MAGIC Gamma Telescope

Mona Kashani

Contents

1. Introduction:

Gamma rays (the highest energy forms of light) give astronomers insight into some of the most extreme events in our universe. Given their high energies, gamma rays are not produced as part of normal thermal processes, e.g., stellar fusion, because the temperatures required would be much too high. Rather, gamma rays are produced in non-thermal processes such as black hole accretion and shock waves created by stellar explosions (also known as supernovae).

Gamma rays lie well beyond the range of light that is visible to our human eyes, and their detection requires substantial efforts of science and engineering. Gamma-ray telescopes rely upon measurements of interaction products created when gamma rays pass through another medium. For ground-based telescopes such as the one providing data for this capstone, that medium is Earth's atmosphere, whereas for space-based telescopes, heavy metals are used to induce electron-positron pair creation (beyond the scope of this capstone).

The detection of gamma rays requires the filtering of noise from other high energy, charged, particles, usually protons and electrons (also known as cosmic rays). Cosmic rays produce similar, but not exactly the same, characteristics within a gamma-ray detector; however, cosmic rays originate from different sources and confuse the signal from gamma-ray sources. This cosmic-ray confusion makes gamma-ray sources appear brighter than they truly are if the cosmic-ray signal is not properly removed.

MAGIC (Major Atmospheric Gamma Imaging Cherenkov Telescope), located in Spain, is designed to record radiations released by Gamma rays in space. High Energy gamma rays initiate electromagnetic showers and charged particles are produced inside this shower which emits radiation. These radiations that leak through our atmosphere are detected and recorded by the MAGIC telescope.

In addition to the Gamma-ray radiation, radiation from hadronic (background) showers created by cosmic rays also gets recorded in the MAGIC telescope. These radiations from hadronic showers have a wavelength comparable to gamma-ray radiation. So, scientists are having a hard time trying to discriminate signals generated by gamma rays from those generated by unintended hadronic rays.

Solution:

Radiation recorded in Telescope has certain dimensional attributes. These attributes show subtle differences for Gamma Rays and Hadronic rays. On exploring these measurements, radiation emitted by gamma rays can be discriminated from the other.

Goal:

As a data scientist, goal is to accurately classify the radiations emitted by Gamma rays based on the numerical data recorded by the Telescope. This will help the Physicists to work on the right radiation measurements and avoid unnecessary radiation measurements.

Approach:

Implementation Machine learning algorithms on a magic dataset

Dataset Source:

https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope

Data Attributes:

This dataset has 19020 records with 11 attributes. Below is the description of attributes. 12,332 of which are gamma-ray events and 6,688 of which are cosmic-ray events. The target is imbalanced with 65% of records being gamma rays. All feature columns are continuous. Ten features and one target column are included in the data provided at the UCI hosting site.

1) **fLength:** continuous # major axis of ellipse [mm]

2) **fWidth:** continuous # minor axis of ellipse [mm]

3) **fSize:** continuous # 10-log of sum of content of all pixels [in #phot]

4) **fConc:** continuous # ratio of sum of two highest pixels over fSize [ratio]

5) **fConc1:** continuous # ratio of highest pixel over fSize [ratio]

6) **fAsym:** continuous # distance from highest pixel to center, projected onto major axis [mm]

7) **fM3Long:** continuous # 3rd root of third moment along major axis [mm]

8) **fM3Trans:** continuous # 3rd root of third moment along minor axis [mm]

9) **fAlpha:** continuous # angle of major axis with vector to origin [deg]

10) **fDist:** continuous # distance from origin to center of ellipse [mm]

11) **class:** g,h # gamma (signal), hadron (background)

2. Data:

Download data:

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data
```

Load data:

In this table you can observe the part of the data.

```
import pandas as pd
```

```
names=[ 'fLength', 'fWidth', 'fSize', 'fConc', 'fConc1',
       'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
```

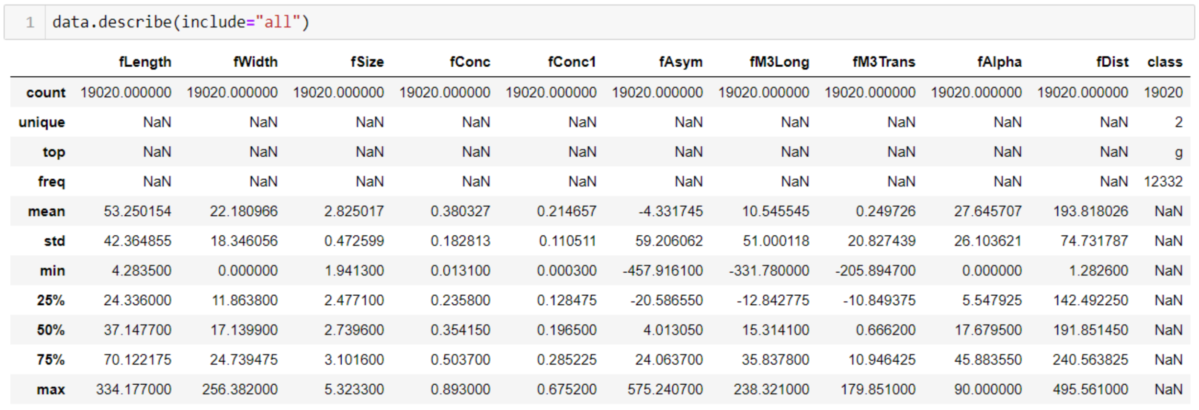
```
data = pd.read_csv('magic04.data', names=names)
data
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	g
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	g
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	g
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	g
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	g
...	...	...	...	...	...	...	...	...	...	...	...
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258	h
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560	h
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166	h
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166	h
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174	h

19020 rows × 11 columns

2.1. Data statistics:

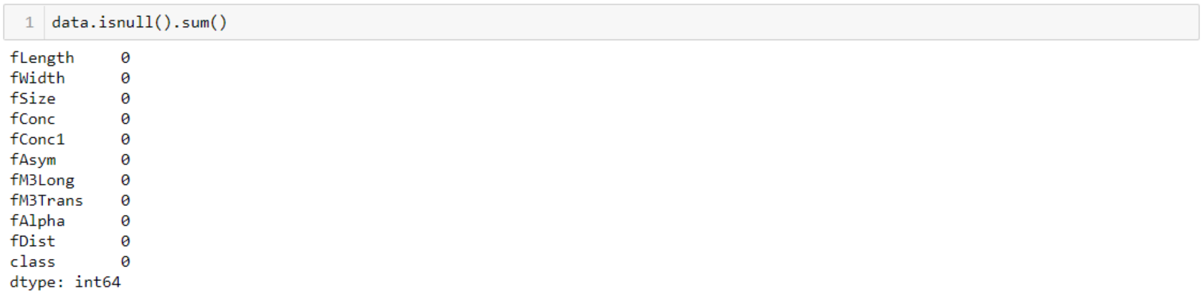
These data do not have missing value so there is no need to exchanging the data:



```
data.describe(include="all")
```

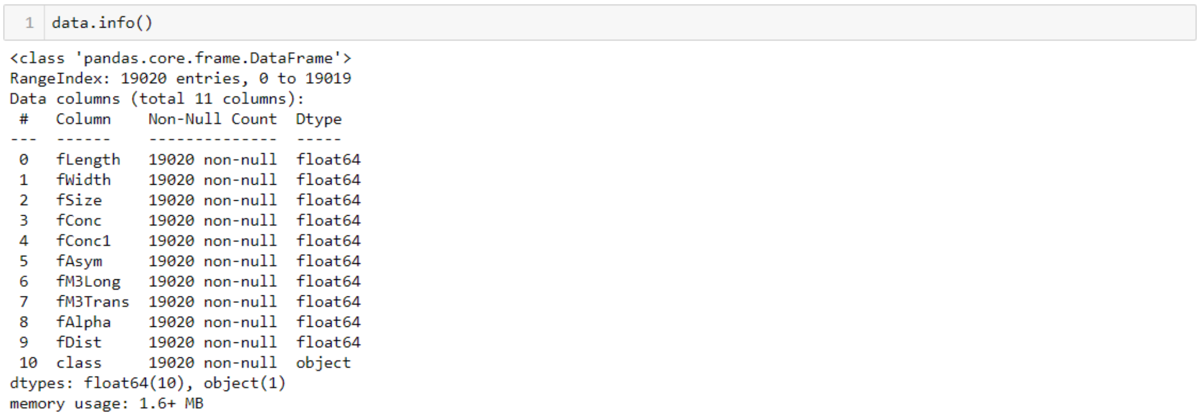
	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
count	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	g
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12332
mean	53.250154	22.180966	2.825017	0.380327	0.214657	-4.331745	10.545545	0.249726	27.645707	193.818026	NaN
std	42.364855	18.346056	0.472599	0.182813	0.110511	59.206062	51.000118	20.827439	26.103621	74.731787	NaN
min	4.283500	0.000000	1.941300	0.013100	0.000300	-457.916100	-331.780000	-205.894700	0.000000	1.282600	NaN
25%	24.336000	11.863800	2.477100	0.235800	0.128475	-20.586550	-12.842775	-10.849375	5.547925	142.492250	NaN
50%	37.147700	17.139900	2.739600	0.354150	0.196500	4.013050	15.314100	0.666200	17.679500	191.851450	NaN
75%	70.122175	24.739475	3.101600	0.503700	0.285225	24.063700	35.837800	10.946425	45.883550	240.563825	NaN
max	334.177000	256.382000	5.323300	0.893000	0.675200	575.240700	238.321000	179.851000	90.000000	495.561000	NaN

