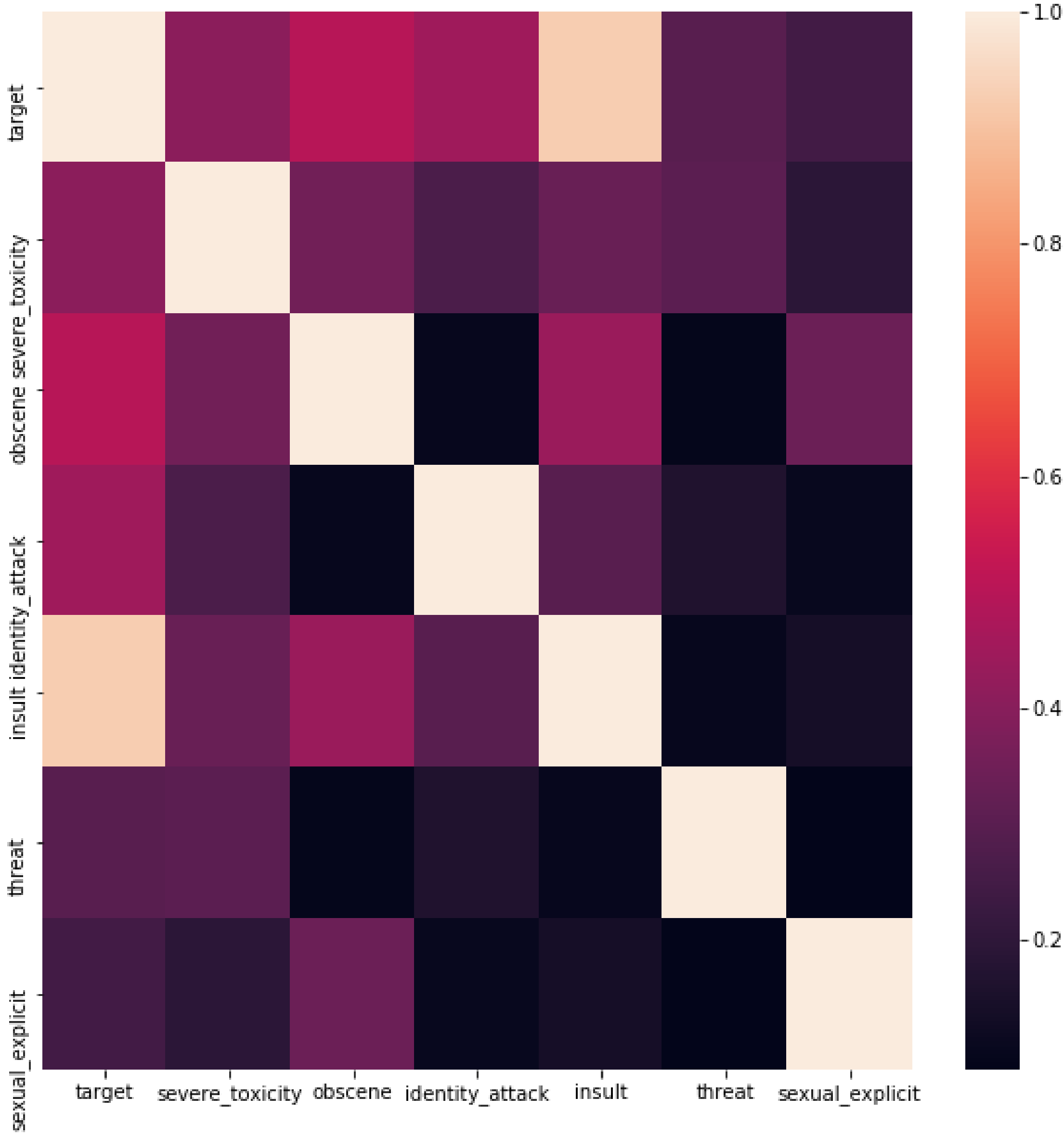
[36]:
```
df_train_sample_std = pd.DataFrame(X_std, columns=df_train_sample.columns)
```

[37]:
```
df_train_sample_std.head(3)
```

[37]:
```
     target  severe_toxicity   obscene  identity_attack    insult    threat  \
0 -0.527767        -0.209535 -0.215539        -0.288058 -0.466598 -0.190872
1 -0.527767        -0.209535 -0.215539        -0.288058 -0.466598 -0.190872
2 -0.527767        -0.209535 -0.215539        -0.288058 -0.466598 -0.190872

   sexual_explicit
0        -0.136735
1        -0.136735
2        -0.136735
```

[38]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

X_std_corr = df_train_sample_std.corr()
```

```
f, ax = plt.subplots(figsize=(10, 10))
_z = sns.heatmap(X_std_corr, annot=False)
```



[45]:
```
X_std_corr[:]
```

[45]:

```
                   target  severe_toxicity   obscene  identity_attack  \
target           1.000000         0.405802  0.500494         0.448081
severe_toxicity  0.405802         1.000000  0.352787         0.267649
obscene          0.500494         0.352787  1.000000         0.104657
identity_attack  0.448081         0.267649  0.104657         1.000000
insult           0.925399         0.333349  0.438020         0.295346
threat           0.296270         0.303908  0.096228         0.164630
```

