Московский авиационный институт
(национальный исследовательский университет)

Кафедра вычислительной математики и программирования

Лабораторная работа №1
по курсу
Искусственный интеллект

Работу выполнил студент группы
М80-307Б-19:
Савин А.А.

Преподаватель: Самир Ахмед

Оценка:

Дата:

Задание

1) Реализовать следующие алгоритмы машинного обучения: , SVM, KNN, Naive Bayes в отдельных классах
2) Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict
3) Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline
4) Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, 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
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

Ход работы

Для корректной работы с данными, сначала было необходимо избавиться от нулевых элементов таблицы, затем нормализовать оставшиеся данные для того, чтобы они были заданы на промежутке от -1 до 1.
Из загруженного файла создаем тренировочные и тестовые подборки. Для обучения и тестирования соответственно. Соотношения разделения данных 8:2.

Решать задачу предсказания доступности воды для питья пришлось заданными методами.

Linear Regression

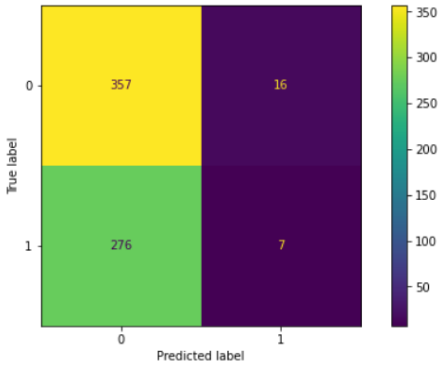
Алгоритм работает на основе градиентного спуска.
Я рассмотрел два варианта использования класса MY_Linear. С помощью GridSearchCV и RandomizedSearchCV. Оба варианта оказались не слишком удачными, но второй был более успешен, показав результат в 0.573 у параметра accuracy, против почти 0.56 у первого варианта.

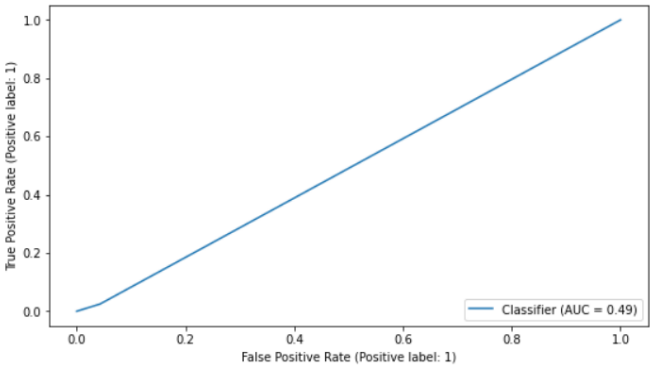
```
class MY_Linear(BaseEstimator, ClassifierMixin):
    def __init__(self, lr, itern, bs):
        self.lr = lr
        self.itern = itern
        self.bs = bs
        pass

    def sigmoid(self, x):
        self.l = 1 / (1 + np.exp(- x))
        return self.l

    def fit(self, data, labels):
```

```
        data = np.concatenate((data, np.ones((data.shape[0],1))), axis = 1
)
        self.W = np.random.normal(0, 1, (len(data[0]),))

        for i in range(self.itern):
            for i in range(0, len(data), self.bs):
                xb = data[i:i + self.bs]
                yb = labels[i:i + self.bs]
                p = np.dot(self.W, xb.T)
                s = self.sigmoid(p)
                dp = np.dot(xb.T, (s - yb).T)
                self.W -= self.lr * dp

    def predict(self, maindata):
        maindata = np.concatenate((maindata, np.ones((maindata.shape[0],1)
)), axis = 1)
        p = np.dot(self.W, maindata.T)
        s = self.sigmoid(p)
        return (s > 0.5).astype('int64')
```