The information and types of data and the number of data in each column which are followed Float64 data type.



```
data.isnull().sum()
```

```
fLength     0
fWidth      0
fSize       0
fConc       0
fConc1      0
fAsym       0
fM3Long     0
fM3Trans    0
fAlpha      0
fDist       0
class       0
dtype: int64
```



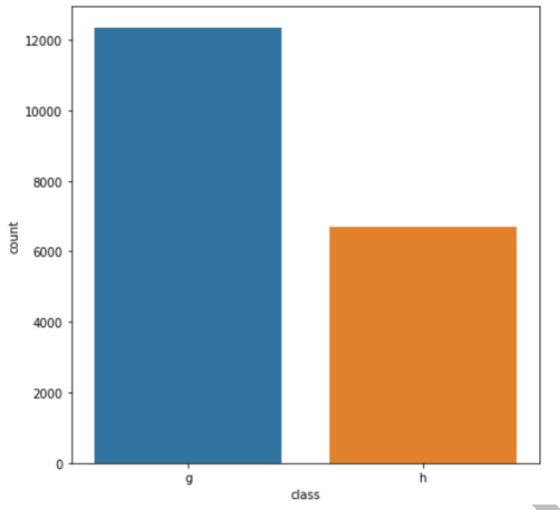
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19020 entries, 0 to 19019
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   fLength   19020 non-null  float64
 1   fWidth    19020 non-null  float64
 2   fSize     19020 non-null  float64
 3   fConc     19020 non-null  float64
 4   fConc1    19020 non-null  float64
 5   fAsym     19020 non-null  float64
 6   fM3Long   19020 non-null  float64
 7   fM3Trans  19020 non-null  float64
 8   fAlpha    19020 non-null  float64
 9   fDist     19020 non-null  float64
 10  class     19020 non-null  object 
dtypes: float64(10), object(1)
memory usage: 1.6+ MB
```

Below, by using ‘matplotlib’ library, I show the number of data in each label:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(7,7))
sns.countplot(data['class'])
plt.show()
```

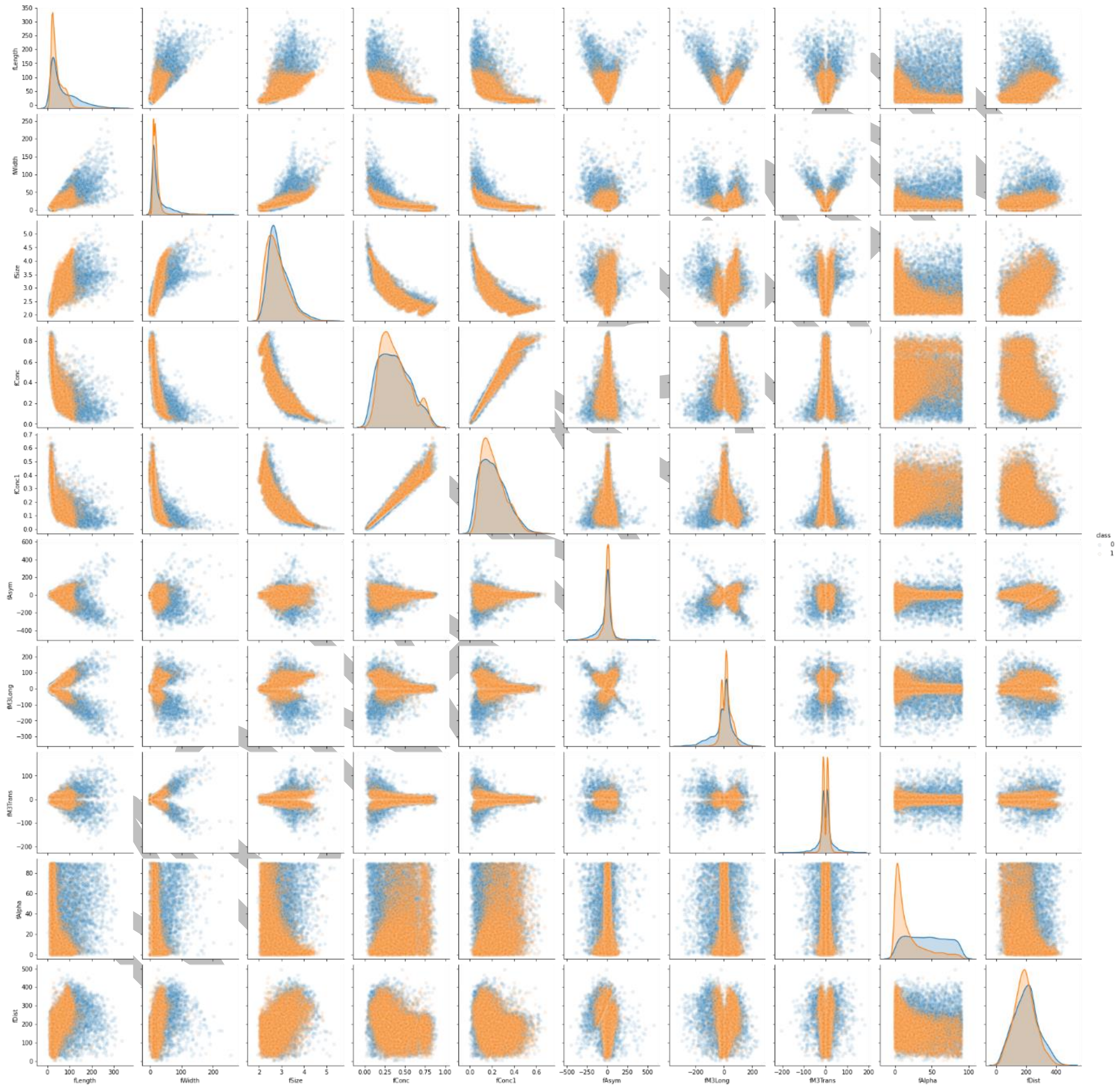


In the following, I converted the class type from ‘string’ to ‘integer, ‘h label’ will be zero, And I assign a value of 1 to ‘g label’.

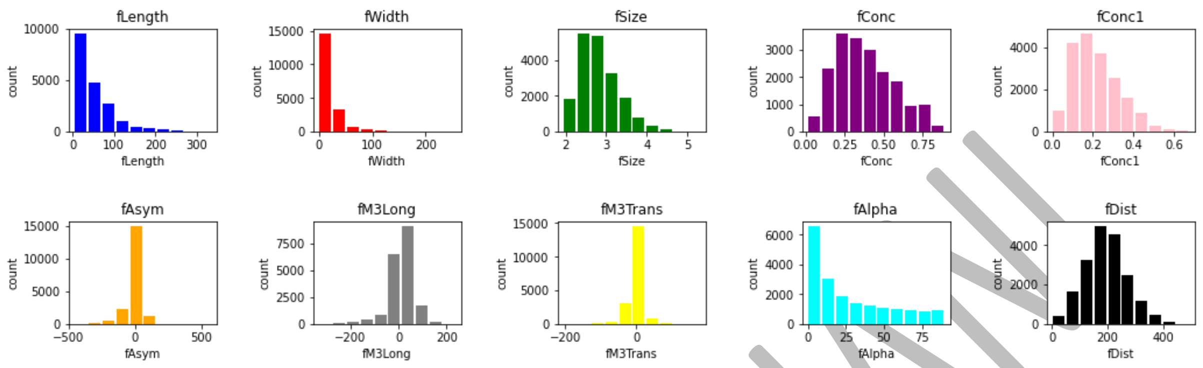
```
data['class'] = [0 if i=='h' else 1 for i in data['class']]
```

Bellow I plotted the figure of ‘pair plot’, which considers the features side by side and the data distribution in the classes shown between these two features.

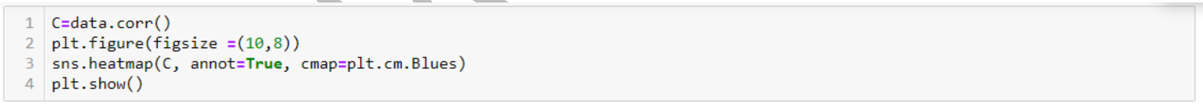
```
plt.figure(figsize=(17,17))
sns.pairplot(data=data, hue='class', plot_kws={'alpha': 0.1})
plt.show()
```



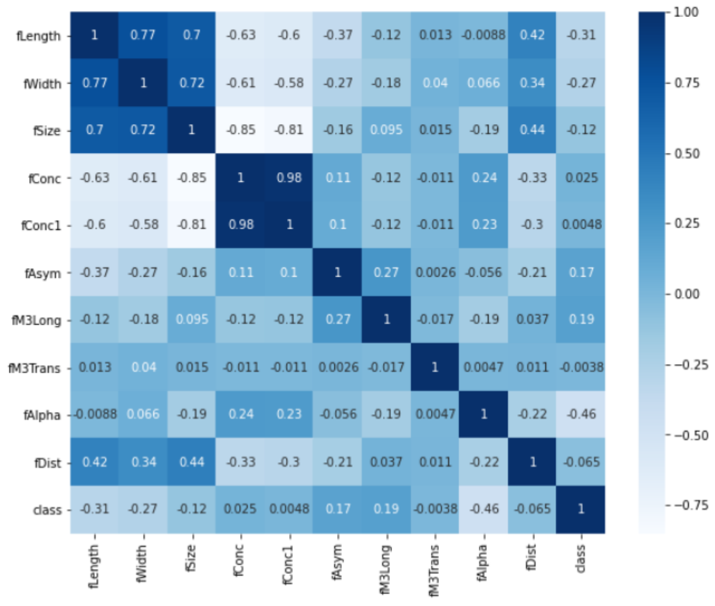
I have plotted the frequency of each column of attributes in each interval in the following diagrams.



