Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1 по курсу «Искусственный интеллект»

Студент: М. А. Инютин
Преподаватели: Д. В. Сошников
С. Х. Ахмед
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №1

Задача: 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 классическим машинным обучением 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!

Формально говоря вам предстоит сделать следующее:

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;
3. Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline;
4. Вы должны настроить гиперпараметры моделей с помощью кросс валидации, 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.

1 Описание

В наборе данных имеется категориальный признак, его нужно разделить на несколько признаков с помощью one-hot encoding. Это нужно, чтобы не создавать лишнюю числовую связь. Pandas имеет метод get_dummies, который это делает.

Для корректной работы моделей признаки нужно нормализовать. scikit-learn позволяет сделать это с помощью normalize [3], я использую метрику «max».

Теперь данные можно разделить на тестовую и обучающую выборку (80 к 20), используя train_test_split из scikit-learn [2].

В задании требуется сделать все модели совместимыми с scikit-learn [4], поэтому получение оценок модели [1] можно сделать методами из этой же библотеки:

```
1  def scores(model, X, y_true):
2      y_pred = model.predict(X)
3      print("Accuracy:", accuracy_score(y_true, y_pred))
4      print("Recall:", recall_score(y_true, y_pred))
5      print("Precision:", precision_score(y_true, y_pred))
6      figure = plt.figure(figsize = (20, 5))
7      matr = confusion_matrix(y_true, y_pred)
8      ax = plt.subplot(1, 2, 1)
9      ConfusionMatrixDisplay(matr).plot(ax = ax)
10     ax = plt.subplot(1, 2, 2)
11     RocCurveDisplay.from_predictions(y_true = y_true, y_pred = y_pred, name = "ROC-
           curve", ax = ax)
12     plt.show()
```

При реализации моделей я использовал шаблон [5] из scikit-learn, в котором учтены все тонкости реализации: наследование от нужных классов (ClassifierMixin, BaseEstimator) и необходимые методы (fit, predict).

1 Метод k-ближайших соседей

Среди всех объектов обучающей выборки ищем k ближайших, среди них классифицируем объект тем классом, которого больше всего среди соседей:

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

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)
        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])
        # Input validation
        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
```

Использую Евклидову метрику из scikit-learn для вычисления расстояний и метод argpartition из numpy [9] для индексной сортировки.

2 Логистическая регрессия

Логистическая регрессия по сути является однослойной нейросетью. Использую наработки прошлых работ, чтобы реализовать модель. Описываю класс сети, которую можно строить из разных слоёв:

```
class Net:
    def __init__(self, loss_function):
        self.layers = []
        self.loss = loss_function()

    def append(self, layer):
        self.layers.append(layer)

    def forward(self, x):
        for layer in self.layers:
            x = layer.forward(x)
        return x

    def backward(self, z):
        for layer in self.layers[::-1]:
            z = layer.backward(z)
        return z

    def forward_loss(self, x, y):
        p = self.forward(x)
        return self.loss.forward(p, y)

    def backward_loss(self, l):
        dp = None
        dp = self.loss.backward(l)
        return self.backward(dp)

    def update(self, step):
        for layer in self.layers:
            if "update" in layer.__dir__():
                layer.update(step)

    def train_epoch(self, x, y, batch_size = 100, step = 1e-7):
        for i in range(0, len(x), batch_size):
            xb = x[i:i + batch_size]
            yb = y[i:i + batch_size]
            loss = self.forward_loss(xb, yb)
            dx = self.backward_loss(loss)
            self.update(step)
```

Линейный слой сети с возможностью обновления весов (изначально матрица весов заполняется случайными числами, а вектор смещения нулями) так же вынесен в отдельный класс:

```
class Linear:
    def __init__(self, n, m):
        mu = 0.0
        sigma = 1.0 / np.sqrt(2.0 * n)
        self.W = np.random.normal(mu, sigma, (m, n))
        self.b = np.zeros((1, m))
        self.dW = np.zeros((m, n))
        self.db = np.zeros((1, m))

    def forward(self, x):
        self.x = x
        z = np.dot(x, self.W.T) + self.b
        return z

    def backward(self, dz):
        dx = np.dot(dz, self.W)
        dW = np.dot(dz.T, self.x)
        db = dz.sum(axis = 0)
        self.dW = dW
        self.db = db
        return dx

    def update(self, step):
        self.W -= step * self.dW
        self.b -= step * self.db
```

В логистической регрессии используется функция активации сигмоида, которая так же описана в отдельном классе:

$$\sigma(x) = (1 + e^{-x})^{-1}$$

```
class Sigmoid:
    def forward(self, x):
        self.y = 1.0 / (1.0 + np.exp(-x))
        return self.y

    def backward(self, dy):
        return self.y * (1.0 - self.y) * dy
```

В качестве функции потерь использую binary cross entropy loss, так как стоит задача бинарной классификации:

```
class BinaryCrossEntropy:
    def forward(self, p, y):
        y = y.reshape((y.shape[0], 1))
        self.p = p
        self.y = y
        res = y * np.log(p) + (1 - y) * np.log(1 - p)
        return -np.mean(res)

    def backward(self, loss):
        res = (self.p - self.y) / (self.p * (1 - self.p))
        return res / self.p.shape[0]
```

Логистическая регрессия содержит нейросеть, состоящую из линейного слоя, сигмиоды и описанной выше функции потерь. Алгоритм обучения сети — стохастический градиентный спуск с постоянным шагом:

```
class LogisticRegression(ClassifierMixin, BaseEstimator):
    def __init__(self, epoches = 1, batch_size = 10, SGD_step = 0.001):
        self.epoches = epoches
        self.batch_size = batch_size
        self.SGD_step = SGD_step
        self.Net = Net(BinaryCrossEntropy)
        self.Net.append(Linear(8, 1))
        self.Net.append(Sigmoid())

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)
        self.X_ = X
        self.y_ = y
        for _ in range(self.epoches):
            self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])
        # Input validation
        X = check_array(X)
        y = self.Net.forward(X)
        res = np.where(y < 0.5, 0, 1)
        return res
```

3 Метод опорных векторов

Метод похож на предыдущий, но функция ошибки требует сами данные, на которых происходит обучение, поэтому встроить в класс сети не получилось. Отдельно описываю модель опорных векторов с мягким зазором:

```
class SoftMarginSVM:
    def __init__(self, n, alpha):
        self.alpha = alpha
        mu = 0.0
        sigma = 1.0 / np.sqrt(n)
        self.W = np.random.normal(mu, sigma, (1, n + 1))

    def forward(self, x):
        z = np.dot(x, self.W.T)
        return z

    def add_ones(self, x):
        ones = np.ones((x.shape[0], 1))
        return np.hstack((x, ones))

    def predict(self, x):
        res = self.forward(self.add_ones(x))
        return np.where(res < 0, 0, 1)

    def train_epoch(self, x, y, batch_size = 100, step = 1e-7):
        x = self.add_ones(x)
        y = np.where(y > 0, 1, -1)
        for i in range(0, len(x), batch_size):
            xb = x[i:i + batch_size]
            yb = y[i:i + batch_size]

            pred = self.forward(xb)
            grad = self.alpha * self.W
            for i in range(len(xb)):
                if (yb[i] * pred[i] < 1):
                    grad -= yb[i] * xb[i]
            self.W -= step * grad
```

Класс получился простой, но не очень универсальный. Чтобы отбросить вектор смещения b, я ввёл искуственный признак, который всегда равен единице [14].

Эта модель затем встраивается в классификатор. Параметры почти такие же, как и в случае с логистической регрессией:

```
class SVM(ClassifierMixin, BaseEstimator):
    def __init__(self, epoches = 1, batch_size = 10, SGD_step = 0.001, alpha = 0.1):
        self.epoches = epoches
        self.batch_size = batch_size
        self.SGD_step = SGD_step
        self.alpha = alpha
        self.Net = SoftMarginSVM(8, alpha)

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        for _ in range(self.epoches):
            self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
        # Return the classifier
        return self

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

4 Наивный байесовский классификатор

Идея модели в наивном предположении о независимости параметров. Так же часто используется модель с нормальным распределением признаков. В прошлой работе гистограммы 4 из 6 распределены нормально, поэтому попробую применить такой метод:

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

    def fit(self, X, y):
        # Check that X and y have correct shape
        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 elem in labels
            ])
        # Return the classifier
        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 fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        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], self.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
```

5 Подбор гиперпараметров

Для подбора гиперпараметров используются кросс-валидации GridSearchCV [7] и RandomizedSearchCV [8]. Приведу пример использования с SVM:

```
gscv = GridSearchCV(Pipeline([("SVM", SVM())]),
                    {"SVM__epoches" : [1, 2, 4],
                     "SVM__batch_size" : [5, 10, 20],
                     "SVM__SGD_step" : [0.01, 0.05, 0.1],
                     "SVM__alpha" : [1.0, 0.1, 0.01, 0.0]})
gscv.fit(train_X, train_y)
best(gscv)
```

```
rscv = RandomizedSearchCV(Pipeline([("SVM", SVM())]),
                    {"SVM__epoches" : [1, 2, 4],
                     "SVM__batch_size" : [5, 10, 20],
                     "SVM__SGD_step" : [0.01, 0.05, 0.1],
                     "SVM__alpha" : [1.0, 0.1, 0.01, 0.0]})
rscv.fit(train_X, train_y)
best(rscv)
```

Наивный байесовский классификатор не имеет параметров, поэтому для него поиск гиперпараметров не осуществляется.

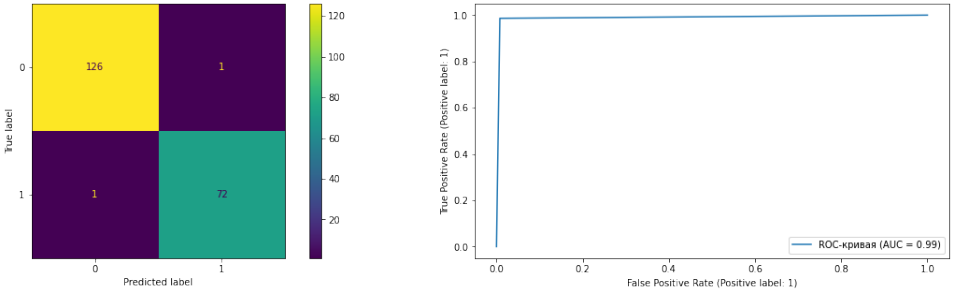
2 Результаты моделей

Так как данные очень хорошо линейно разделимы, явные выбросы были удалены, все модели дают одинаковый результат с поразительной точностью 99%. Ниже приведены результаты всех моделей. Сначала модели, реализованные вручную, затем из библиотеки.

1 Метод k-ближайших соседей

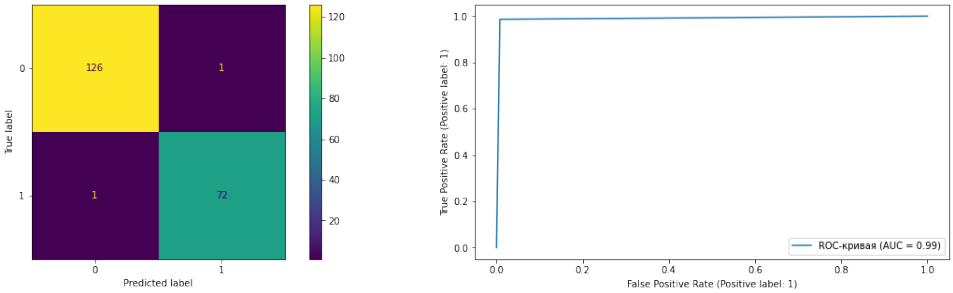
1.1 kNN

```
Лучшие гиперпараметры модели: 'knn__k': 3
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```



1.2 sklearn.neighbors.KNeighborsClassifier

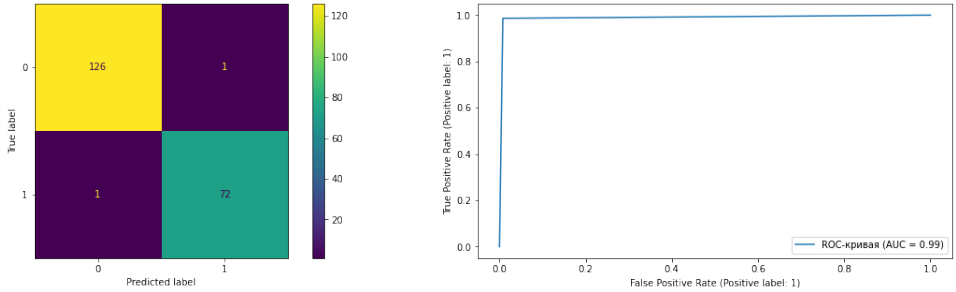
```
Лучшие гиперпараметры модели: 'knn__n_neighbors': 3
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```



2 Логистическая регрессия

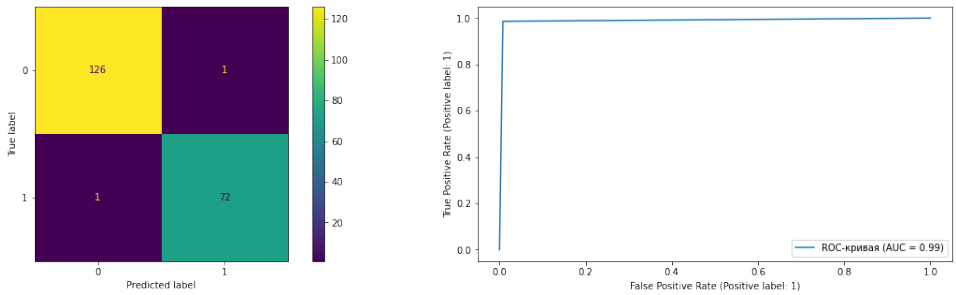
2.1 LogisticRegression

```
Лучшие гиперпараметры модели: 'logreg__SGD_step': 0.05,
'logreg__batch_size': 5,'logreg__epoches': 4
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```



2.2 sklearn.linear_model.LogisticRegression

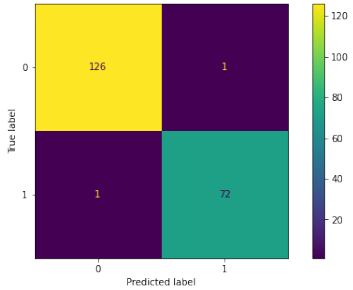
```
Лучшие гиперпараметры модели: 'logreg__penalty': 'l2',
'logreg__solver': 'newton-cg'
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```

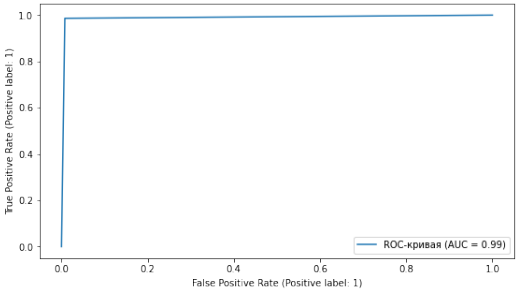


3 Метод опорных векторов

3.1 SVM

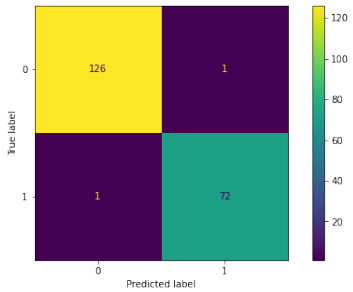
Лучшие гиперпараметры модели: 'SVM__SGD_step': 0.01,
'SVM__alpha': 1.0,'SVM__batch_size': 20,'SVM__epoches': 4
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136

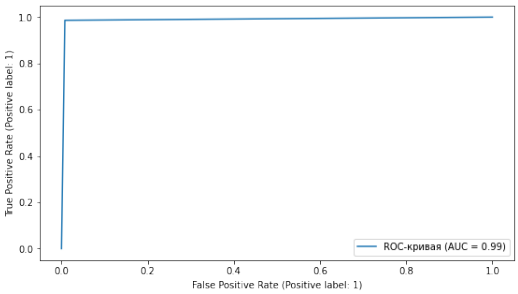




3.2 sklearn.svm.SVC

Лучшие гиперпараметры модели: 'svc__kernel': 'poly'
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136

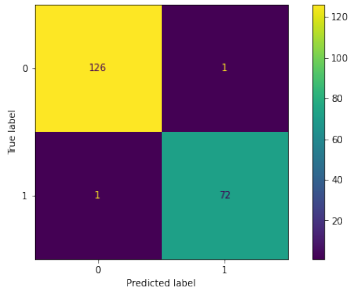


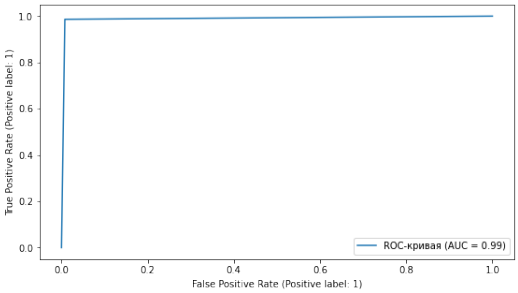


4 Наивный байесовский классификатор

4.1 NaiveBayes

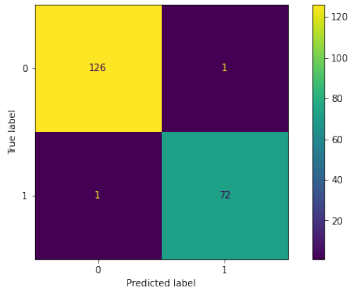
```
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```

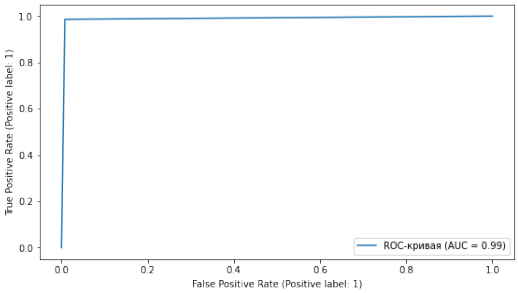




4.2 sklearn.naive_bayes.GaussianNB

```
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```





3 Выводы

В ходе выполнения лабораторной работы я познакомился с линейными моделями классического машинного обучения: логистической регрессией, методом опорных векторов, наивным байесовским классификатором и методом k-ближайших соседей. Больше все заинтересовал алгоритм k-ближайших соседей для классификации, так как он кажется интуитивно наиболее хорошим. Однако его недостаток в большой вычислительной сложности.

Основная сложность в работе — реализация каждого метода вручную, пришлось искать очень много методов из библиотек numpy и scikit-learn, чтобы легко работать с данными. Пожалуй, это заняло большую часть времени.

В результате набор данных Smoker Condition получилось разделить линейными моделями с поразительной точностью 99%. Точность такая высокая, потому что перед обучением я удалил около двух десятков выбросов и неполных данных.

Особенно хочется отметить, что такой точности получается добиться далеко не всегда, потому что реальный мир сложнее, чем линейная модель. При поиске подходящего набора данных я находил такие, где точки классов перемешанны самым разнообразным образом. Летом хочу из интереса попробовать применить алгоритмы классического машинного обучения на этих наборах.

Список литературы

[1] *Оценка качества моделей — Учебник по ML от ШАД*
URL: https://ml-handbook.ru/chapters/model_evaluation/intro
(дата обращения: 08.05.2022).

[2] *sklearn.model_selection.train_test_split*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
(дата обращения: 09.05.2022).

[3] *sklearn.preprocessing.normalize*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
(дата обращения: 09.05.2022).

[4] *Developing scikit-learn estimators*
URL: https://scikit-learn.org/stable/developers/develop.html
(дата обращения: 09.05.2022).

[5] *project-template/_template.py at master · scikit-learn-contrib*
URL: https://github.com/scikit-learn-contrib/project-template/blob/master/skltemplate/_template.py
(дата обращения: 09.05.2022).

[6] *sklearn.pipeline.Pipeline — scikit-learn 1.0.2 documentation*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
(дата обращения: 09.05.2022).

[7] *sklearn.model_selection.GridSearchCV*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
(дата обращения: 09.05.2022).

[8] *sklearn.model_selection.RandomizedSearchCV*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
(дата обращения: 09.05.2022).

[9] *numpy.argpartition — NumPy v1.22 Manual*
URL: https://numpy.org/doc/stable/reference/generated/numpy.argpartition.html
(дата обращения: 09.05.2022).

[10] *numpy.unique — NumPy v1.22 Manual*
URL: https://numpy.org/doc/stable/reference/generated/numpy.unique.html
(дата обращения: 09.05.2022).

[11] *sklearn.neighbors.KNeighborsClassifier*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
(дата обращения: 09.05.2022).

[12] *numpy.where — NumPy v1.22 Manual*
URL: https://numpy.org/doc/stable/reference/generated/numpy.where.html
(дата обращения: 09.05.2022).

[13] *sklearn.linear_model.LogisticRegression*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
(дата обращения: 10.05.2022).

[14] *SVM. Объяснение с нуля и реализация на Python. Подробный разбор метода опорных векторов*
URL: https://habr.com/ru/company/ods/blog/484148/
(дата обращения: 10.05.2022).

[15] *sklearn.svm.SVC — scikit-learn 1.0.2 documentation*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
(дата обращения: 10.05.2022).

[16] *Машинное обучение. Байесовская классификация. К.В. Воронцов, Школа анализа данных, Яндекс.*
URL: https://www.youtube.com/watch?v=qMndsltzNGA&t
(дата обращения: 10.05.2022).

[17] *numpy.mean — NumPy v1.22 Manual*
URL: https://numpy.org/doc/stable/reference/generated/numpy.mean.html
(дата обращения: 10.05.2022).

[18] *numpy.std — NumPy v1.22 Manual*
URL: https://numpy.org/doc/stable/reference/generated/numpy.std.html
(дата обращения: 10.05.2022).

[19] *sklearn.naive_bayes.GaussianNB*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
(дата обращения: 10.05.2022).