GridSearchC

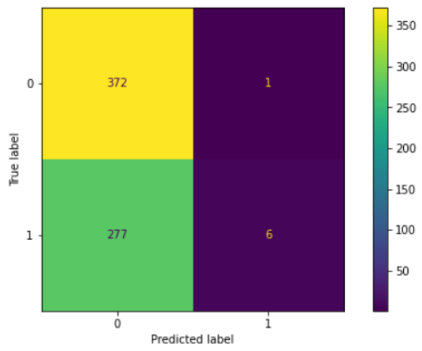
```
Accuracy: 0.5548780487804879
Recall: 0.024734982332155476
Precision: 0.30434782608695654
```

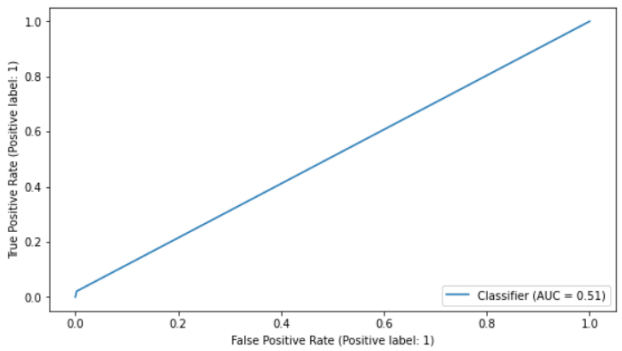




RandomizedSearchCV

```
Accuracy: 0.5762195121951219
Recall: 0.02120141342756184
Precision: 0.8571428571428571
```





SVM

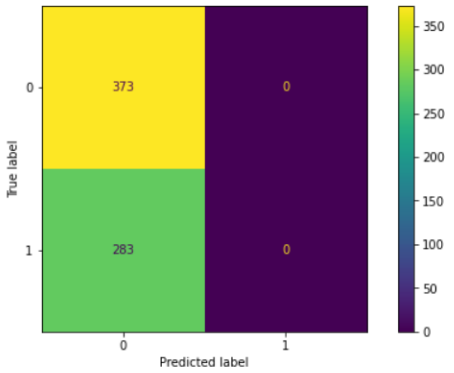
Для SVM используется код из 1 лабораторной работы по персептрону. Класс Net, Layer и прочий функционал.

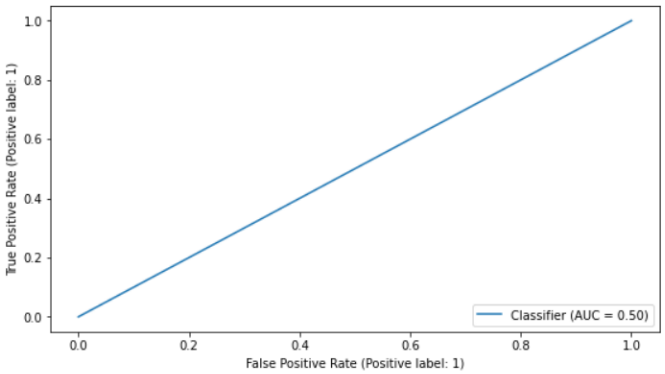
Цель алгоритма - определить гиперплоскость (также называется “разделяющей” или “ГПР”), которая разделяет точки на два класса. Все векторы, которые падают ниже разделяющей гиперплоскости, принадлежат классу -1, а если выше, то 1. Точность вычислений методом опорных векторов равна 0.56. Такой результат достигается как библиотечным, так и написанным вручную методом.

```
class MY_SVM(ClassifierMixin, BaseEstimator):
    def __init__(self, epoches=1, batch_size=10, SGD_step=0.001, alpha=0.1
, nin=10):
        self.epoches = epoches
        self.batch_size = batch_size
        self.SGD_step = SGD_step
        self.nin = nin
        self.alpha = alpha
        self.Net = SoftMarginSVM(nin, alpha)

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        self.is_fitted_ = True
        for _ in range(self.epoches):
            self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
        return self

    def predict(self, X):
        y = self.Net.predict(X)
        return y

    def getW(self):
        return self.Net.W
```