Correlation among features represent which correlation between features are not equal. And the highest correlation between 'fconc' and 'fconc1' is 0.98.



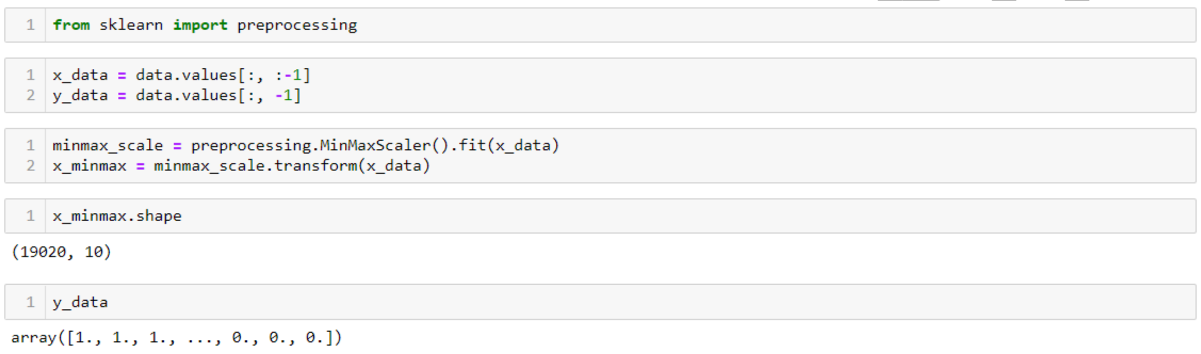
```
C=data.corr()
plt.figure(figsize =(10,8))
sns.heatmap(C, annot=True, cmap=plt.cm.Blues)
plt.show()
```



	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
fLength	1	0.77	0.7	-0.63	-0.6	-0.37	-0.12	0.013	-0.0088	0.42	-0.31
fWidth	0.77	1	0.72	-0.61	-0.58	-0.27	-0.18	0.04	0.066	0.34	-0.27
fSize	0.7	0.72	1	-0.85	-0.81	-0.16	0.095	0.015	-0.19	0.44	-0.12
fConc	-0.63	-0.61	-0.85	1	0.98	0.11	-0.12	-0.011	0.24	-0.33	0.025
fConc1	-0.6	-0.58	-0.81	0.98	1	0.1	-0.12	-0.011	0.23	-0.3	0.0048
fAsym	-0.37	-0.27	-0.16	0.11	0.1	1	0.27	0.0026	-0.056	-0.21	0.17
fM3Long	-0.12	-0.18	0.095	-0.12	-0.12	0.27	1	-0.017	-0.19	0.037	0.19
fM3Trans	0.013	0.04	0.015	-0.011	-0.011	0.0026	-0.017	1	0.0047	0.011	-0.0038
fAlpha	-0.0088	0.066	-0.19	0.24	0.23	-0.056	-0.19	0.0047	1	-0.22	-0.46
fDist	0.42	0.34	0.44	-0.33	-0.3	-0.21	0.037	0.011	-0.22	1	-0.065
class	-0.31	-0.27	-0.12	0.025	0.0048	0.17	0.19	-0.0038	-0.46	-0.065	1

2.2. Data Normalization:

Data normalization is used to eliminate different scales in features and the reason is to preventing more impact of a feature on training models. Here I used ‘minmax_scale’ to normalize and commensurability of them and as the result I set the scale of all features between 0 and 1.



```python
from sklearn import preprocessing
```

```python
x_data = data.values[:, :-1]
y_data = data.values[:, -1]
```

```python
minmax_scale = preprocessing.MinMaxScaler().fit(x_data)
x_minmax = minmax_scale.transform(x_data)
```

```python
x_minmax.shape
```

```
(19020, 10)
```

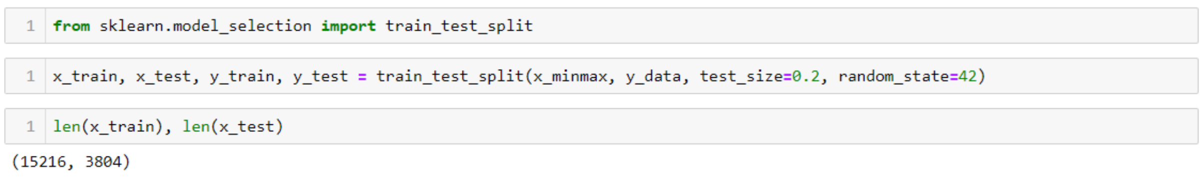
```python
y_data
```

```
array([1., 1., 1., ..., 0., 0., 0.])
```

2.3. Splitting Data

Divide the data into two parts ‘train’ and ‘test’ (which 20% of the data is for testing and the rest is for train).

Finally, the number of test data is 3804 and the train data has 15216 data.



```python
from sklearn.model_selection import train_test_split
```

```python
x_train, x_test, y_train, y_test = train_test_split(x_minmax, y_data, test_size=0.2, random_state=42)
```

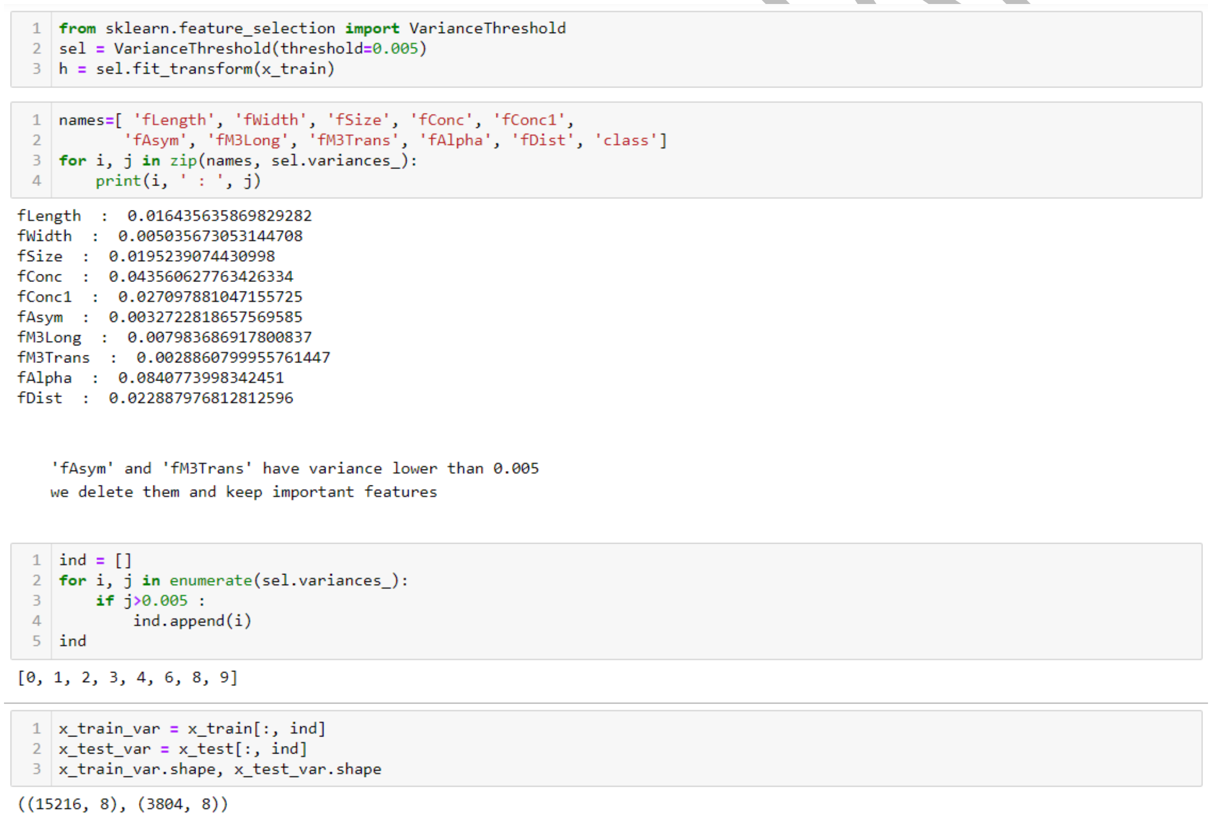
```python
len(x_train), len(x_test)
```

```
(15216, 3804)
```

3. Feature Engineering

For feature engineering, I use ‘variance’ and delete the columns which have less variance. For each column, the variance value is calculated and the columns whose variance value is less than 0.005 will be removed from test set and train set. (This parameter has been tested with different values which gives the best result 0f of 0.005 by deleting two columns).