```
sexual_explicit  0.244588         0.193290  0.338451         0.107064

                  insult   threat  sexual_explicit
target          0.925399  0.296270         0.244588
severe_toxicity 0.333349  0.303908         0.193290
obscene         0.438020  0.096228         0.338451
identity_attack 0.295346  0.164630         0.107064
insult          1.000000  0.104562         0.139459
threat          0.104562  1.000000         0.087756
sexual_explicit 0.139459  0.087756         1.000000
```

Weights assignement for building `new_feature` Weights are issued from correlation matrix.

[49]:
```
list_feature
```

[49]:
```
['target',
 'severe_toxicity',
 'obscene',
 'identity_attack',
 'insult',
 'threat',
 'sexual_explicit']
```

[50]:
```
X_std_corr['target']
```

[50]:
```
target             1.000000
severe_toxicity    0.405802
obscene            0.500494
identity_attack    0.448081
insult             0.925399
threat             0.296270
sexual_explicit    0.244588
Name: target, dtype: float64
```

[51]:
```
df_train_filtered_sample = df_weight_newFeature(df_train_sample,␣
 ↪X_std_corr['target'], list_feature, 'new_feature')
```

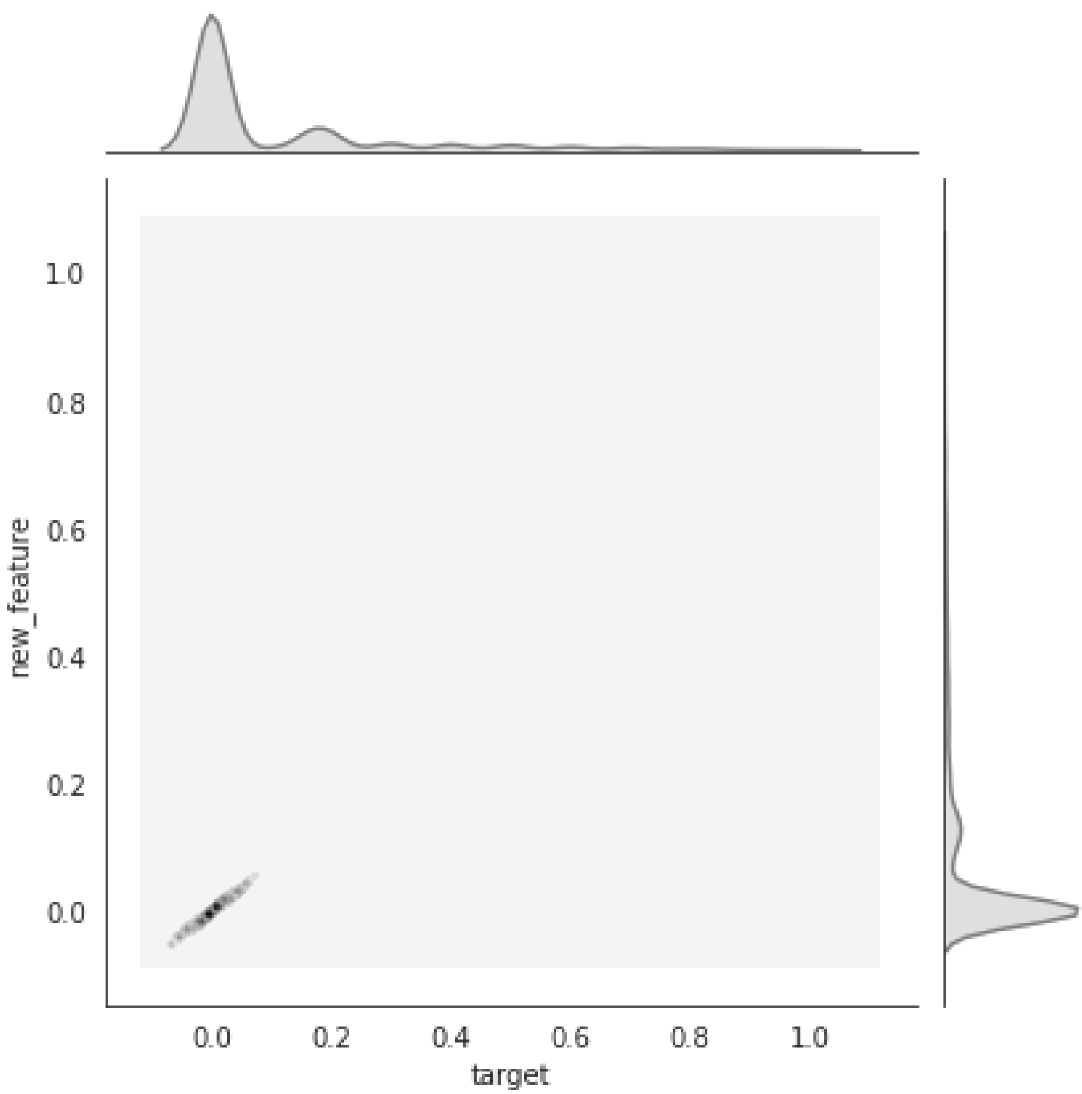
```
target 1.0
severe_toxicity 0.40580207453035044
obscene 0.5004938914679019
identity_attack 0.44808065477165393
insult 0.9253985594798301
threat 0.29627000932665676
sexual_explicit 0.24458800527543303
```

[52]:
```
df_train_filtered_sample['new_feature'].min(),␣
 ↪df_train_filtered_sample['new_feature'].max()
```

[52]:
```
(0.0, 1.0)
```

[53]:
```
import scipy

import p3_util_plot
var1='target'
var2 = 'new_feature'

pearson = scipy.stats.
↪pearsonr(df_train_filtered_sample[var1],df_train_filtered_sample[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
↪pearson[0]))
p3_util_plot.df_sns_joint_plot(df_train_filtered_sample, var1, var2,␣
↪parameter_kind='kde', parameter_color='grey')
```

```
Pearson coefficient between (target,new_feature) = 0.9897087258200451
```



Using correlation matrix for weighting features when building new_feature improve global correlation between new_feature and target (0.989 vs 0.977).

Display Pearson coefficient between all features against target

[83]:
```
df_train_filtered_sample.columns
```

[83]:
```
Index(['severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat',
       'sexual_explicit'],
      dtype='object')
```

[55]:
```
import scipy
import p3_util_plot

var1 = 'target'
for var2 in df_train_filtered_sample.columns :
    if var2 != 'target' :
        #p3_util_plot.df_sns_joint_plot(df_train_filtered_sample, var1, var2,␣
 ↪parameter_kind='reg', parameter_color='grey')


        pearson = scipy.stats.
 ↪pearsonr(df_train_filtered_sample[var1],df_train_filtered_sample[var2])
        print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
 ↪pearson[0]))
    else :
        pass
```

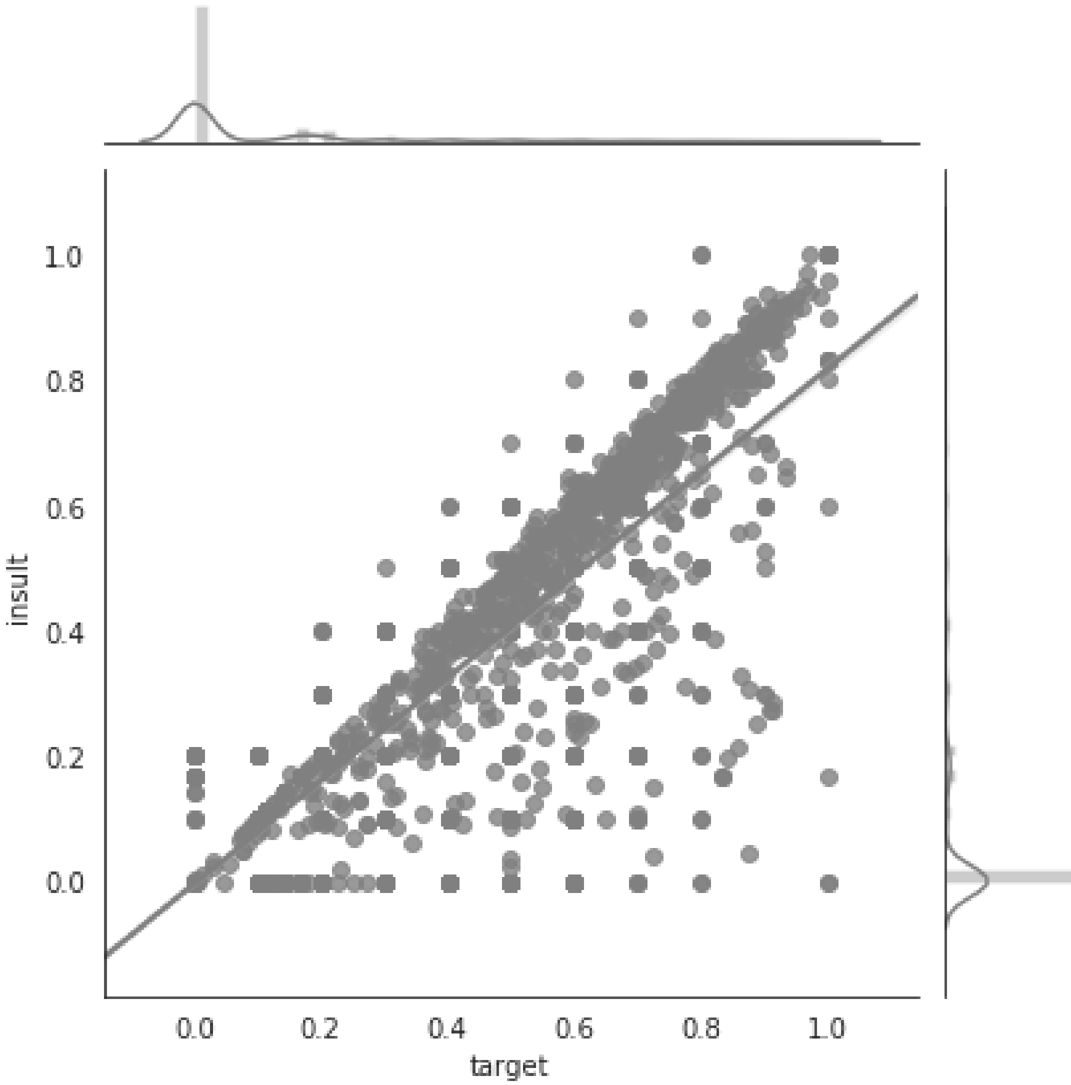
```
Pearson coefficient between (target,severe_toxicity) = 0.4058020745302237

Pearson coefficient between (target,obscene) = 0.5004938914680612

Pearson coefficient between (target,identity_attack) = 0.448080654771592

Pearson coefficient between (target,insult) = 0.9253985594798431

Pearson coefficient between (target,threat) = 0.29627000932655323

Pearson coefficient between (target,sexual_explicit) = 0.2445880052754865

Pearson coefficient between (target,new_feature) = 0.9897087258200451
```

Correlation between insult and target

[56]:
```
import scipy
import p3_util_plot
```

```python
var1 = 'target'
var2 = 'insult'
p3_util_plot.df_sns_joint_plot(df_train_filtered_sample, var1, var2,␣
 ↪parameter_kind='reg', parameter_color='grey')


pearson = scipy.stats.
 ↪pearsonr(df_train_filtered_sample[var1],df_train_filtered_sample[var2])
print("\nPearson coefficient between ({},{}) = {}".format(var1,var2,␣
 ↪pearson[0]))
```

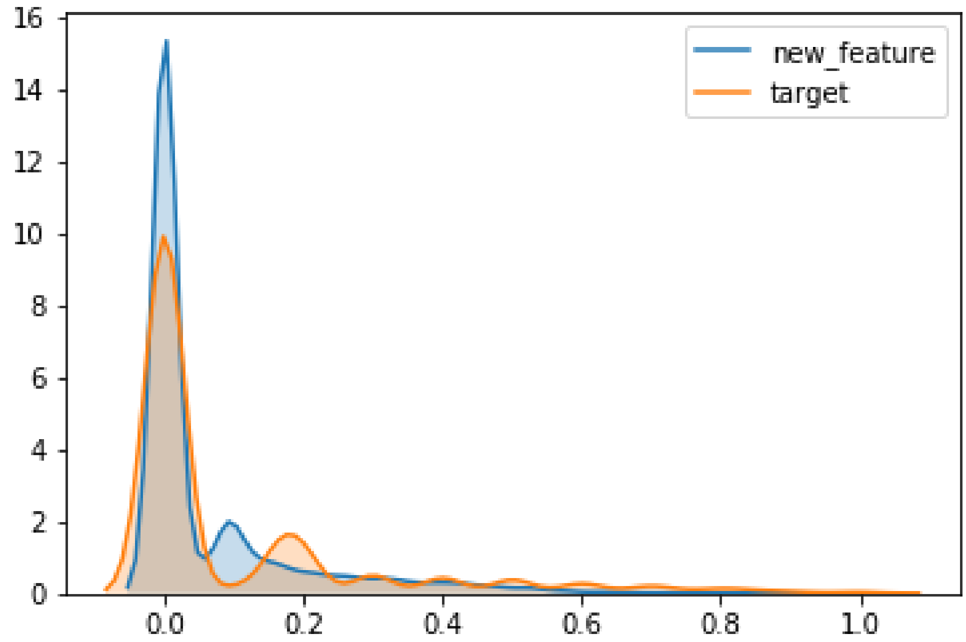
```
/home/bangui/.local/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval


Pearson coefficient between (target,insult) = 0.9253985594798431
```



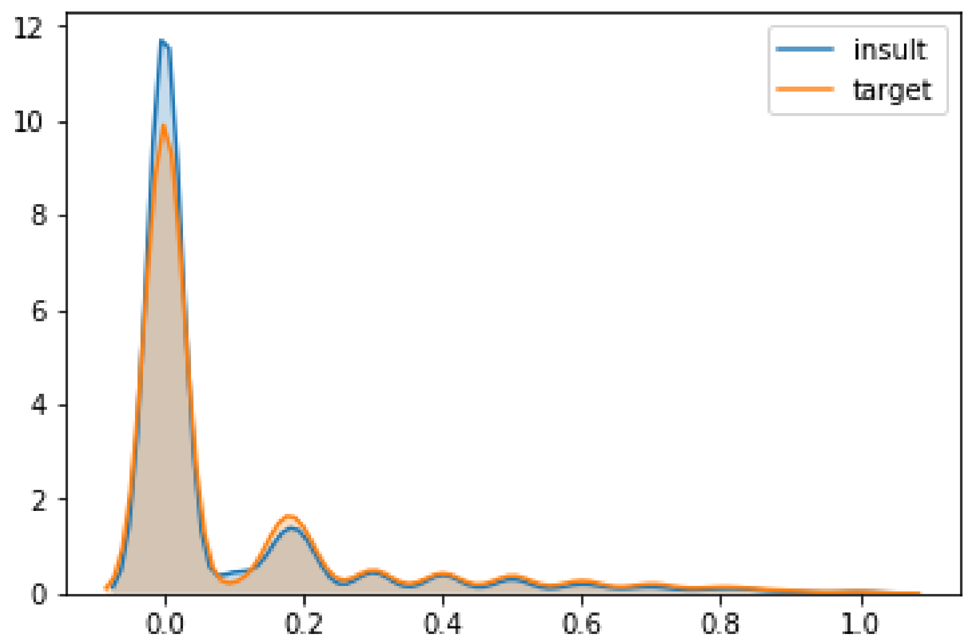
[82]:
```
df_train_filtered_sample.columns
```

[82]:
```
Index(['severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat',
       'sexual_explicit'],
      dtype='object')
```

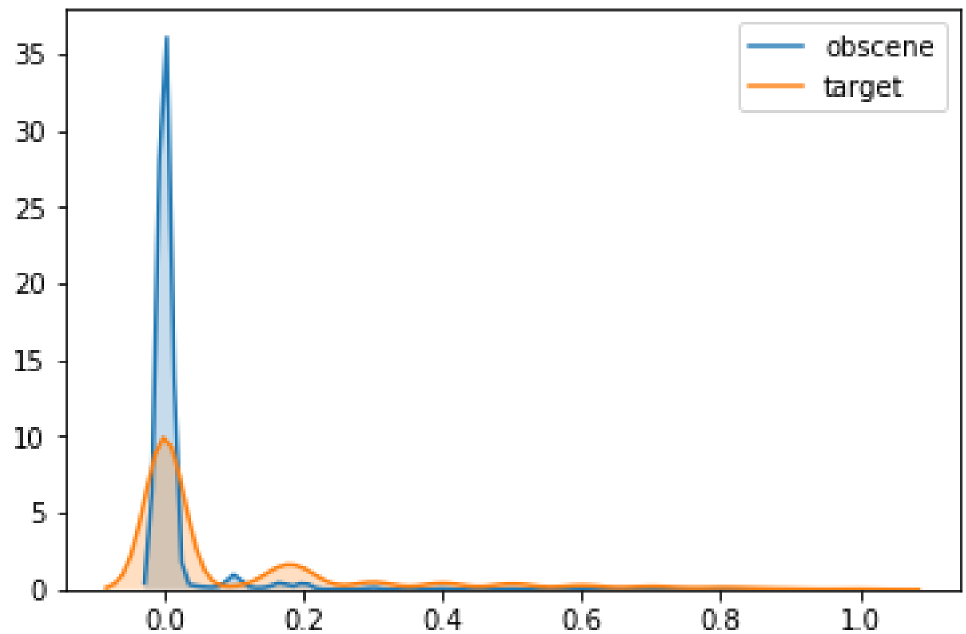
[116]:
```
import seaborn as sns
for col in ['new_feature','target']:
    sns.kdeplot(df_train_sample[col], shade=True)
```



[118]:
```
import seaborn as sns
for col in ['insult','target']:
    sns.kdeplot(df_train_sample[col], shade=True)
```



[119]:
```python
import seaborn as sns
for col in ['obscene','target']:
    sns.kdeplot(df_train_sample[col], shade=True)
```



[115]:
```python
df_train_sample.columns
```

[115]:
```
Index(['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult',
       'threat', 'sexual_explicit', 'new_feature'],
      dtype='object')
```

3.2.1 Weights issued from a linear regression

Target is extracted from dataframe Dataframe keeps features used only for linear regression.

[57]:
```python
df_train_sample.columns
```

[57]:
```
Index(['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult',
       'threat', 'sexual_explicit', 'new_feature'],
      dtype='object')
```

[58]:
```python
if 'target' in df_train_sample.columns:
    df_train_label = df_train_sample.pop('target')

if 'new_feature' in df_train_sample.columns:
    del(df_train_sample['new_feature'])
```

[59]:
```python
df_train_sample.shape
```

[59]:
```
(18049, 6)
```

Data is standardized

```
[60]: from sklearn.preprocessing import StandardScaler
if False :
    std_scaler = StandardScaler()
    print(std_scaler.fit(df_train_sample))

    X_std = std_scaler.transform(df_train_sample)
else :
    X_std = df_train_sample.copy()
```

Linear regression model is built

```
[61]: from sklearn.linear_model import LinearRegression

model_regression = LinearRegression().fit(X_std, df_train_label)
```

Displaying correlation coefficient Coefficient issued from linear regression and those issued from matrix correlation are both displayed.

Some of them have closed values, such as `threat` and `identity_attack`, others strongly differ.

```
[62]: for feature, lr_coef, corr_coef in zip(df_train_sample.columns,␣
 ↪model_regression.coef_,X_std_corr['target'][1:]):
    print("Feature: {} : linear regression coeff= {} / Correlation matrix␣
 ↪coeff= {}".format(feature, lr_coef, corr_coef))
```

```
Feature: severe_toxicity : linear regression coeff= -0.05362561443422439 /
Correlation matrix coeff= 0.40580207453035044
Feature: obscene : linear regression coeff= 0.2619629875316454 / Correlation
matrix coeff= 0.5004938914679019
Feature: identity_attack : linear regression coeff= 0.40888237843007297 /
Correlation matrix coeff= 0.44808065477165393
Feature: insult : linear regression coeff= 0.9155839990091565 / Correlation
matrix coeff= 0.9253985594798301
Feature: threat : linear regression coeff= 0.6734955460989804 / Correlation
matrix coeff= 0.29627000932665676
Feature: sexual_explicit : linear regression coeff= 0.31847938964928624 /
Correlation matrix coeff= 0.24458800527543303
```

```
[69]: df_test_sample.columns
```

```
[69]: Index(['id', 'target', 'comment_text', 'severe_toxicity', 'obscene',
       'identity_attack', 'insult', 'threat', 'created_date', 'publication_id',
       'article_id', 'rating', 'funny', 'wow', 'sad', 'likes', 'disagree',
       'sexual_explicit', 'identity_annotator_count',
       'toxicity_annotator_count'],
      dtype='object')
```

[70]:
```python
if 'target' in df_test_sample.columns :
    df_test_target = df_test_sample.pop('target')
df_test_sample.columns
```

[70]:
```
Index(['id', 'comment_text', 'severe_toxicity', 'obscene', 'identity_attack',
       'insult', 'threat', 'created_date', 'publication_id', 'article_id',
       'rating', 'funny', 'wow', 'sad', 'likes', 'disagree', 'sexual_explicit',
       'identity_annotator_count', 'toxicity_annotator_count'],
      dtype='object')
```

[71]:
```python
list_feature
if 'target' in list_feature :
    list_feature.remove('target')
print(list_feature)
```

```
['severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat',
'sexual_explicit']
```

[72]:
```python
df_test_sample = df_test_sample[list_feature]
```

[ ]:
```python
X_std = std_scaler.transform(df_test_sample)
```

[76]:
```python
model_regression.score(X_std, df_test_target)
```

[76]:
```
-48.750536382222116
```

3.2.2 Distribution of target from test dataset

[122]:
```python
df_test_sample.columns
```

[122]:
```
Index(['id', 'target', 'comment_text', 'severe_toxicity', 'obscene',
       'identity_attack', 'insult', 'threat', 'created_date', 'publication_id',
       'article_id', 'rating', 'funny', 'wow', 'sad', 'likes', 'disagree',
       'sexual_explicit', 'identity_annotator_count',
       'toxicity_annotator_count'],
      dtype='object')
```

[123]:
```python
import numpy as np
import pandas as pd

df_train_sample.shape
arr_unit = np.array([1. for value in list_feature])
ser_weight_unit =pd.Series( arr_unit, index=list_feature)
print(ser_weight_unit)
```

```
target             1.0
severe_toxicity    1.0
obscene            1.0
identity_attack    1.0
insult             1.0
```

```
threat              1.0
sexual_explicit     1.0
dtype: float64
```

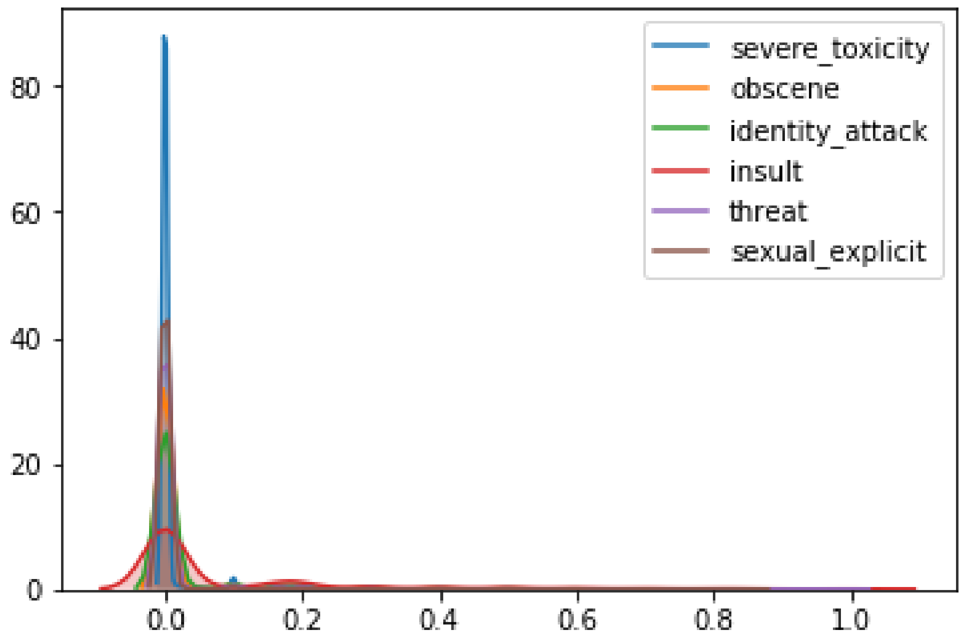
[124]:

```
import p9_util

df_test_sample = p9_util.df_weight_newFeature(df_test_sample, ser_weight_unit,␣
 ↪list_feature, 'new_feature')
print()
print(df_test_sample.columns)
print(df_test_sample.shape)
```

```
target 1.0
severe_toxicity 1.0
obscene 1.0
identity_attack 1.0
insult 1.0
threat 1.0
sexual_explicit 1.0

Index(['id', 'target', 'comment_text', 'severe_toxicity', 'obscene',
       'identity_attack', 'insult', 'threat', 'created_date', 'publication_id',
       'article_id', 'rating', 'funny', 'wow', 'sad', 'likes', 'disagree',
       'sexual_explicit', 'identity_annotator_count',
       'toxicity_annotator_count', 'new_feature'],
      dtype='object')
(6016, 21)
```

[74]:

```
import seaborn as sns

for col in df_test_sample.columns:
    sns.kdeplot(df_test_sample[col], shade=True)
```



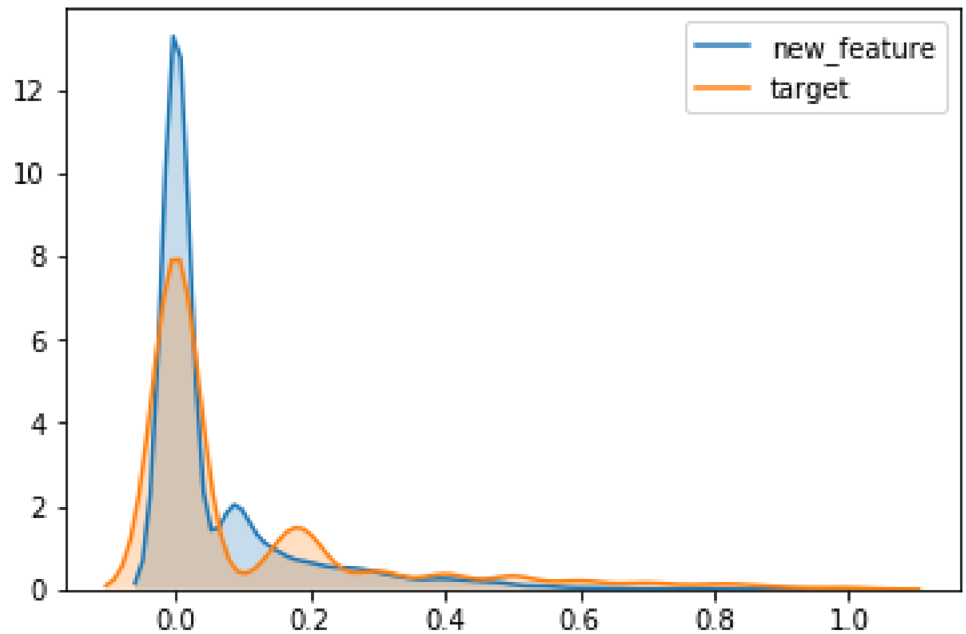
This display shows a long trail above 0.2 `target` column is added to `df_test_sample`. This will allow to extract a subset of `df_test_sample` filtered with `target` >0.2

[75]:
```
if 'target' not in df_test_sample.columns:
    df_test_sample['target'] = df_test_target

df_test_sample_filtered = df_test_sample.query('target<0.2')
```

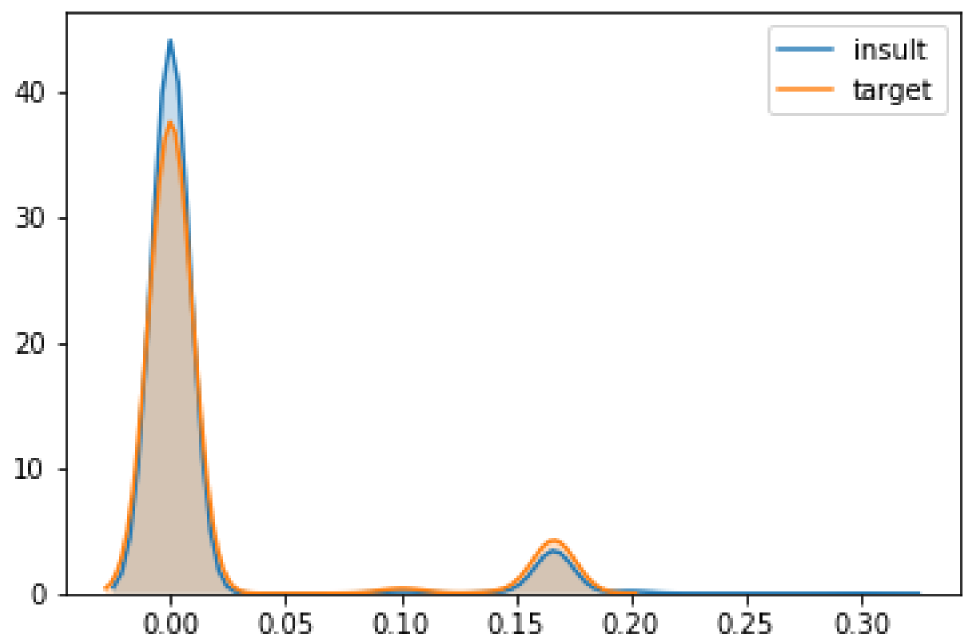
[76]:
```
df_test_sample_filtered.columns
```

[76]:
```
Index(['severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat',
       'sexual_explicit', 'target'],
      dtype='object')
```

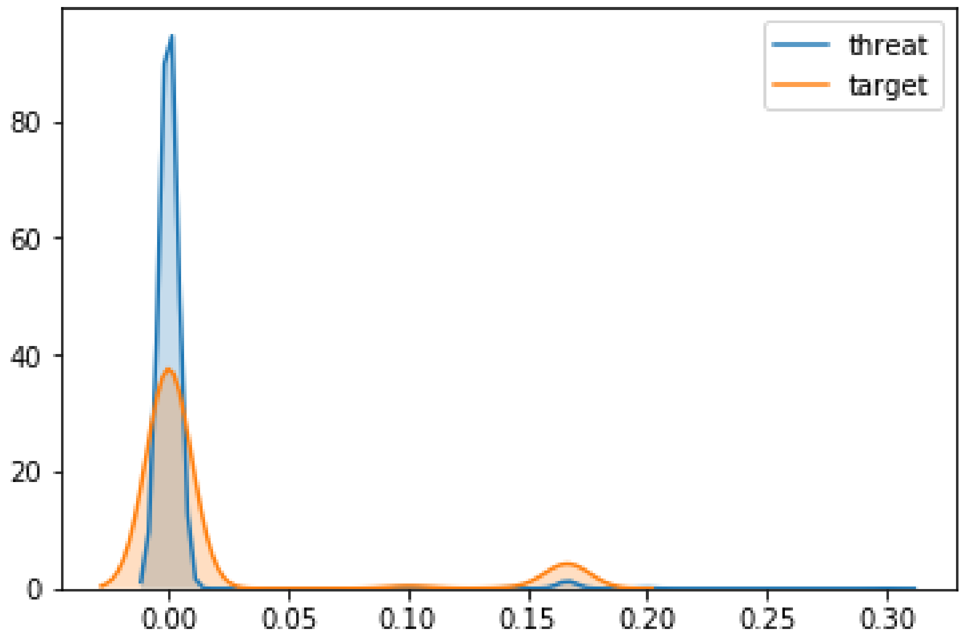
[125]:
```
import seaborn as sns

for col in ['new_feature','target']:
    z_ = sns.kdeplot(df_test_sample[col], shade=True)
```



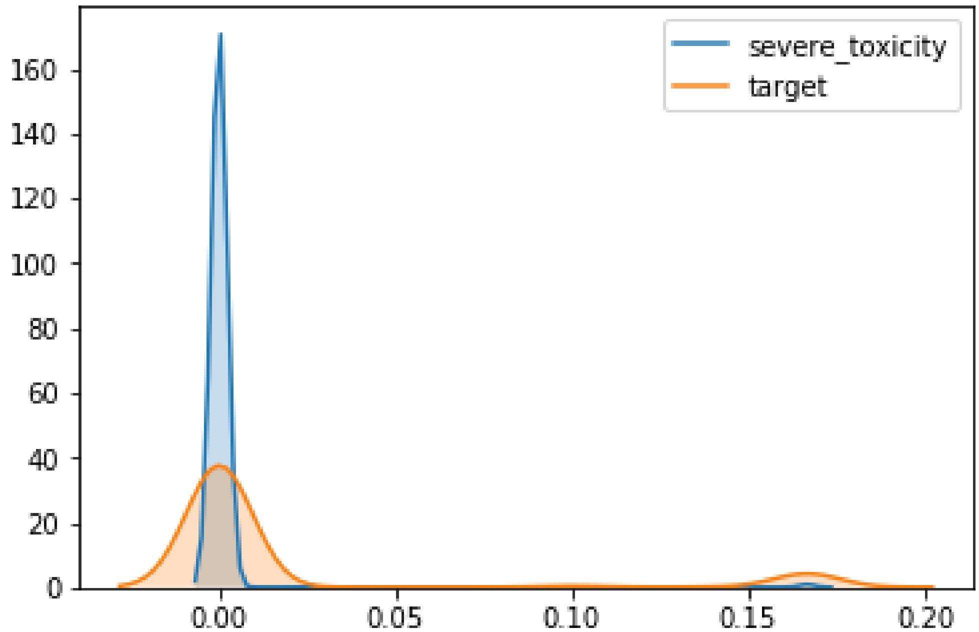
[79]:
```
import seaborn as sns

for col in ['insult','target']:
    z_ = sns.kdeplot(df_test_sample_filtered[col], shade=True)
```



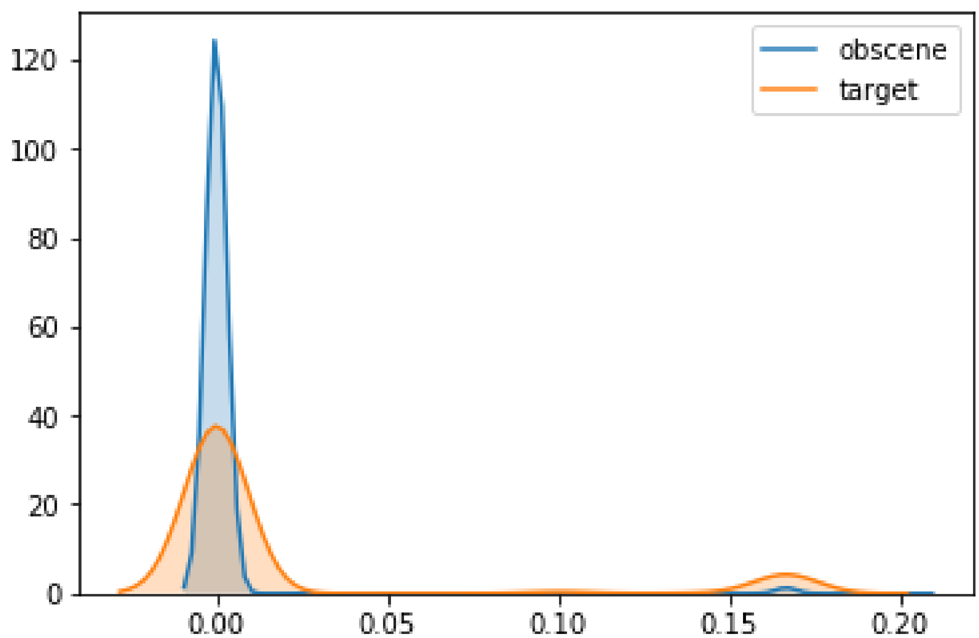
[430]:
```
import seaborn as sns

for col in ['threat','target']:
    z_ = sns.kdeplot(df_test_sample_filtered[col], shade=True)
```



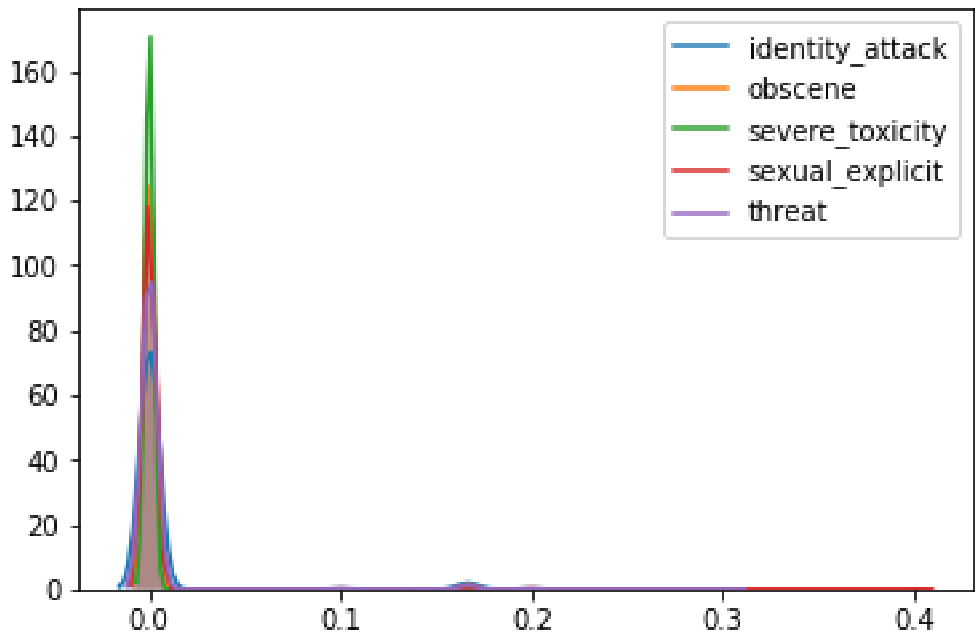
[431]:
```
import seaborn as sns

for col in ['severe_toxicity','target']:
    z_ = sns.kdeplot(df_test_sample_filtered[col], shade=True)
```



[432]:
```
import seaborn as sns

for col in ['obscene','target']:
    z_ = sns.kdeplot(df_test_sample_filtered[col], shade=True)
```



[433]:
```
import seaborn as sns

