Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Искусственный интеллект»

Лабораторная работа№ 1

Студент: Батова Е.Д.

Группа: М8О-301Б-19

Дата:

Оценка:

Москва, 2022

1. Постановка задачи

1. реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
2. Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict (подробнее: https://scikit-learn.org/stable/developers/develop.html)
3. Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline (подробнее: https://scikit-learn.org/stable/modules/compose.html)
4. Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, подробнее здесь: https://scikit-learn.org/stable/modules/grid_search.html), вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями
5. Проделать аналогично с коробочными решениями
6. Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve (подробнее: Hands on machine learning with python and scikit learn chapter 3, mlcourse.ai, https://ml-handbook.ru/chapters/model_evaluation/intro)
7. Проанализировать полученные результаты и сделать выводы о применимости моделей
8. Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

2. Описание программы

Метод	SKLEARN	Реализация
KNN	0.6825	0.55
Логистическая регрессия	0.72	0.6025
SVM	0.7898	0.7175
Naive Bayes	0.6071726	0.74

3. Вывод

Было разделено на два класса: с оценкой больше 5 и меньше 5. Наилучшие оценки показывали реализации sklearn. Наиболее интересным было задание гиперпараметров. Наилучшие показатели были даны в методе SVM.

1. реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
2. Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict (подробнее: https://scikit-learn.org/stable/developers/develop.html)
3. Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline (подробнее: https://scikit-learn.org/stable/modules/compose.html)
4. Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, подробнее здесь: https://scikit-learn.org/stable/modules/grid_search.html), вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями
5. Проделать аналогично с коробочными решениями
6. Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve (подробнее: Hands on machine learning with python and scikit learn chapter 3, mlcourse.ai, https://ml-handbook.ru/chapters/model_evaluation/intro)
7. Проанализировать полученные результаты и сделать выводы о применимости моделей
8. Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pprint
import math
import copy
import joblib
from sklearn import naive_bayes
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.utils import check_random_state
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV, GridSearchCV, train_test_sp
from sklearn.metrics import auc, accuracy_score, confusion_matrix, recall_score, pre
import seaborn as sns
from scipy.special import expit
from scipy.linalg import norm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from scipy import stats
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import euclidean_distances
from collections import defaultdict, Counter
from sklearn.metrics.pairwise import rbf_kernel
```

In [2]:

```
url = "https://raw.githubusercontent.com/e-k-a/RedWineQuality/main/Red_Wine_Quality/
df = pd.read_csv(url)
```

In [3]:

```
df
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	..
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 12 columns

In [4]:

```
import plotly.express as px
df_new = df['quality'].value_counts().rename_axis('Winequality').reset_index(name='c
df_new
fig = px.pie(df_new, values='counts', names='Winequality')
fig.show()
```

In [5]:
```
def boxoutlier(var):
    for x in var.iloc[:,:-1].columns :
        Q1=var[x].quantile(0.25)
        Q3=var[x].quantile(0.75)
        IQR=Q3-Q1
        Lower = Q1-(1.5*IQR)
        Upper = Q3+(1.5*IQR)

        var.loc[:,x]=np.where(var[x].values > Upper,Upper,var[x].values)
        var.loc[:,x]=np.where(var[x].values < Lower,Lower,var[x].values)
    return var

df=boxoutlier(df)
```

Дальше разделим на две части: оценки > 5 и < 5.

In [6]:
```
df['Good'] =df['quality'].apply(lambda x : 1 if(x>5) else 0)
```

In [7]:
```
df
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	..
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 13 columns

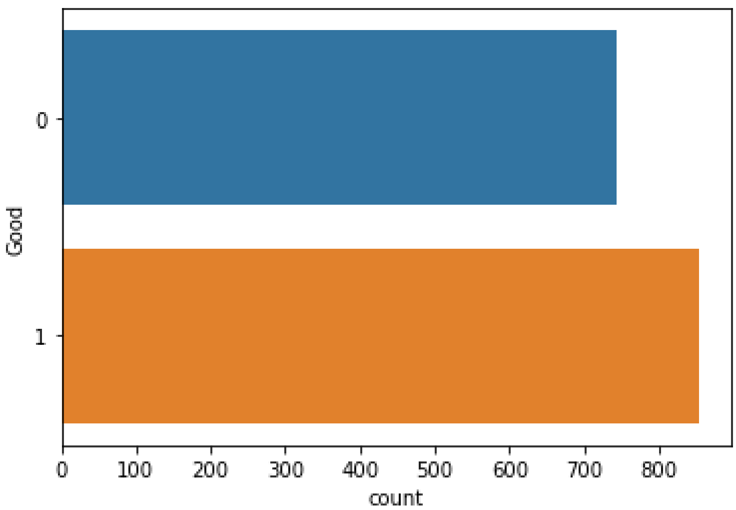
In [8]:
```
df_new = df['Good'].value_counts().rename_axis('Winequality').reset_index(name='coun
df_new
```