And the result shows that the ‘fAsym’ and ‘fM3Trans’ are deleted.



```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=0.005)
h = sel.fit_transform(x_train)
```

```
names=[ 'fLength', 'fWidth', 'fSize', 'fConc', 'fConc1',
       'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
for i, j in zip(names, sel.variances_):
    print(i, ' : ', j)
```

```
fLength  :  0.016435635869829282
fWidth  :  0.005035673053144708
fSize  :  0.0195239074430998
fConc  :  0.043560627763426334
fConc1  :  0.027097881047155725
fAsym  :  0.0032722818657569585
fM3Long  :  0.007983686917800837
fM3Trans  :  0.0028860799955761447
fAlpha  :  0.0840773998342451
fDist  :  0.022887976812812596
```

'fAsym' and 'fM3Trans' have variance lower than 0.005
we delete them and keep important features

```
ind = []
for i, j in enumerate(sel.variances_):
    if j>0.005 :
        ind.append(i)
ind
```

```
[0, 1, 2, 3, 4, 6, 8, 9]
```

```
x_train_var = x_train[:, ind]
x_test_var = x_test[:, ind]
x_train_var.shape, x_test_var.shape
```

```
((15216, 8), (3804, 8))
```

4. Model Implementation

In the model implementation part:

Results will be going to report with ‘feature selection’ and without ‘feature engineering’.

GaussianNB:

Naïvebayes can be considered a model based on conditional probability. Suppose x = (x_1,...x_n) Express the vector of n attributes to forums are independent variables. In this way the probability of occurrence of C_k isp(C_k|x_1,...x_n) can be considered as one of the States. The Class show different events in different ways as different k which you can see below:

$$p(C_k \mid X) = \frac{p(C_k)\, p(X \mid C_k)}{p(X)}$$

as can be seen the above relation is the same as the bias theorem. Is a reminder of the boys centurion based on the probability of maximum event (Prior, posterior, likelihood and evidence) you can see that in this formula:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Thus, to calculate the probability p(C_k|x_1,...x_n) it is enough to get help from joint probability and with help of simplify the conditional probability according to the Independence of the variables.

$\mathrm{p}(C_k, x_1, \dots, x_n) = \mathrm{p}(x_1, \dots, x_n, C_k)$

$= \mathrm{p}(x_1 | x_2, \dots, x_n, C_k)\ \mathrm{p}(x_2, \dots, x_n, C_k)$

$= \mathrm{p}(x_1 | x_2, \dots, x_n, C_k)\mathrm{p}(x_2 | x_3, \dots, x_n, C_k)\mathrm{p}(x_3, \dots, x_n, C_k)$

$= \dots$

$= \mathrm{p}(x_1 | x_2, \dots, x_n, C_k)\ \mathrm{p}(x_2 | x_3, \dots, x_n, C_k)\ \dots\ \mathrm{p}(x_{n-1} | \ x_n, C_k)\mathrm{p}(x_n | C_k)$

Assuming the independence of components or some of different x_i features which we can write the probabilities simpler. As this equation:

$$\mathrm{p}(x_i|x_{i+1}, \dots, x_n, C_k) \approx \mathrm{p}(x_i|C_k)$$

Therefore, we can write the probability as multiplication of the probability.

$$\mathrm{p}(C_k|x_1, \dots, x_n) \infty \, \mathrm{p}(C_k, x_1, \dots, x_n)$$
$$\approx \mathrm{p}(C_k)\, p(x_1|\, C_k)\;\; p(x_2|\, C_k)\, p(x_3|\, C_k) \dots$$

$$= \; p(C_k) \prod_{i=1}^{n} p(x_i|\, C_k)$$

Note: in the first equation the denominator in same calculations and it is stable so the conditional probability can be proportional to probably considered together. In above equation, we have shown this proportion with ∞.

Considering all above together, we can determine the conditional probability of equation 2 introduced in equation 1. As a result, it is possible that an observation in accordance of X with the following specific relationship will be like this.

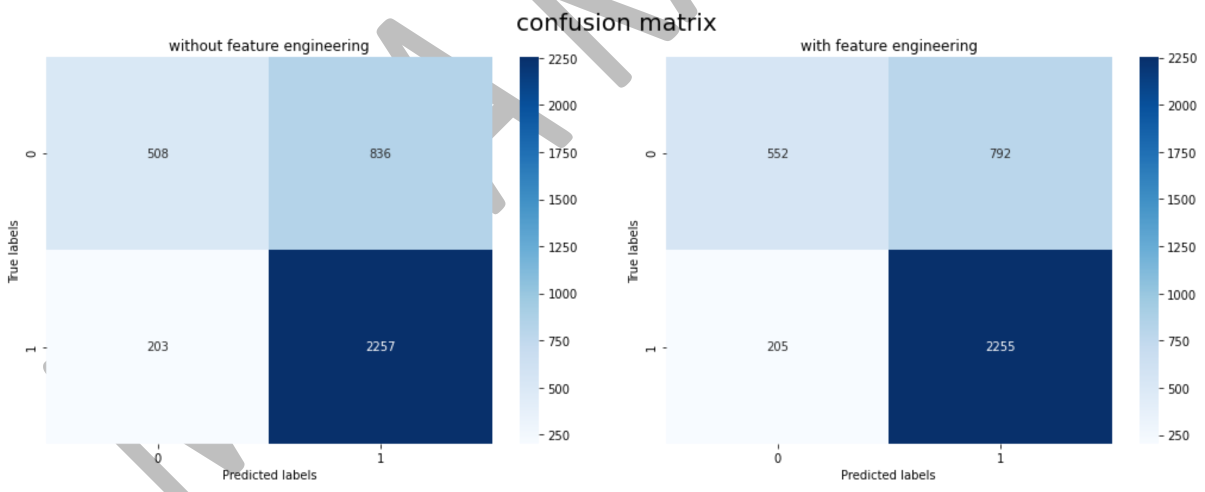
$$p(C_K|x_1, \dots, x_n) = \frac{1}{Z}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Note that here the probability of evidence observations in the form of

Z = p(x)= $\sum_k p(C_k)\, p(x|C_k)$ in consideration has been taken. It is clear that Z is dependent on evidence and observations and these observation $x_1, \dots, x_n$.

```python
from sklearn.naive_bayes import GaussianNB
clf1 = GaussianNB()
clf1.fit(x_train, y_train)
pred_test1 = clf1.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test1)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test1))

clf1 = GaussianNB()
clf1.fit(x_train_var, y_train)
pred_test1_2 = clf1.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test1_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test1_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
```

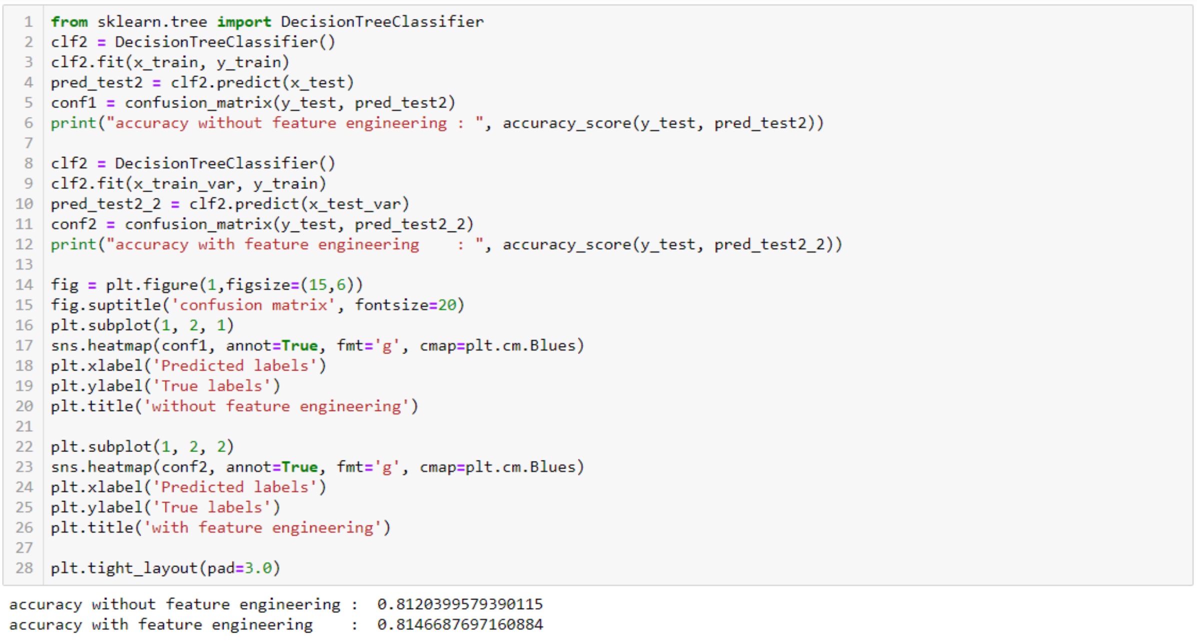
```
accuracy without feature engineering :  0.7268664563617245
accuracy with feature engineering    :  0.7379074658254469
```



<u>Decision Tree:</u>

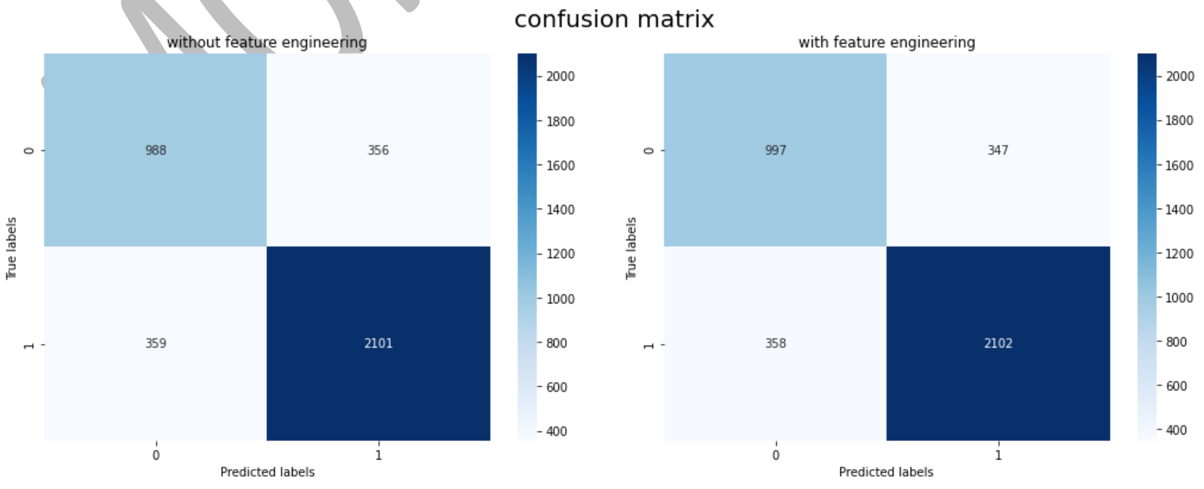
In this algorithm, the problem is to decide to determine the class of test data, the first training Dana in the form of one implement tree so while evaluating it, you can see in new class of data related to predicted.

In this case, by traversing the tree from the roots to the leaves, based on the conditions related to the branches, which can achieve to one of these two to classes h or g.



```python
from sklearn.tree import DecisionTreeClassifier
clf2 = DecisionTreeClassifier()
clf2.fit(x_train, y_train)
pred_test2 = clf2.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test2)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test2))

clf2 = DecisionTreeClassifier()
clf2.fit(x_train_var, y_train)
pred_test2_2 = clf2.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test2_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test2_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
```

```
accuracy without feature engineering :  0.8120399579390115
accuracy with feature engineering    :  0.8146687697160884
```

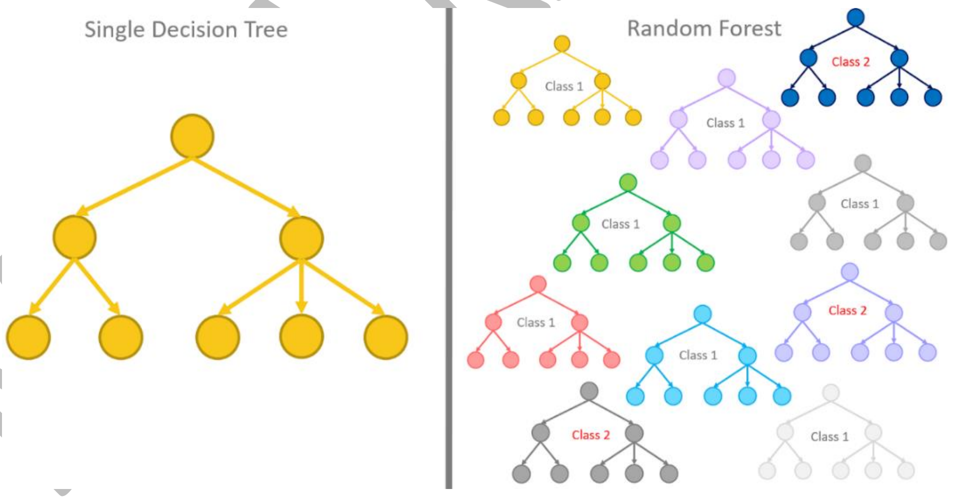


Random Forest:

Random Forest is an easy-to-use machine tracking algorithm that is often provide excellent results, even without adjusting its permeability parameters. This algorithm is due to the simplicity and usability, both for classification and regression. It is one of the most widely used machine learning algorithm.

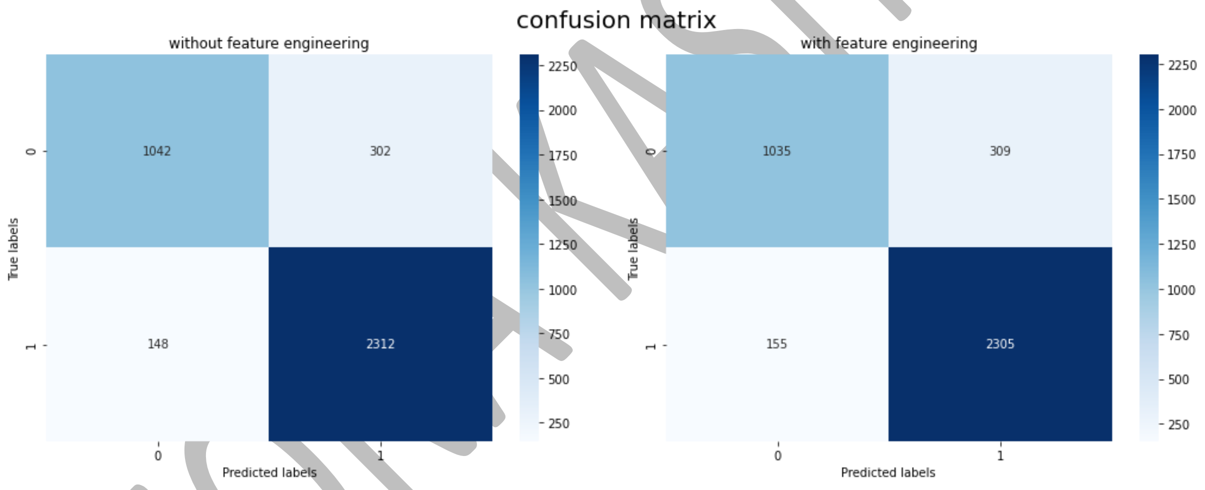
Random Forest is a supervised learning algorithm. As the name implies, this algorithm randomly generates the forest algorithm, Forest is made, in fact, a group of decision trees. Building a forest using trees is often done with bagging method.

In General result of the model simply put a random forest of several trees besides and merge them together. It allows for more accurate and stable predictions. In the difference between decision tree and Forest algorithm accidentally illustrate in summary form:



```python
from sklearn.ensemble import RandomForestClassifier
clf3 = RandomForestClassifier()
clf3.fit(x_train, y_train)
pred_test3 = clf3.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test3)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test3))

clf3 = RandomForestClassifier()
clf3.fit(x_train_var, y_train)
pred_test3_2 = clf3.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test3_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test3_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
```

```
accuracy without feature engineering :  0.8817034700315457
accuracy with feature engineering    :  0.8780231335436383
```



Logistic Regression:

One of the methods of classification in the topic of supervised learning is logistic regression. In regression method the concept and method of calculation odds ratio is used this is done in 3 Steps:

The first step:

in this system the probability of Y in the regression formula instead of Y calculated as a result parameter: which is entered. In this case the left side of the regression equation is the number between 0 and 1.

$$P(Y = 1) = \pi, \quad P(Y = 0) = 1-\pi$$

The second Step:

involves using a function of probability of Y in the equation this function is called 'odds ratio'. At this point the left side of this regression equation is between 0 and positive Infinite.

$$\text{OddsRatio}\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \text{OR} = \frac{\pi}{1-\pi}$$

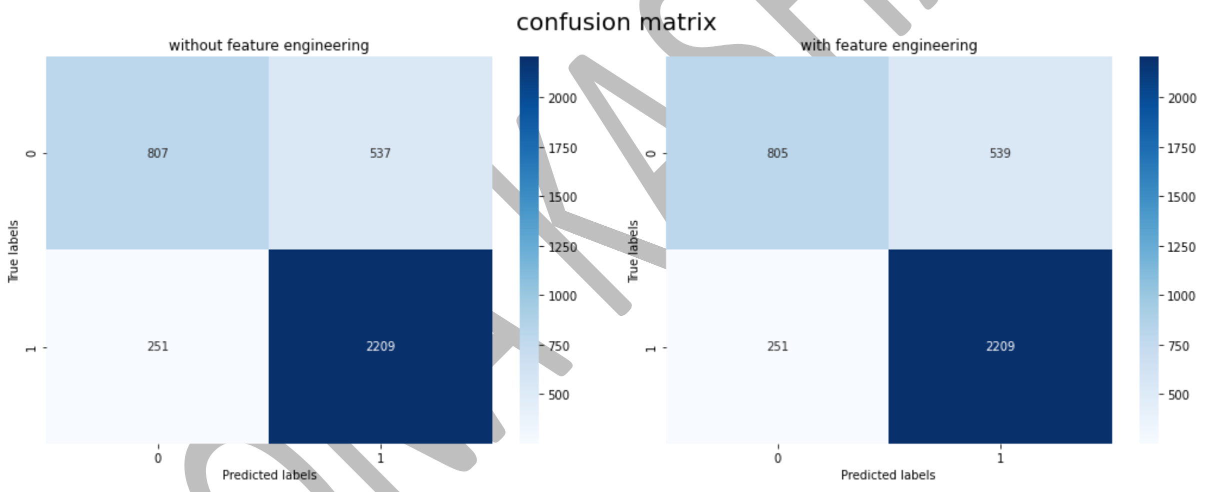
The third step:

it's involved of using logarithm as a result variable in the equation to face the problem the left part of the equation is between in minus Infinite to 0.

$$\text{Ln}(\text{OR}) = \text{Ln}\left(\frac{\pi}{1-\pi}\right)$$

```python
from sklearn.linear_model import LogisticRegression
clf4 = LogisticRegression()
clf4.fit(x_train, y_train)
pred_test4 = clf4.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test4)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test4))

clf4 = LogisticRegression()
clf4.fit(x_train_var, y_train)
pred_test4_2 = clf4.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test4_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test4_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
```

```
accuracy without feature engineering :  0.7928496319663512
accuracy with feature engineering    :  0.7923238696109358
```



KNN:

K Nearest Neighbor (KNN) used for both classification and regression problems. This algorithm while training the data shuffles in the n-dimensional space and placed them in different categories and during test time is determined by determining the distance between the test data and the centers of the categories. This distance can use Euclidean method.

Euclidean distance provides two examples of hypothetical data with n properties in the form bellow:

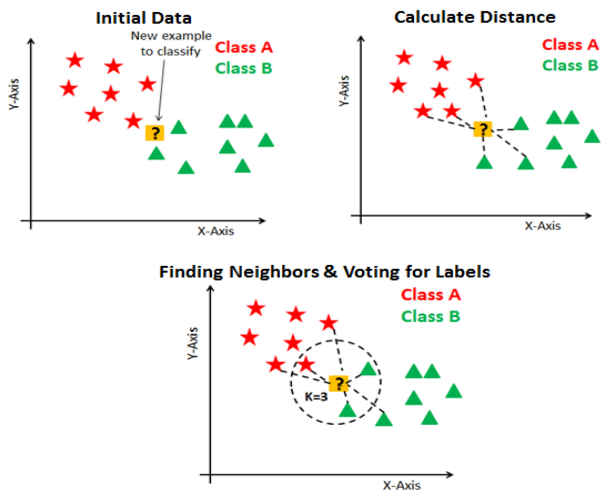
$$P = (p_1, p_2, \ldots, p_n)$$

$$q = (q_1, q_2, \ldots, q_n)$$

$$d(p,q)= \sqrt{(p_1 - q_1)^2 + (p_2 - \ q_2)^2 + \cdots + (p_n - q_n\ )^2} = \ \sqrt{\sum_{i=1}^{n} (p_i - q_i\ )^2}$$

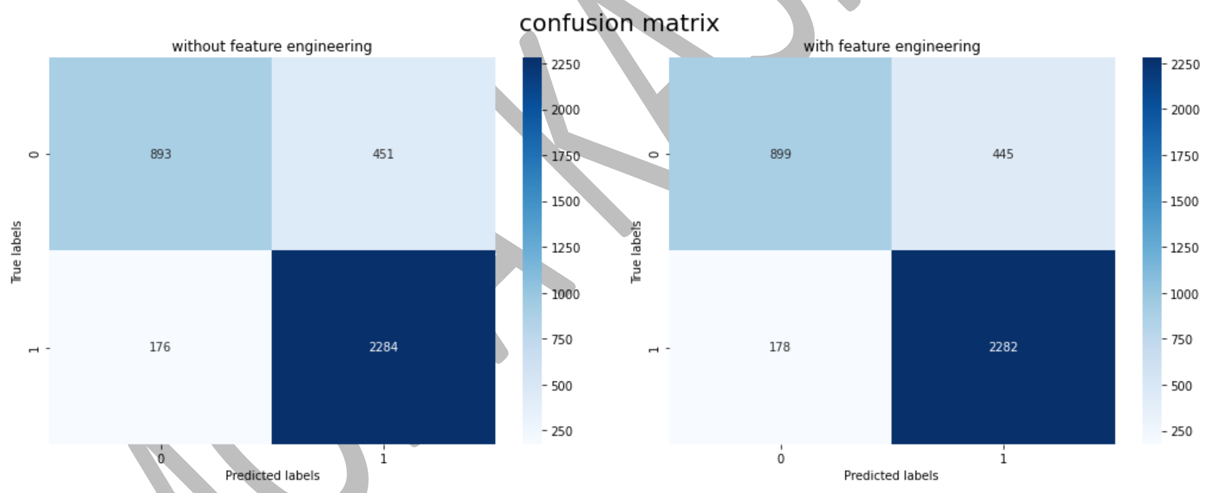
KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels



```python
from sklearn.neighbors import KNeighborsClassifier
clf5 = KNeighborsClassifier()
clf5.fit(x_train, y_train)
pred_test5 = clf5.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test5)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test5))

clf5 = KNeighborsClassifier()
clf5.fit(x_train_var, y_train)
pred_test5_2 = clf5.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test5_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test5_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
```

```
accuracy without feature engineering :  0.8351735015772871
accuracy with feature engineering    :  0.8362250262881178
```

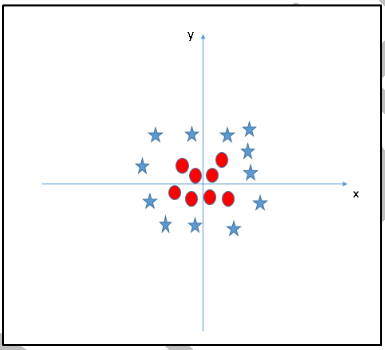


SVM(SVC):

SVM is used for both classification and regression problems. However mostly used in classification issues. in SVM each example draw data as a dot in the next n-dimensional space on the day the scatter chart. (n is the number of properties that has an example of data).

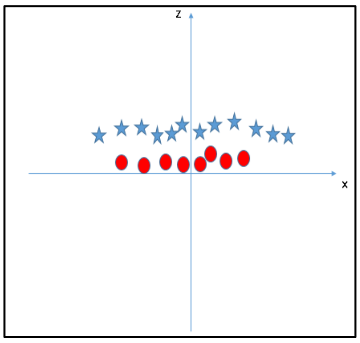
And the value of each criteria refers to data, is define coordinates of the point on the graph. Then, by drawing is straight line, different and distinct data will be categorized.

SVM can easily solve non-linear problems in cases where you cannot do it with two Straight lines, separate the class from One data, this method can be really useful. This is an example you can see:



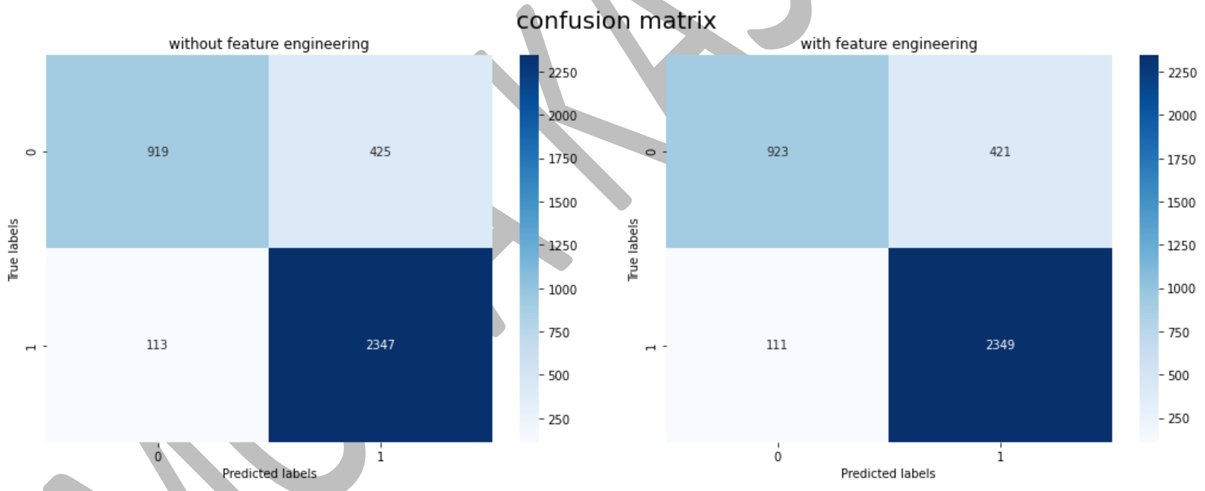
This algorithm changes such data by converting non-linear properties to Linear properties in in a way that divide the data into categories to using a straight line. In this particular example, apply the following data conversion into two dimensions x and z.

z=x^2+y^2



```python
from sklearn.svm import SVC
clf6 = SVC()
clf6.fit(x_train, y_train)
pred_test6 = clf6.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test6)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test6))

clf6 = SVC()
clf6.fit(x_train_var, y_train)
pred_test6_2 = clf6.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test6_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test6_2))

fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
plt.show()
```

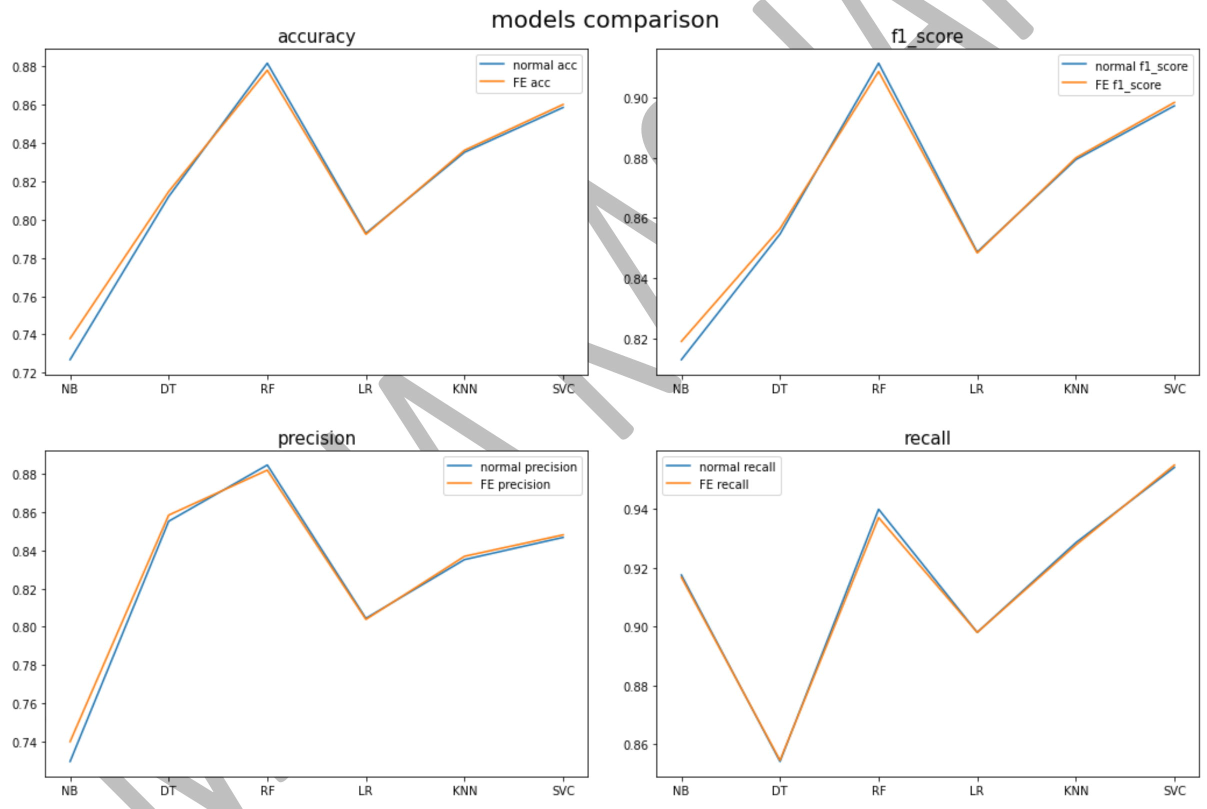
```
accuracy without feature engineering :  0.8585699263932702
accuracy with feature engineering    :  0.8601472134595163
```



5. Comparison

The following diagrams show the comparison of accuracy ‹f-measure ‹ precision and recall for 6 algorithms which has used in this paper in two models: ‘without feature engineering’ (blue color) and ‘with feature engineering’ (orange color).

And as we can see that in different models, except in ‘random forest’, the orange line is higher than the blue line which means how ‘feature engineering’ has a positive effect on the results of the models.



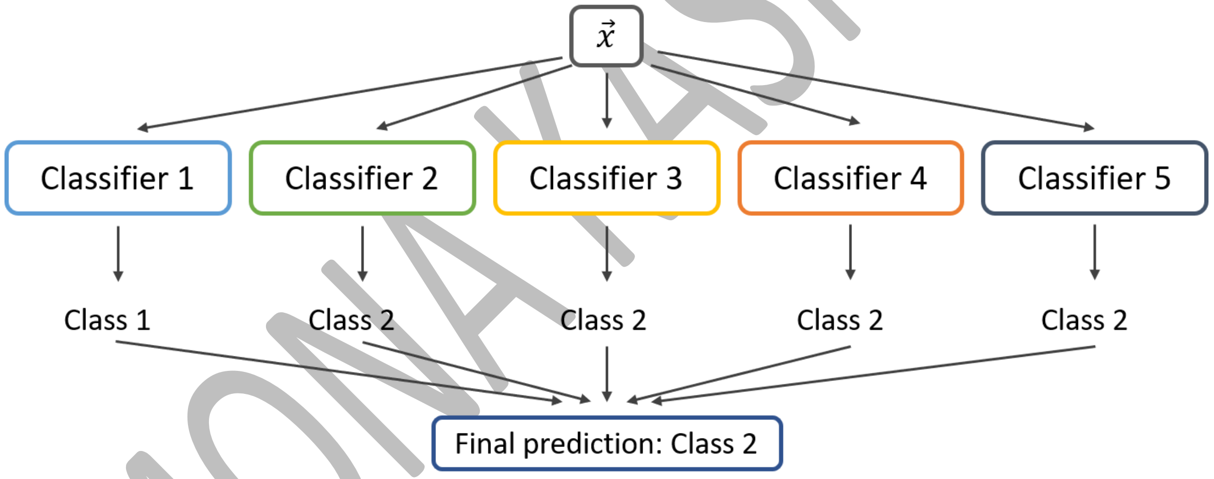
6. Ensemble learning

Combined categories actually use a combination of the results of several classifiers. In fact, each of these categories build their own model on the data and store it.

Finally, for the final classification, a vote is taken between these clauses, and the class that gets the most votes is the final class.

In the figure below, you can see the voting method which given class 2 that received the most votes from classifiers, finally the voting algorithm predicts this class as an example.

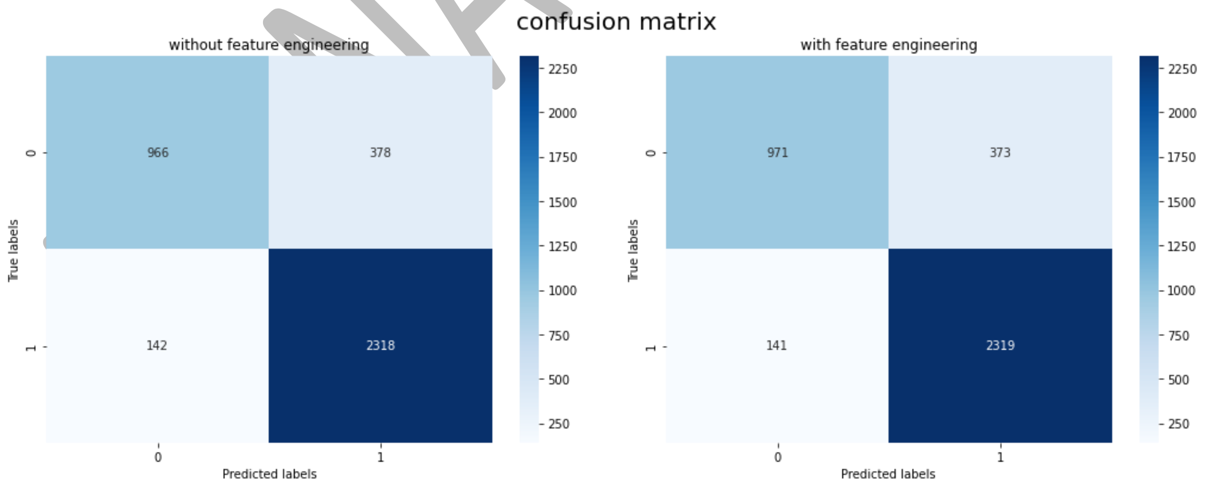
Ensemble in machine learning is the combination of the result of different algorithms. Here I combined the algorithms with 'VotingClassifier' module and the result was an ۸۶/۴۸٪ increasing in accuracy.



```python
from sklearn.ensemble import VotingClassifier

clf1 = GaussianNB()
clf2 = DecisionTreeClassifier()
clf3 = RandomForestClassifier()
clf4 = LogisticRegression()
clf5 = KNeighborsClassifier()
clf6 = SVC()

eclf = VotingClassifier(estimators=[('nb', clf1), ('dt', clf2), ('rf', clf3),
                                    ('lr', clf4), ('knn', clf5), ('svm', clf6)])
eclf.fit(x_train, y_train)
pred_test_vc = eclf.predict(x_test)
conf1 = confusion_matrix(y_test, pred_test_vc)
print("accuracy without feature engineering : ", accuracy_score(y_test, pred_test_vc))

clf1 = GaussianNB()
clf2 = DecisionTreeClassifier()
clf3 = RandomForestClassifier()
clf4 = LogisticRegression()
clf5 = KNeighborsClassifier()
clf6 = SVC()

eclf = VotingClassifier(estimators=[('nb', clf1), ('dt', clf2), ('rf', clf3),
                                    ('lr', clf4), ('knn', clf5), ('svm', clf6)])
eclf.fit(x_train_var, y_train)
pred_test_vc_2 = eclf.predict(x_test_var)
conf2 = confusion_matrix(y_test, pred_test_vc_2)
print("accuracy with feature engineering    : ", accuracy_score(y_test, pred_test_vc_2))