MY_SVM

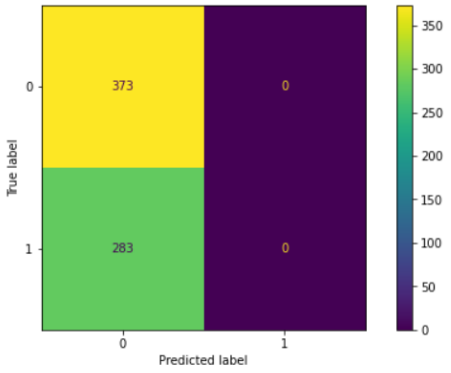
```
Accuracy: 0.5685975609756098
Recall: 0.0
Precision: 0.0
```

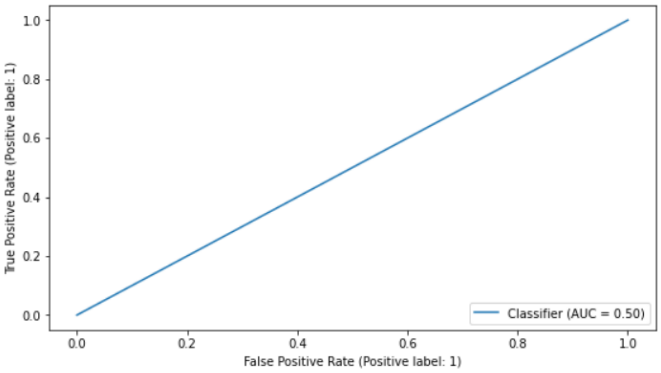




LinearSV

```
Accuracy: 0.5685975609756098
Recall: 0.0
Precision: 0.0
```





KNN

Суть метода в анализе характеристик соседних точек и предположения собственной характеристики на основании данных соседей. Для анализа используется евклидова метрика:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

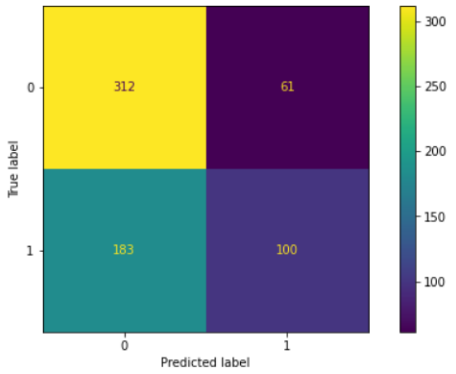
Метод Метод k-ближайших соседей показал себя лучше остальных. Показав идентичные результаты для библиотечного класса и написанного вручную. Параметр accuracy равен ~0.62.

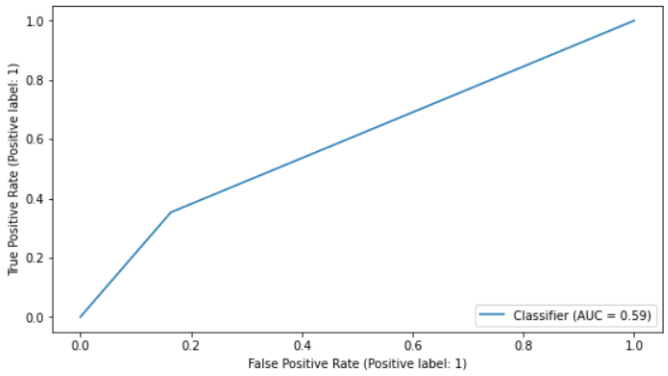
```
class MY_KNN(ClassifierMixin, BaseEstimator):
    def __init__(self, k = 1):
        self.k = k

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        return self

    def predict(self, X):
        check_is_fitted(self, ["X_", "y_"])

        X = check_array(X)

        y = np.ndarray((X.shape[0],))
        for (i, elem) in enumerate(X):
            distances = euclidean_distances([elem], self.X_)[0]
            neighbors = np.argpartition(distances, kth = self. k - 1)
            k_neighbors = neighbors[:self.k]
```

```
            labels, cnts = np.unique(self.y_[k_neighbors], return_counts =
 True)
            y[i] = labels[cnts.argmax()]
        return y
```

MY_KNN

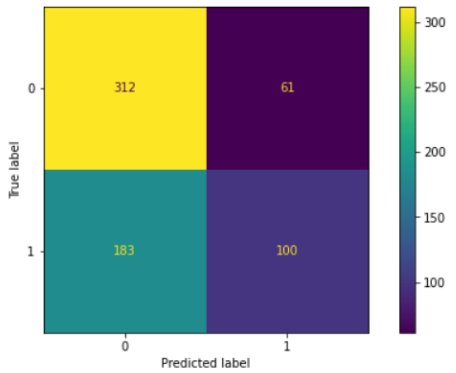
```
Accuracy: 0.6280487804878049
Recall: 0.35335689045936397
Precision: 0.6211180124223602
```

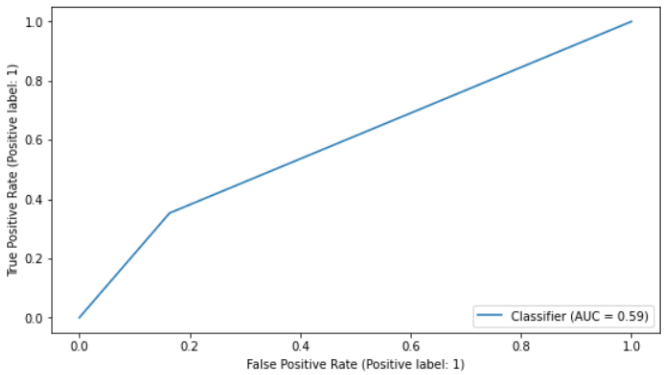




KNeighborsClassifier

```
Accuracy: 0.6280487804878049
Recall: 0.35335689045936397
Precision: 0.6211180124223602
```





Naive Bayes

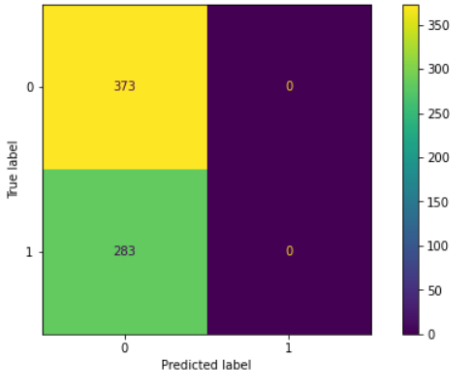
Метод заключается в предположении о независимости параметров системы. В этом и заключается наивность метода. Т.е. здесь по теореме Байеса зная вероятность наступления события от некоторой причины, алгоритм просчитывает, могла ли эта причина привести к текущей ситуации.
В данном случае, библиотечный вариант выигрывает у MY_Naive_Bayes 0.04 по параметру accuracy.

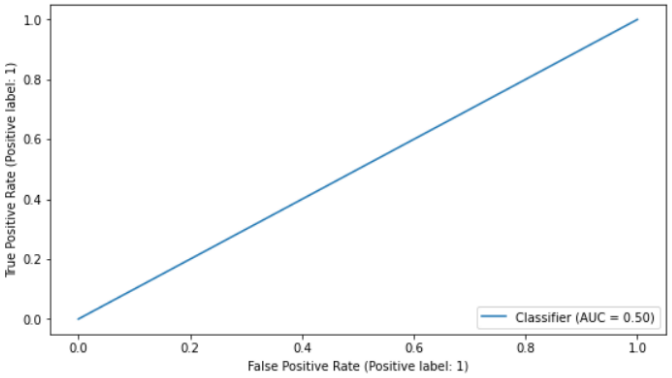
```
class MY_Naive_Bayes(ClassifierMixin, BaseEstimator):
    def __init__(self):
        None

    def fit(self, X, y):
        X, y = check_X_y(X, y)

        self.X_ = X
        self.y_ = y

        labels, cnts = np.unique(self.y_, return_counts = True)
```

```
        self.labels = labels
        self.p_of_y = np.array([elem / self.y_.shape[0] for elem in cnts])
        self.means = np.array([self.X_[self.y_ == elem].mean(axis = 0) for
 elem in labels])
        self.stds = np.array([self.X_[self.y_ == elem].std(axis = 0) for e
lem in labels])
        return self

    def gaussian(self, mu, sigma, x0):
        return np.exp(-(x0 -
 mu) ** 2 / (2 * sigma)) / np.sqrt(2.0 * pi * sigma)

    def predict(self, X):
        check_is_fitted(self, ['X_', 'y_'])

        X = check_array(X)

        res = np.zeros(X.shape[0])
        for (i, elem) in enumerate(X):
            p = np.array(self.p_of_y)
            for (j, label) in enumerate(self.labels):
                p_x_cond_y = np.array([self.gaussian(self.means[j][k], sel
f.stds[j][k], elem[k]) for k in range(X.shape[1])])
                p[j] *= np.prod(p_x_cond_y)
            res[i] = np.argmax(p)
        return res
```

MY Navie Bayess

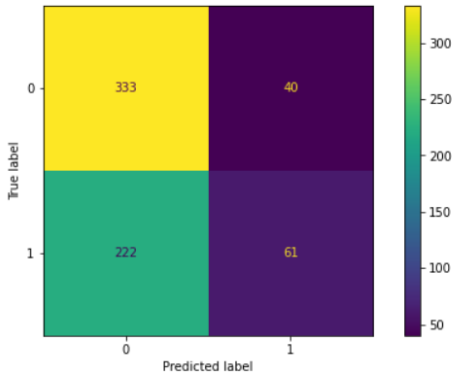
```
Accuracy: 0.5685975609756098
Recall: 0.0
Precision: 0.0
```

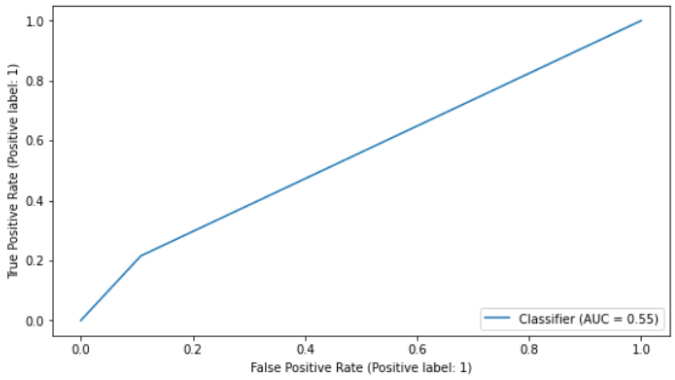




GaussianNB

```
Accuracy: 0.600609756097561
Recall: 0.21554770318021202
Precision: 0.6039603960396039
```





Вывод

Так получилось, что при разных гиперпараметрах точность предсказания все еще довольно мала. Такие предсказания все еще не пригодны для полноценного использования.

Лучший результат показал метод k ближайших соседей, показав параметр accuracy равный 0.628.

Посмотрев код других пользователей для данной модели на kaggle, я обнаружил, что у работы с самым большим рейтингом параметр accuracy равен 0.66 (https://www.kaggle.com/code/jaykumar1607/water-quality-analysis-plotly-and-modelling)

Из других работ также нашалсь программа с точностью в 76% (https://www.kaggle.com/code/d4rklucif3r/water-quality-luciferml-76-deployment).

Таким образом, напрашивается вывод, что предсказать можно ли будет пить воду по заданным в датасете характеристикам не так уж и просто.