```
fig = px.pie(df_new, values='counts', names='Winequality')
fig.show()
```

In [9]:

```
sns.countplot(y = df['Good'])
```

Out[9]: <AxesSubplot:xlabel='count', ylabel='Good'>

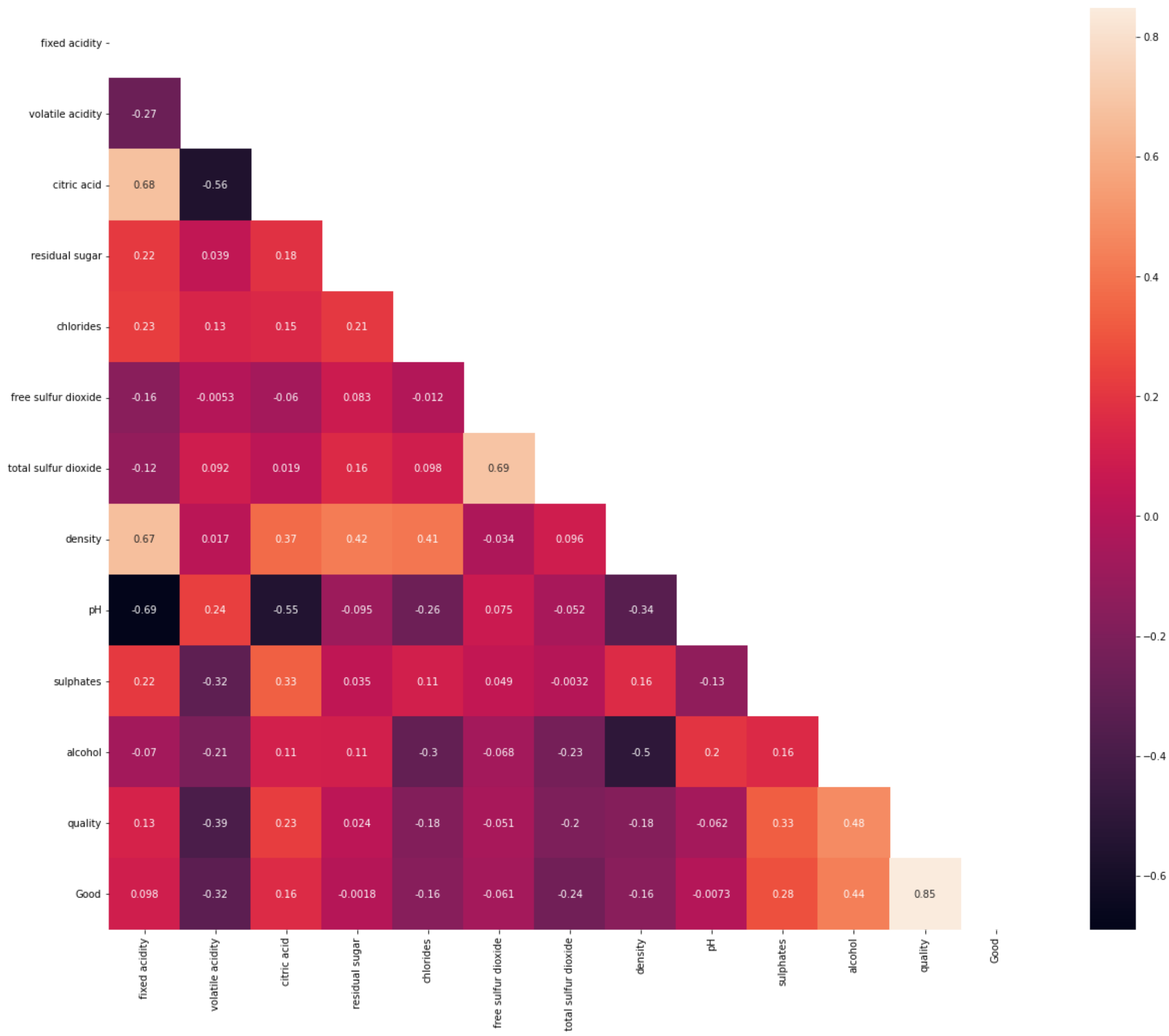


Удалим ненужные стобцы

In [10]:

```
matrix = np.triu(df.corr())
plt.subplots(figsize=(21, 17))
```

```
sns.heatmap(df.corr(), square = True,annot=True, mask=matrix);
plt.savefig('foo2.png', bbox_inches='tight')
```



In [11]:
```
df = df.drop(columns=["pH","residual sugar", "free sulfur dioxide",'residual sugar',
```

Разделим на обучающую и тестовую выборки

In [12]:
```
train, test = train_test_split(df, test_size=0.25)
```

In [13]:
```
train.shape, test.shape
```

Out[13]: ((1199, 9), (400, 9))

In [14]:
```
x_train = train.drop(columns=['Good'])
y_train = train['Good']
```

In [15]:
```
x_test = test.drop(columns=['Good'])
y_test = test['Good']
```

Логистическая регрессия

In [16]:
```
param_grid = {'max_iter': [250,500,1000,2000,3000]}
```

```
base_estimator = LogisticRegression()
base_estimator.get_params().keys()
sh = GridSearchCV(base_estimator, param_grid, cv=5)
sh.fit(x_test, y_test)
sh.best_estimator_
```

Out[16]: LogisticRegression(max_iter=250)

In [17]:

```
sklg = LogisticRegression(max_iter=250)
sklg.fit(x_train, y_train)
sklg.score(x_test, y_test)
```

Out[17]: 0.7325

In [18]:

```
class LogisticReg(BaseEstimator, ClassifierMixin):
    def __init__(self, maxiter=1000, tol=1e-6):
        self.maxiter = maxiter
        self.tol = tol

    def predict(self, X):
        return np.rint(self.sigmoid(X)).astype(np.int)

    def sigmoid(self, X):
        return expit(X @ self.weights)

    def fit(self, X, y):
        m, n = X.shape
        self.weights = np.zeros((n, ))
        alpha = 2*m / norm(X)
        for _ in range(self.maxiter):
            grad = X.T @ (self.sigmoid(X) - y) / m
            self.weights -= alpha * grad
            if norm(grad)**2 < self.tol:
                break

        return self
```

In [19]:

```
param_grid = {'maxiter': [250,500,1000,2000,3000]}
base_estimator = LogisticReg()
base_estimator.get_params().keys()
sh = GridSearchCV(base_estimator, param_grid, cv=5)
sh.fit(x_test, y_test)
sh.best_estimator_
```

Out[19]: LogisticReg(maxiter=3000)

In [20]:

```
sklg = LogisticReg(maxiter=500)
sklg.fit(x_train, y_train)
y_pred_test = sklg.predict(x_test)
y_pred_train = sklg.predict(x_train)
MSE_train = mean_squared_error(y_train, y_pred_train)
MSE_test  = mean_squared_error(y_test,  y_pred_test)
print('MSE train = ', MSE_train, 'MSE test = ', MSE_test)
acc_train = accuracy_score(y_train, y_pred_train)
acc_test = accuracy_score(y_test,  y_pred_test)
print('acc train = ', acc_train, 'acc test = ', acc_test)
```

```
MSE train =  0.359466221851543 MSE test =  0.3975
```

```
acc train =  0.640533778148457 acc test =  0.6025
```

In [21]:

```
print(recall_score(y_test, y_pred_test))
print(precision_score(y_test, y_pred_test))
```

```
0.8613861386138614
0.5704918032786885
```

In [22]:

```
print(confusion_matrix(y_test, y_pred_test))
print(confusion_matrix(y_train, y_pred_train))
```

```
[[ 67 131]
 [ 28 174]]
[[200 346]
 [ 85 568]]
```

In [ ]:

In [23]:

```
joblib.dump(param_grid, "LogisticReg.pkl")
```

Out[23]:

```
['LogisticReg.pkl']
```

knn

In [24]:

```
k_range = list(range(2, 50))
param_grid = {'leaf_size': [ 1, 2, 3, 5, 10],
              'n_neighbors': k_range,
              'n_jobs': [ 1, 2, 3]}
base_estimator = KNeighborsClassifier(5)

sh = GridSearchCV(base_estimator, param_grid, cv=5).fit(x_test, y_test)
sh.best_estimator_
```

Out[24]:

```
KNeighborsClassifier(leaf_size=1, n_jobs=1, n_neighbors=4)
```

In [25]:

```
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
knn.score(x_test, y_test)
```

Out[25]:

```
0.6675
```

In [26]:

```
def eucl_dist(x1, x2):
    return np.sqrt(np.sum(x1-x2)**2)

class KNN(BaseEstimator,ClassifierMixin):
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        X, y = check_X_y(X, y, accept_sparse=True)
        self.is_fitted_ = True
        self.X_train = X
        self.y_train = y
        return self

    def predict(self, X):
```

```
        X = check_array(X, accept_sparse=True)
        k = self.k
        pred = []
        for row in X:
            distances = []
            for i,x in enumerate(self.X_train):
                distance = eucl_dist(row, x)
                distances.append((distance, self.y_train[i]))
            distances.sort()
            result = [x[1] for x in distances[0:k]]
            pred.append(np.argmax([result.count(0), result.count(1), result.count(2)
        self.pred = pred
        return self.pred
```

In [27]:

```
sklg = KNN(k = 10)
sklg.fit(x_train, y_train)
```

Out[27]:

```
KNN(k=10)
```

In [28]:

```
y_pred_test = sklg.predict(x_test)
y_pred_train = sklg.predict(x_train)
```

In [29]:

```
acc_train = accuracy_score(y_train, y_pred_train)
acc_test = accuracy_score(y_test,  y_pred_test)
print('acc train = ', acc_train, 'acc test = ', acc_test)
MSE_train = mean_squared_error(y_train, y_pred_train)
MSE_test  = mean_squared_error(y_test,  y_pred_test)
print('MSE train = ', MSE_train, 'MSE test = ', MSE_test)
```

```
acc train =  0.6872393661384487 acc test =  0.55
MSE train =  0.3127606338615513 MSE test =  0.45
```

In [30]:

```
print(recall_score(y_test, y_pred_test))
print(precision_score(y_test, y_pred_test))
```

```
0.5198019801980198
0.5585106382978723
```

In [31]:

```
print(confusion_matrix(y_test, y_pred_test))
print(confusion_matrix(y_train, y_pred_train))
```

```
[[115  83]
 [ 97 105]]
[[377 169]
 [206 447]]
```

In [32]:

```
joblib.dump(param_grid, "knn.pkl")
```

Out[32]:

```
['knn.pkl']
```

SVM

In [33]:

```
gamma_range = list(np.arange(0.0001, 1, 0.01))
param_grid = {'break_ties': [2, 3, 5, 10],
             'cache_size': [1, 2, 3, 5, 10],
             'gamma': gamma_range}
base_estimator = SVC()
```

```
sh = GridSearchCV(base_estimator, param_grid, cv=5).fit(x_test, y_test)
sh.best_estimator_
```

Out[33]: SVC(break_ties=2, cache_size=1, gamma=0.0901)

In [34]:

```
svc = SVC(break_ties=2, cache_size=1, gamma=0.1301)
svc.fit(x_train, y_train)
svc.score(x_train, y_train)
```

Out[34]: 0.7898248540450375

In [35]:

```
class SVM(BaseEstimator, ClassifierMixin):

    def linear(x_1, x_2):
        return x_1 @ x_2.T

    def __init__(
        self,
        lr: float=1e-3,
        epochs: int=2,
        batch_size: int=64,
        lmbd: float=1e-4,
        kernel_function=None,
    ):
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.lmbd = lmbd


    def fit(self, X, Y):
        assert (np.abs(Y) == 1).all()
        n_obj = len(X)
        X, Y = torch.FloatTensor(X), torch.FloatTensor(Y)
        K = X@X.T

        self.betas = torch.full((n_obj, 1), fill_value=0.001, dtype=X.dtype, require
        self.bias = torch.zeros(1, requires_grad=True)

        optimizer = optim.SGD((self.betas, self.bias), lr=self.lr)
        for epoch in range(self.epochs):
            perm = torch.randperm(n_obj)
            sum_loss = 0.
            for i in range(0, n_obj, self.batch_size):
                batch_inds = perm[i:i + self.batch_size]
                x_batch = X[batch_inds]
                y_batch = Y[batch_inds]
                k_batch = K[batch_inds]

                optimizer.zero_grad()

                preds = k_batch @ self.betas + self.bias
                preds = preds.flatten()
                loss = self.lmbd * self.betas[batch_inds].T @ k_batch @ self.betas +
                loss.backward()
                optimizer.step()
                sum_loss += loss.item()
        self.X = X
        return self

    def predict_scores(self, batch):
```

```
        with torch.no_grad():
            batch = torch.from_numpy(batch).float()
            K = batch @self.X.T
            return (K @ self.betas + self.bias).flatten()

    def predict(self, batch):
        scores = self.predict_scores(batch)
        answers = np.full(len(batch), -1, dtype=np.int64)
        answers[scores > 0] = 1
        return answers
```

In [36]:

```
clf = SVC(kernel='linear').fit(x_train, y_train)
pred = clf.predict(x_test)
```

In [37]:

```
y_pred_test = clf.predict(x_test)
y_pred_train = clf.predict(x_train)
```

In [38]:

```
acc_train = accuracy_score(y_train, y_pred_train)
acc_test = accuracy_score(y_test,  y_pred_test)
print('acc train = ', acc_train, 'acc test = ', acc_test)
MSE_train = mean_squared_error(y_train, y_pred_train)
MSE_test  = mean_squared_error(y_test,  y_pred_test)
print('MSE train = ', MSE_train, 'MSE test = ', MSE_test)
```

```
acc train =  0.7589658048373644 acc test =  0.7175
MSE train =  0.24103419516263552 MSE test =  0.2825
```

In [39]:

```
print(recall_score(y_test, y_pred_test))
print(precision_score(y_test, y_pred_test))
```

```
0.6683168316831684
0.7458563535911602
```

In [40]:

```
print(confusion_matrix(y_test, y_pred_test))
print(confusion_matrix(y_train, y_pred_train))
```

```
[[152  46]
 [ 67 135]]
[[419 127]
 [162 491]]
```

In [41]:

```
joblib.dump(param_grid, "SVM.pkl")
```

Out[41]: ['SVM.pkl']

In [ ]:

NaiveBayes

In [43]:

```
g_range = list(range(1, 25))
```

In [49]:

```
param_grid = {'var_smoothing':g_range}
base_estimator = GaussianNB()
```

```
sh = GridSearchCV(base_estimator, param_grid, cv=5).fit(x_test, y_test)
sh.best_estimator_
```

Out[49]: GaussianNB(var_smoothing=7)

In [54]:
```
g = GaussianNB(var_smoothing=2)
g.fit(x_train, y_train)
y_pred = g.predict(x_test)
g.score(x_train, y_train)
```

Out[54]: 0.6071726438698916

In [ ]:

In [55]:
```
class Naive_Bayes(BaseEstimator, ClassifierMixin) :

    def __init__(self) :
        pass

    def fit(self,X,y) :
        self.classes_ = list(set(y))
        self.classes_.sort()
        self.mus_ = []
        self.sig2s_ = []
        self.y_probs_ = []
        for i in range(len(self.classes_)) :
            self.mus_.append(np.average(X[y==self.classes_[i]],axis=0))
            self.sig2s_.append(np.var(X[y==self.classes_[i]],axis=0))
            self.y_probs_.append(sum(y == self.classes_[i])/len(y))
        return self

    def predict_proba(self,X) :
        m = X.shape[0]
        n = len(self.classes_)
        probs = np.zeros((m,n))
        for i in range(n) :
            probs[:,i] = np.log(self.y_probs_[i])*np.ones_like(probs[:,i]) + \
            np.sum(-(X-self.mus_[i])**2/(2*self.sig2s_[i]),axis=1) - np.sum(0.5*np.l
        return probs

    def predict(self,X) :
        probs = self.predict_proba(X)
        indices = np.argmax(probs,axis=1)
        labels = [self.classes_[indices[i]] for i in range(len(indices))]
        return labels
```

In [56]:
```
mygnb = Naive_Bayes()
mygnb.fit(x_train,y_train)
```

Out[56]: Naive_Bayes()

In [57]:
```
y_pred_test1 = mygnb.predict(x_test)
y_pred_train1 = mygnb.predict(x_train)
```

In [58]:
```
acc_train = accuracy_score(y_train, y_pred_train1)
acc_test = accuracy_score(y_test,  y_pred_test1)
```

```
print('acc train = ', acc_train, 'acc test = ', acc_test)
MSE_train = mean_squared_error(y_train, y_pred_train1)
MSE_test  = mean_squared_error(y_test,  y_pred_test1)
print('MSE train = ', MSE_train, 'MSE test = ', MSE_test)
```

```
acc train =  0.7347789824854045 acc test =  0.74
MSE train =  0.2652210175145955 MSE test =  0.26
```

In [59]:

```
print(recall_score(y_test, y_pred_test1))
print(precision_score(y_test, y_pred_test1))
```

```
0.6782178217821783
0.7784090909090909
```

In [60]:

```
print(confusion_matrix(y_test, y_pred_test1))
print(confusion_matrix(y_train, y_pred_train1))
```

```
[[159  39]
 [ 65 137]]
[[424 122]
 [196 457]]
```

In [61]:

```
joblib.dump(param_grid, "naiveBayes.pkl")
```

Out[61]: ['naiveBayes.pkl']

In [ ]: