Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Дисциплина: «Искусственный интеллект»

Лабораторная работа 1

Тема: Линейные модели

Студент: Шарудин Данила Викторович
Группа: М80-301Б-19
Дата: 10.05.2022
Оценка:

Москва, 2022

Неформальное описание задачи

Вы собрали данные и их проанализировали, визуализировали и представили отчет своим партнерам и спонсорам. Они согласились, что ваша задача имеет перспективу и продемонстрировали заинтересованность в вашем проекте. Самое время реализовать прототип! Вы считаете, что нейронные сети переоценены (просто боитесь признаться, что у вас не хватает ресурсов и данных), и считаете что за машинным обучением классическим будущее и потому собираетесь использовать классические модели. Вашим первым предположением является предположение, что данные и все в этом мире имеет линейную зависимость, ведь не зря же в конце каждой нейронной сети есть линейный слой классификации. В качестве первых моделей вы выбрали, линейную / логистическую регрессию и SVM. Так как вы очень осторожны и боитесь ошибиться, вы хотите реализовать случай, когда все таки мы не делаем никаких предположений о данных, и взяли за основу идею "близкие объекты дают близкий ответ" и идею, что теорема Байеса имеет ранг королевской теоремы. Так как вы не доверяете другим людям, вы хотите реализовать алгоритмы сами с нуля без использования scikit-learn (почти). Вы хотите узнать насколько хорошо ваши модели работают на выбранных вам данных и хотите замерить метрики качества. Ведь вам нужно еще отчитаться спонсорам!

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 гипер параметры моделей с помощью кросс валидации (GridSearchCV, RandomSearchCV, подробнее здесь: https://scikit-learn.org/stable/modules/grid_search.html), вывести и сохранить эти гипер параметры 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 гипер параметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

Точность предсказаний

	sklearn реализация	Моя реализация
LR	0.7696629213	0.7032348805
KNN	0.8033707865	0.7078651685
Naive Bayes	0.7805747126	0.7359550562
SVM	0.7222222222	0.6183141762

Анализ полученных результатов

Наилучшие результаты показали модели Naive Bayes(моя реализация) и KNN(реализация sklearn). С одной стороны, кажется, что результаты предсказаний моделей моих реализаций могли бы быть и получше, но учитывая, что реализации sklearn показывают не намного лучшие результаты (на 5-10%) в целом проведенную работу можно считать успешной.

Выводы

Реализовывать алгоритмы машинного обучения было увлекательно и крайне полезно - это позволило глубже понять как они устроены, а также чем отличаются их подходы. Также полезным оказался анализ полученных результатов - пришлось подумать почему точность оказалось такой, и как сильно мои модели отличаются от реализаций sklearn.

К сожалению, не удалось использовать pipeline. Теоретически мне понятно для чего он используется: он позволяет скомпоновать отдельные мелкие функции по

предварительной обработке данных, обучению, тестированию модели, т.е. переиспользовать участки кода, отвечающие за этот функционал. Но реализовать pipeline у меня не получилось, пришлось отдельно шаг за шагом обрабатывать данные, проводить обучение для каждой модели, хотя можно было это упростить.

Наиболее полезной для меня частью лабораторной работы оказалось знакомство с четвертым пунктом (настройка гипер параметров модели с помощью кросс-валидации), так как это позволило повысить точность предсказания модели без каких либо трудозатрат с моей стороны - нужно было лишь задать варианты гипер параметров в GridSearchCV, из которых выбирается наиболее удачный с точки зрения точности. Хоть GridSearchCV и не дал большого прироста (на 2-3% процента), это все равно полезный инструмент.

Коробочные решения помогли понять принципы работы моделей, некоторые нюансы реализации. Во первых благодаря документации, а во вторых благодаря возможности быстрого тестирования гипотез на уже готовых моделях.

Лабораторная работа позволила познакомится с основными алгоритмами машинного обучения не только в теории, но и на практике.

```
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.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_split
from sklearn.metrics import auc, accuracy_score, confusion_matrix, recall_score, precision
```

Загрузим дату титаника и удалим все лишнее

```
d1 = pd.read_csv('titanic_data.csv')
d2 = pd.read_csv('titanic_surv.csv')
data = pd.concat([d1, d2], axis=1)
```

```
data = data.drop(columns=['Cabin'])
```

```
mean = data['Age'].mean()
std = data['Age'].std()
number_of_nulls = data['Age'].isnull().sum()
random_ages = np.random.randint(mean - std, mean + std, size=number_of_nulls)
```

```
new_ages = data['Age'].copy()
new_ages[np.isnan(new_ages)] = random_ages
data['Age'] = new_ages
```

```
data = data[data['Embarked'].notnull()]
```

```
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data['Embarked'] = data['Embarked'].map({"S": 0, "C": 1, "Q": 2})
```

```
data = data.drop(columns=['Name', 'PassengerId', 'Ticket'])
```

```
data.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
0	3	0	22.0	1	0	7.2500	0	0
1	1	1	38.0	1	0	71.2833	1	1
2	3	1	26.0	0	0	7.9250	0	1
3	1	1	35.0	1	0	53.1000	0	1
4	3	0	35.0	0	0	8.0500	0	0

Разделим на тестовую и обучающую выборки

```
train, test = train_test_split(data, test_size=0.2)
```

```
train.shape, test.shape
```

```
((711, 8), (178, 8))
```

```
x_train = train.drop(columns=['Survived'])
y_train = train['Survived']
```

```
x_test = test.drop(columns=['Survived'])
y_test = test['Survived']
```

Логистическая регрессия

```
class LR(BaseEstimator, ClassifierMixin):
    def __init__(self, step=10**-2, it_c=1):
        self.step = step
        self.it_c = it_c

    def get_coeff(self, x, y):
        err = 0
        x = x.to_numpy()
        y = y.to_numpy()

        for i in range(len(y)):
            if self.predict(x[i]) == y[i]:
                err += 1
        return err / len(y)

    def fit(self, x, y):
        x = x.to_numpy()
        y = y.to_numpy()
```

```
        np.c_[x, np.ones(len(x))]
        self.w = np.zeros(x.shape[1])
        for i in range(self.it_c):
            z = [sum([x[i][j]*self.w[j] for j in range(len(x[i]))]) for i in range(len(x))
            res = np.array([1 / (1 + np.exp(-a)) for a in z])
            grad = np.dot(x.T, res - y) / y.size
            self.w -= self.step * grad
    def predict(self, x):
        np.append(x,1)
        return (1 / (1 + np.exp(-np.dot(x, self.w)))).round()
```

Настройка гиперпараметров:

```
param_grid = {'max_iter': [500,1000,2000,3000]}
base_estimator = LogisticRegression()
sh = GridSearchCV(base_estimator, param_grid, cv=5).fit(x_test, y_test)
sh.best_estimator_
```

```
LogisticRegression(max_iter=500)
```

```
# Моя реализация

param_grid = {'step': [1, 2, 3, 5, 10]}
base_estimator = LR()
sh = HalvingGridSearchCV(base_estimator, param_grid, cv=5,
                         factor=2).fit(x_test, y_test)
sh.best_estimator_
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: c
```

```
mlg = LR(step=5)
hist = mlg.fit(x_train, y_train)
print(f"моя LR: {mlg.get_coeff(x_test, y_test)}")
```

```
моя LR: 0.7191011235955056
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: over
```

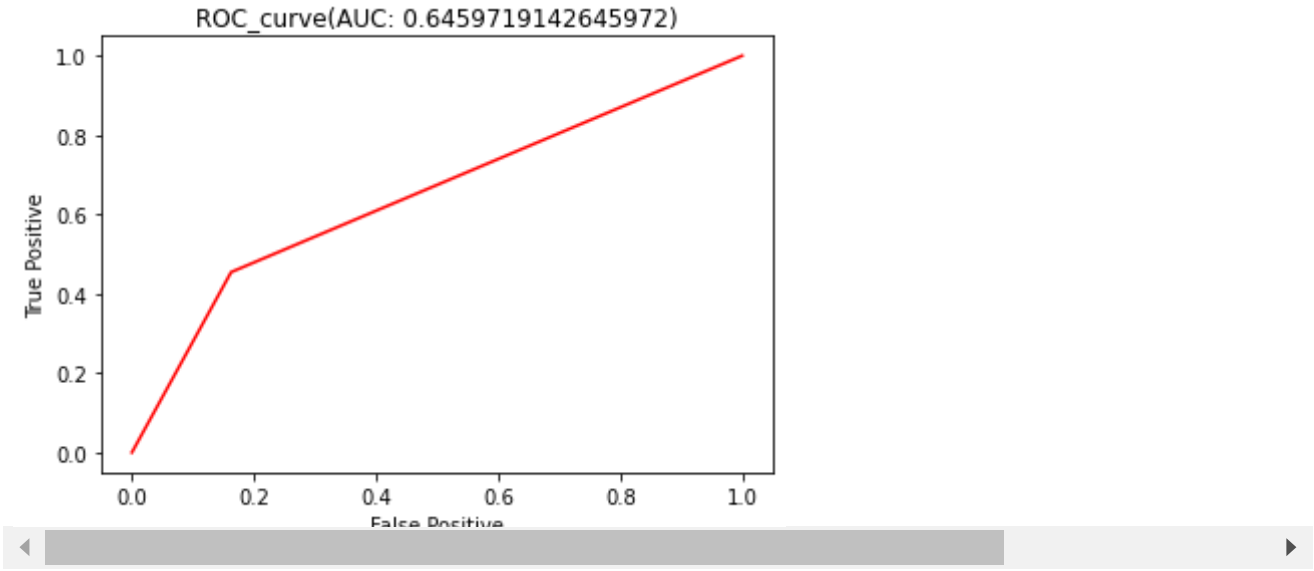
```
sklg = LogisticRegression(max_iter=500)
sklg.fit(x_train, y_train)
print(f"sklearn: {sklg.score(x_test, y_test)}")
```

```
sklearn: 0.8033707865168539
```

```
y_pred_test = mlg.predict(x_test)
false_positive, true_positive, threshold = roc_curve(y_test, y_pred_test, pos_label=1)
print('threshold: ' + str(threshold))
print('false_positive: ' + str(false_positive))
print('true_positive: ' + str(true_positive))
AUC = auc(false_positive, true_positive)
print('AUC: ' + str(AUC))
print('\n\n')
plt.plot(false_positive, true_positive, 'r')
plt.title('ROC_curve' + '(AUC: ' + str(AUC) + ')' )
```

```
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: over
threshold: [2. 1. 0.]
false_positive: [0.         0.16260163 1.        ]
true_positive: [0.         0.45454545 1.        ]
AUC: 0.6459719142645972
```




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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: over
```

```
recall_score(y_train, y_pred_train)
```

```
0.4842105263157895
```

```
recall_score(y_test, y_pred_test)
```

```
0.45454545454545453
```

```
precision_score(y_train, y_pred_train)
```

```
0.6359447004608295
```

```
precision_score(y_test, y_pred_test)
```

```
0.5555555555555556
```

```
confusion_matrix(y_train, y_pred_train)
```

```
array([[347,  79],
       [147, 138]])
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[103,  20],
       [ 30,  25]])
```

```
accuracy_score(y_train, y_pred_train)
```

```
0.6821378340365682
```

```
accuracy_score(y_test, y_pred_test)
```

```
0.7191011235955056
```

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

```
['linear_regression.pkl']
```

KNN

```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self,K,x,y):
        self.K = K
        self.x = x.to_numpy()
        self.y = y.to_numpy()

    def fit(self,x,y):
        x = x.to_numpy()
        res = []
        err = 0
        for d in range(len(x)):
            dist = []
            for i in range(len(self.x)):
                dd = [(x[d][idx] - self.x[i][idx])**2 for idx in range(len(self.x[i]))]
                dist.append([sum(dd)**0.5, self.y[i]])
            dist = sorted(dist)[:self.K]
            inverse_distances = [1/c[0] for c in dist]
            sum_of_inverses = sum(inverse_distances)
            weights = [[inverse / sum_of_inverses, dist[idx][1]] for idx, inverse in enume
            prob = {c : 0 for c in y.unique()}
            for el in weights:
                prob[el[1]] += el[0]
            res.append(max(prob, key=prob.get))
            if res[-1] == y.to_numpy()[d]:
```

```
                err += 1
        return err / len(y.to_numpy())

    def predict(self,f):
        decision = np.dot(f, self.coef.T)
        pred = decision.argmax(axis=0)
        return pred
```

Настройка гиперпараметров:

```
param_grid = {'leaf_size': [ 1, 2, 3, 5, 10],
              'n_neighbors': [ 1, 2, 3, 5, 10, 20, 30],
              '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_
```

```
KNeighborsClassifier(leaf_size=1, n_jobs=1, n_neighbors=10)
```

```
model = KNN(7,x_train,y_train)
model.fit(x_test,y_test)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: divi
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: RuntimeWarning: inva
0.5898876404494382
```

```
scores = []
for i in range(2, 50):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    scores.append((y_test == y_pred).sum() / len(y_test))
max(scores)
```

```
0.7584269662921348
```

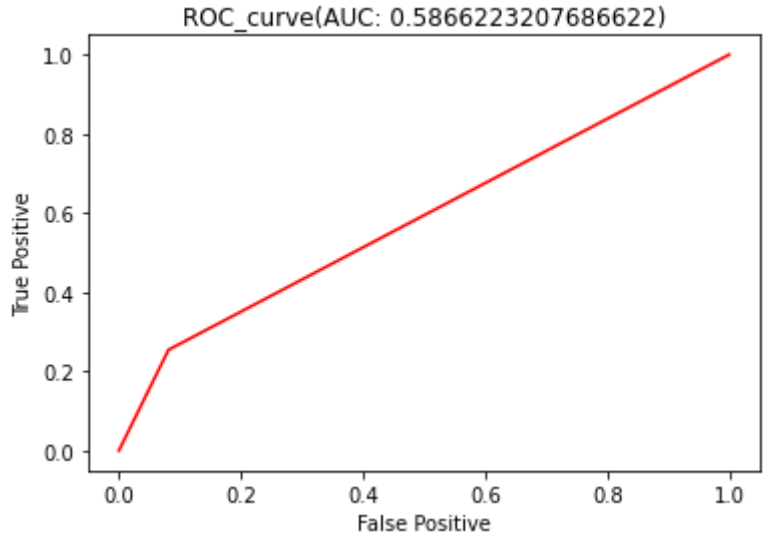
Нормированная дата

```
X_train_normalized = x_train.apply(lambda x: (x-x.mean()) / x.std(), axis=0)
X_test_normalized = x_test.apply(lambda x: (x-x.mean()) / x.std(), axis=0)
scores = []
for i in range(2, 100):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train_normalized, y_train)
    Y_pred = knn.predict(X_test_normalized)
    scores.append((y_test == Y_pred).sum() / len(y_test))
max(scores)
```

```
0.8146067415730337
```

```
y_pred_test = sh.predict(x_test)
false_positive, true_positive, threshold = roc_curve(y_test, y_pred_test, pos_label=1)
print('threshold: ' + str(threshold))
print('false_positive: ' + str(false_positive))
print('true_positive: ' + str(true_positive))
AUC = auc(false_positive, true_positive)
print('AUC: ' + str(AUC))
print('\n\n')
plt.plot(false_positive, true_positive, 'r')
plt.title('ROC_curve' + '(AUC: ' + str(AUC) + ')' )
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()
```

```
threshold: [2 1 0]
false_positive: [0.         0.08130081 1.        ]
true_positive: [0.         0.25454545 1.        ]
AUC: 0.5866223207686622
```




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

```
recall_score(y_train, y_pred_train)
```

```
0.2736842105263158
```

```
recall_score(y_test, y_pred_test)
```

```
0.2545454545454545
```

```
precision_score(y_train, y_pred_train)
```

```
0.65
```

```
precision_score(y_test, y_pred_test)
```

```
0.5833333333333334
```

```
confusion_matrix(y_train, y_pred_train)
```

```
array([[384,  42],
       [207,  78]])
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[113,  10],
       [ 41,  14]])
```

```
accuracy_score(y_train, y_pred_train)
```

```
0.6497890295358649
```

```
accuracy_score(y_test, y_pred_test)
```

```
0.7134831460674157
```

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

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

NaiveBayesClassifier

```
class NBC(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self,x,y):
        self.classes = np.unique(y)
        x = x.to_numpy()
        y = y.to_numpy()

        mean = []
        for j in range(len(x[0])):
            r = []
            for clas in self.classes:
                s = []
                num = 0
                for i in range(len(x)):
                    if y[i] == clas:
```

```
                        s.append(x[i][j])
                        num += 1
                r.append(sum(s) / num)
            mean.append(r)
        self.mean = mean

        stdev = []
        for j in range(len(x[0])):
            r = []
            for clas in range(len(self.classes)):
                s = []
                num = 0
                for i in range(len(x)):
                    if y[i] == self.classes[clas]:
                        s.append(pow(x[i][j] - mean[j][clas],2))
                        num += 1
                r.append(sum(s) / (num-1))
            stdev.append(r)
        self.stdev = stdev

    def predict(self,x):
        res = []
        for clas in range(len(self.classes)):
            r = 1
            for j in range(len(x)):
                r *= (np.exp((-1/2) * ((x[j]-self.mean[j][clas])**2) / (2 * self.stdev[j][
            res.append(r)
        m = 0
        answ = -1
        for clas in range(len(self.classes)):
            if res[clas] > m:
                m = res[clas]
                answ = self.classes[clas]
        return answ

    def predictAll(self,x,y):
        x = x.to_numpy()
        y = y.to_numpy()

        err = 0
        res = []
        for i in range(len(x)):
            answ = self.predict(x[i])
            res.append(answ)
            if answ == y[i]:
                err += 1

        return res, err / len(x)
```

Настройка гиперпараметров:

```
param_grid = {'var_smoothing': [1, 2, 3, 5, 10]}
base_estimator = GaussianNB()
```

```
for parametr in base_estimator.get_params().keys():
  print(parametr)
sh = GridSearchCV(base_estimator, param_grid, cv=5).fit(x_test, y_test)
sh.best_estimator_
```

```
priors
var_smoothing
GaussianNB(var_smoothing=1)
```

```
nbc = NBC()
nbc.fit(x_train, y_train)
res,err = nbc.predictAll(x_test,y_test)
err
```

```
0.7921348314606742
```

```
gaussian = GaussianNB(var_smoothing=3)
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_test)
gaussian.score(x_train, y_train)
```

```
0.6104078762306611
```

```
y_pred_test = nbc.predictAll(x_train, y_train)
false_positive, true_positive, threshold = roc_curve(y_train, y_pred_test[0], pos_label=1)
print('threshold: ' + str(threshold))
print('false_positive: ' + str(false_positive))
print('true_positive: ' + str(true_positive))
AUC = auc(false_positive, true_positive)
print('AUC: ' + str(AUC))
print('\n\n')
plt.plot(false_positive, true_positive, 'r')
plt.title('ROC_curve' + '(AUC: ' + str(AUC) + ')' )
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()
```

```
threshold: [2 1 0]
false_positive: [0.         0.16431925 1.        ]
true_positive: [0.         0.70175439 1.        ]
AUC: 0.7687175685693106
```

```
y_pred_train = nbc.predictAll(x_train, y_train)
y_pred_test = nbc.predictAll(x_test, y_test)
```

```
recall_score(y_train, y_pred_train[0])
```

```
0.7017543859649122
```

```
recall_score(y_test, y_pred_test[0])
```

```
0.6545454545454545
```

```
precision_score(y_train, y_pred_train[0])
```

```
0.7407407407407407
```

```
precision_score(y_test, y_pred_test[0])
```

```
0.6666666666666666
```

```
confusion_matrix(y_train, y_pred_train[0])
```

```
array([[356,  70],
       [ 85, 200]])
```

```
confusion_matrix(y_test, y_pred_test[0])
```

```
array([[105,  18],
       [ 19,  36]])
```

```
accuracy_score(y_train, y_pred_train[0])
```

```
0.7819971870604782
```

```
accuracy_score(y_test, y_pred_test[0])
```

```
0.7921348314606742
```

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

```
['naive_bause.pkl']
```

SVM - Support Vector Machine

```
def projection_simplex(v, z=1):
    n_features = v.shape[0]
    u = np.sort(v)[::-1]
    cssv = np.cumsum(u) - z
    ind = np.arange(n_features) + 1
    cond = u - cssv / ind > 0
    rho = ind[cond][-1]
    theta = cssv[cond][-1] / float(rho)
    w = np.maximum(v - theta, 0)
    return w


class SVM(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1, max_iter=100, eps=0.01, random_state=None, verbose=0):
        self.C = C
        self.max_iter = max_iter
        self.eps = eps
        self.random_state = random_state
        self.verbose = verbose

    def partial_gradient(self,f, t, i):
        print(f[0],self.coef.T)
        f = f.to_numpy()
        g = np.dot(f[i], self.coef.T) + 1
        g[int(t[i])] -= 1
        return g

    def violation(self, g, t, i):
        smallest = np.inf
        for k in range(g.shape[0]):
            if k == t[i] and self.dual_coef[k, i] >= self.C:
                continue
            elif k != t[i] and self.dual_coef[k, i] >= 0:
                continue

            smallest = min(smallest, g[k])
        return g.max() - smallest

    def solver(self, g, t, norms, i):
        Ci = np.zeros(g.shape[0])
        Ci[int(t[i])] = self.C
        beta_hat = norms[i] * (Ci - self.dual_coef[:, i]) + g / norms[i]
        z = self.C * norms[i]
        beta = projection_simplex(beta_hat, z)
        return Ci - self.dual_coef[:, i] - beta / norms[i]

    def fit(self,f, t):
        n_samples, n_features =f.shape
        n_classes = 4
        self.dual_coef = np.zeros((n_classes, n_samples), dtype=np.float64)
```

```
            self.coef = np.zeros((n_classes, n_features))
            norms = np.sqrt(np.sum(f ** 2, axis=1))
            rs = check_random_state(self.random_state)
            ind = np.arange(n_samples)
            rs.shuffle(ind)
            violation_init = None
            for it in range(self.max_iter):
                violation_sum = 0
                for idx in range(n_samples):
                    i = ind[idx]

                    if norms[i] == 0:
                        continue
                    g = self.partial_gradient(f, t, i)
                    v = self.violation(g, t, i)
                    violation_sum += v
                    if v < 1e-12:
                        continue
                    delta = self.solver(g, t, norms, i)
                    self.coef += (delta *f[i][:, np.newaxis]).T
                    self.dual_coef[:, i] += delta
                if it == 0:
                    violation_init = violation_sum
                vratio = violation_sum / violation_init
                if self.verbose >= 1:
                    print("iter", it + 1, "violation", vratio)
                if vratio < self.eps:
                    if self.verbose >= 1:
                        print("Converged")
                    break
            return self

        def predict(self,f):
            decision = np.dot(f, self.coef.T)
            pred = decision.argmax(axis=0)
            return pred

        def get_coeff(self, features, target):
            cnt = 0
            for i in range(target.shape[0]):
                if self.predict(features[i]) == target[i]:
                    cnt += 1
            return cnt / target.shape[0]
```