for col in ['identity_attack','obscene',␣
↪'severe_toxicity','sexual_explicit','threat']:
    z_ = sns.kdeplot(df_test_sample_filtered[col], shade=True)
```

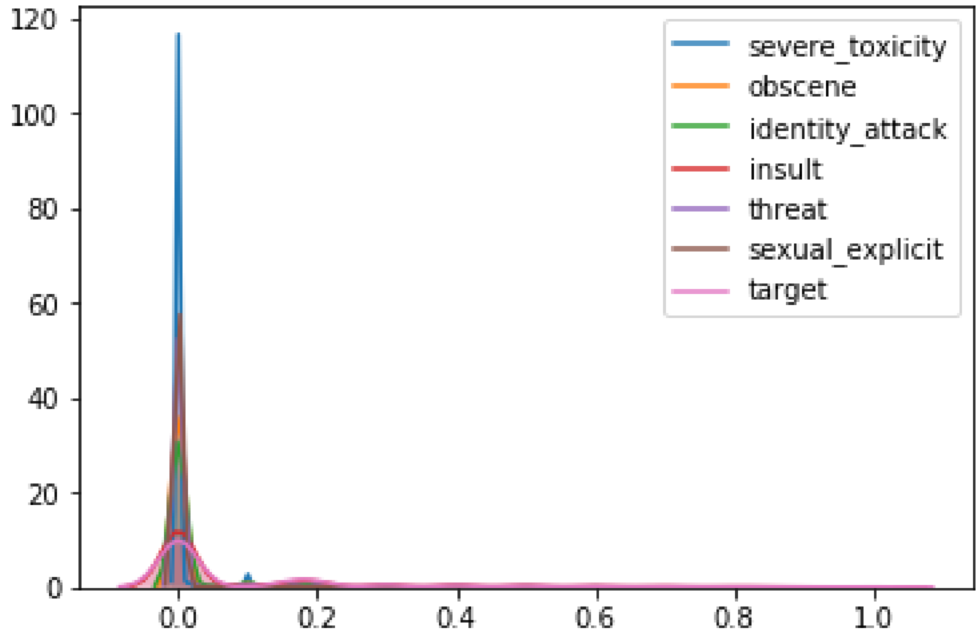


[436]:
```
from scipy.stats import kstest
col='insult'
kstest(df_test_sample_filtered[col], 'norm')
```

[436]:
```
KstestResult(statistic=0.5, pvalue=0.0)
```

3.2.3 Distribution of `target` from train dataset

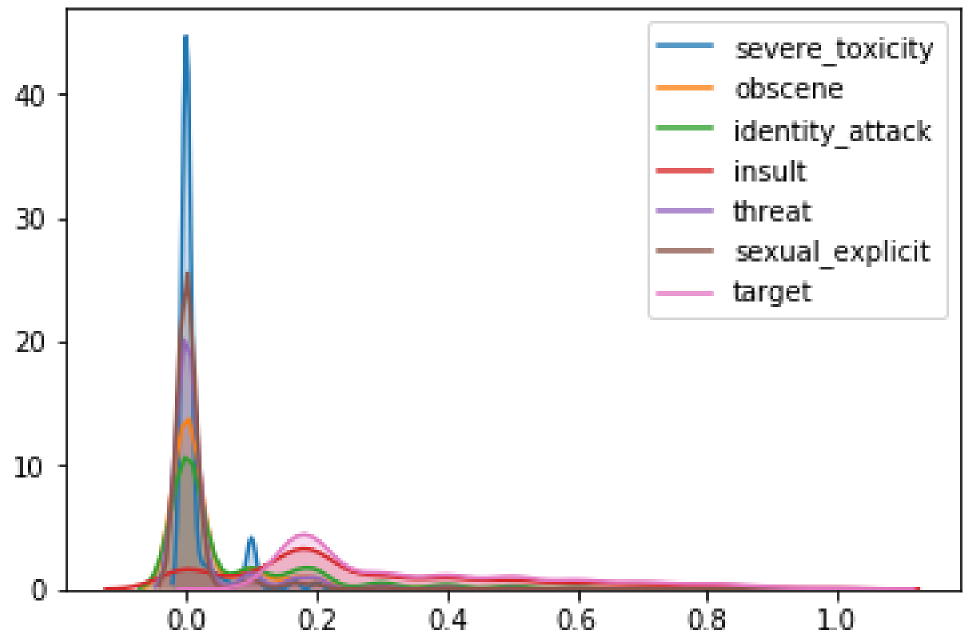
[439]:
```
import seaborn as sns

for col in df_train_sample.columns:
    sns.kdeplot(df_train_sample[col], shade=True)
```



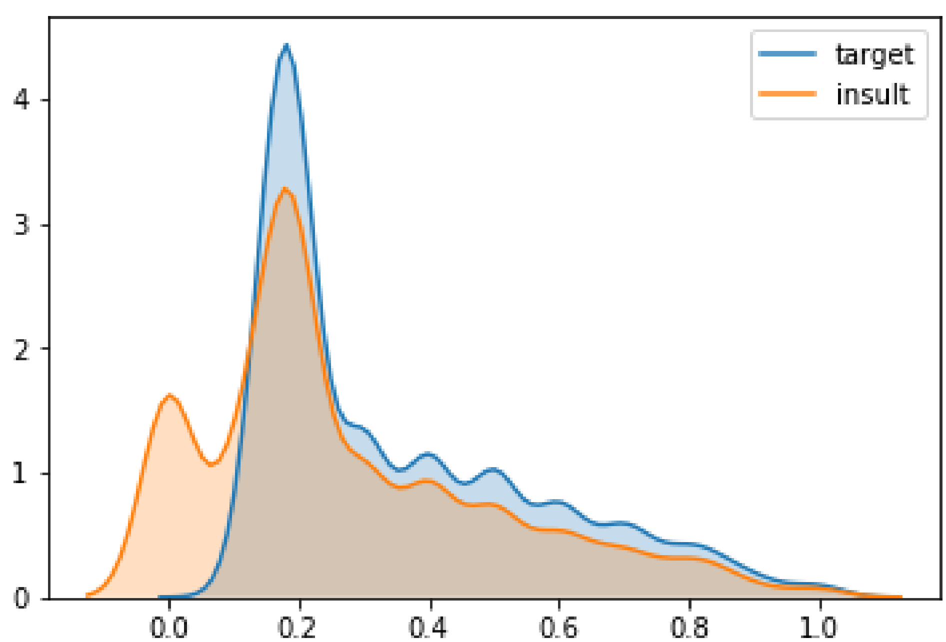
[455]:
```
if 'target' not in df_train_sample.columns:
    df_train_sample['target'] = df_train_label

df_train_sample_filtered = df_train_sample.query('target>0.1')
df_train_sample_filtered.shape, df_train_sample.shape
```

[455]: ((5296, 7), (18049, 7))

[456]:
```
import seaborn as sns

for col in df_train_sample_filtered.columns:
    sns.kdeplot(df_train_sample_filtered[col], shade=True)
```



[457]:
```
import seaborn as sns

for col in ['target','insult'] :
    sns.kdeplot(df_train_sample_filtered[col], shade=True)
```



We can conclude that insult contributes the most to a toxic comment

[121]:
```
df_test_sample.shape, df_train_sample.shape
```

[121]:
```
((6016, 20), (18049, 8))
```

[ ]:
```

```