```

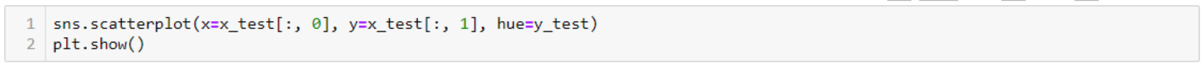
```python
fig = plt.figure(1,figsize=(15,6))
fig.suptitle('confusion matrix', fontsize=20)
plt.subplot(1, 2, 1)
sns.heatmap(conf1, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('without feature engineering')

plt.subplot(1, 2, 2)
sns.heatmap(conf2, annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('with feature engineering')

plt.tight_layout(pad=3.0)
plt.show()
```

```
accuracy without feature engineering :  0.8633017875920084
accuracy with feature engineering    :  0.8648790746582544
```

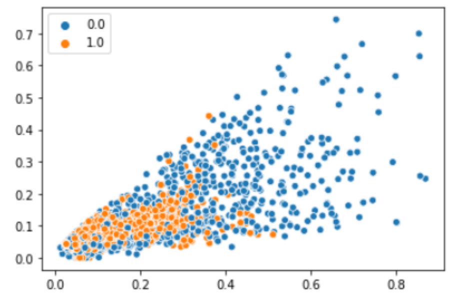


7. Visualization

To illustrate how each algorithm predicts test data classes, we plotted them. Here the test data is based on the values of the first column and the second column, with use of main labels of test data, then I plotted a scatter diagram and assign it to the zero label (h) in blue and one label (g) in orange. However, due to the scattering of data and the mixing of labels in this two-dimensional space for better display, use PCA to reduce the dimensions of the test data, which will disperse the data and make the display more appropriate.



```
sns.scatterplot(x=x_test[:, 0], y=x_test[:, 1], hue=y_test)
plt.show()
```



PCA:

The main goal of PCA analysis is how it can recognize patterns in the data, basic component analysis of PCA intends to converge identify between variables. If there is a strong correlation between the variables, efforts to reduce the dimensions will be significant. In general, what happens in PCA is to find the maximum direction of variance in high-dimensional data and its design to be sub-spatial with less dimensions so that most of the information is retained.

Often, the expected goal is to reduce the dimensions of a dimensional data set 'd' by redesigning it in a sub-space with k dimensions (k < d) so that an important part of information remains in increasing the computational efficiency.

To illustrate how each algorithm predicts test data classes, normally we use visualization. Here the test data is based on the values of the first column and the second column, using the main labels of test set, then I plotted that in a 'scatter' pattern and label0(h) is blue and label1(g) is orange. However, due to scattering and mixing of the labels in this two-dimensional space it is better to use 'PCA' to reduce the dimensions of the test data, which causes scattering data and displays it to become more appropriate.

As you can see in this figure after implementing 'PCA' and reduce the dimensions of data test in two columns, the scatter of the data is more convenient for illustration and better displays.

```
from sklearn.decomposition import PCA
pc=PCA(n_components=2)
data2=pc.fit_transform(x_test)
```

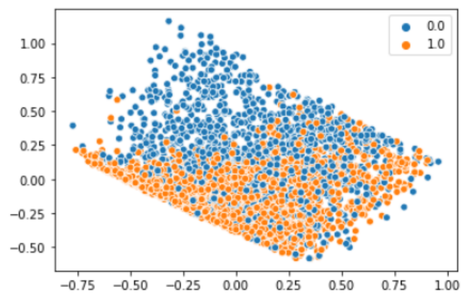
```
data2.shape
```

```
(3804, 2)
```

```
len(y_test), len(data2)
```

```
(3804, 3804)
```

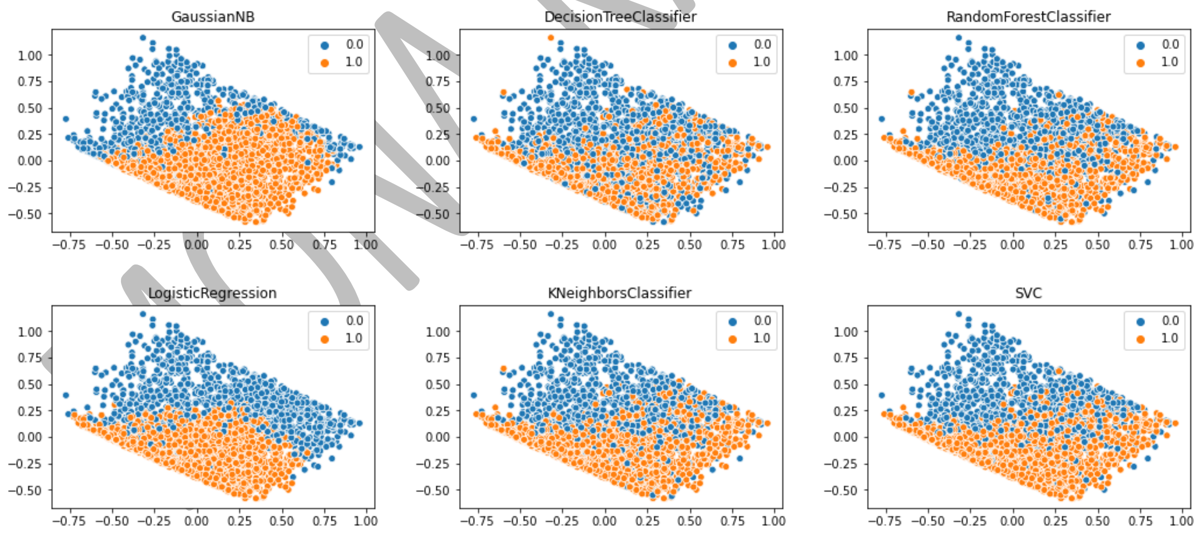
```
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=y_test)
plt.show()
```



Here are the test data that I used 'PCA' to reduce their dimensions to two columns based on 'predicted labels' which are illustrated by 6 algorithms used in this paper. (As I find an optimization result due to using 'feature engineering', Then in this section I plotted the results of these algorithms by using scatter plot)

Comparing these figures with the scatter plot above there are lots of similarities in main labels and good performance of these models:

```python
plt.figure(1,figsize=(15,7))

plt.subplot(2, 3, 1)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test1_2)
plt.title("GaussianNB")

plt.subplot(2, 3, 2)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test2_2)
plt.title("DecisionTreeClassifier")

plt.subplot(2, 3, 3)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test3_2)
plt.title("RandomForestClassifier")

plt.subplot(2, 3, 4)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test4_2)
plt.title("LogisticRegression")

plt.subplot(2, 3, 5)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test5_2)
plt.title("KNeighborsClassifier")

plt.subplot(2, 3, 6)
sns.scatterplot(x=data2[:, 0], y=data2[:, 1], hue=pred_test6_2)
plt.title("SVC")

plt.tight_layout(pad=3.0)
plt.show()
```



8. Conclusion:

According to the ‘pair plot’ diagram, we find that the data in this data set are intertwined and by plotting them using different features, we can see the intertwining and overlapping of data in all charts in pairs. In fact, after plotting the diagram, we were only able to see examples of one of the classes. By placing the alpha attribute in this chart equal to 0.1, we made the points a bit clearer (each representing a sample in the data) and also, we were able to display the data of the following samples and see the samples of both classes in this chart.

As previously reported, SVM with the use of data reshaping it can model two or more classes with simpler lines, and can appropriately separate between classes. This property is suitable for datasets with intertwined and complex samples with high overlap. As we can see this feature clearly in the results of this paper. With use of SVM algorithm, we were able to achieve an accuracy of 86.01. which is higher than the accuracy of Naive Bayes, Decision Tree, Random Forest, KNN, Logistic Regression.

We were also able to achieve 86.49 accuracy using the Ensemble Learning which indicates very good performance of the voting algorithm among other algorithms.

References

- Bock, R.K., Chilingarian, A., Gaug, M., Hakl, F., Hengstebeck, T., Jirina, M., Klaschka, J., Kotrc, E., Savicky, P., Towers, S., Vaicilius, A., Wittek W. (2004).
  Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope.
  Nucl.Instr.Meth. A, 516, pp. 511-528.

- P. Savicky, E. Kotrc.
  Experimental Study of Leaf Confidences for Random Forest.
  Proceedings of COMPSTAT 2004, In: Computational Statistics. (Ed.: Antoch J.) - Heidelberg, Physica Verlag 2004, pp. 1767-1774.

- J. Dvorak, P. Savicky.
  Softening Splits in Decision Trees Using Simulated Annealing.
  Proceedings of ICANNGA 2007, Warsaw, (Ed.: Beliczynski et. al), Part I, LNCS 4431, pp. 721-729.