Настройка гиперпараметров:

```
param_grid = {'break_ties': [2, 3, 5, 10],
              'cache_size': [1, 2, 3, 5, 10],
              'gamma': [0.001, 0.0001]}
base_estimator = SVC(decision_function_shape='ovr')
```

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

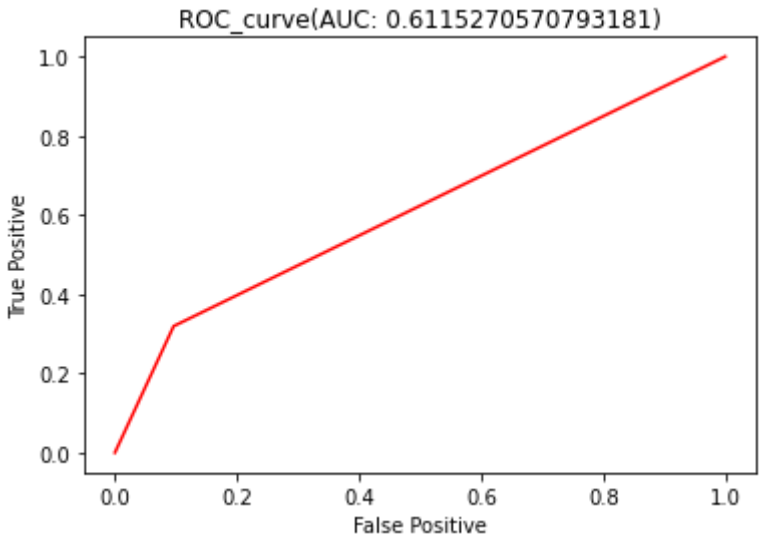
```
SVC(break_ties=2, cache_size=1, gamma=0.001)
```

```
svc = SVC(break_ties=2, cache_size=1, gamma=0.0001)
svc.fit(x_train, y_train)
print('Результат реализации sklearn: {}'
.format(svc.score(x_train, y_train)))
```

```
Результат реализации sklearn: 0.6694796061884669
```

```
y_pred_test = svc.predict(x_train)
false_positive, true_positive, threshold = roc_curve(y_train, y_pred_test, pos_label=1)
print('threshold: ' + str(threshold))
print('false_positive: ' + str(false_positive))
print('true_positive: ' + str(true_positive))
AUC = auc(false_positive, true_positive)
print('AUC: ' + str(AUC))
print('\n\n')
plt.plot(false_positive, true_positive, 'r')
plt.title('ROC_curve' + '(AUC: ' + str(AUC) + ')' )
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()
```

```
threshold: [2 1 0]
false_positive: [0.         0.09624413 1.        ]
true_positive: [0.         0.31929825 1.        ]
AUC: 0.6115270570793181
```




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

```
recall_score(y_train, y_pred_train)
```

```
0.3192982456140351
```

```
recall_score(y_test, y_pred_test)
```

```
0.2909090909090909
```

```
precision_score(y_train, y_pred_train)
```

```
0.6893939393939394
```

```
precision_score(y_test, y_pred_test)
```

```
0.6666666666666666
```

```
confusion_matrix(y_train, y_pred_train)
```

```
array([[385,  41],
       [194,  91]])
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[115,   8],
       [ 39,  16]])
```

```
accuracy_score(y_train, y_pred_train)
```

```
0.6694796061884669
```

```
accuracy_score(y_test, y_pred_test)
```

```
0.7359550561797753
```

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

```
['svm.pkl']
```

✓ 0 сек. выполнено в 11:51