

**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, Naïve 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. Прodelать 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 библиотеки:

```
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-
12    curve", ax = ax)
13    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:

```
1 class kNN(ClassifierMixin, BaseEstimator):
2     def __init__(self, k = 1):
3         self.k = k
4
5     def fit(self, X, y):
6         # Check that X and y have correct shape
7         X, y = check_X_y(X, y)
8         # Store the classes seen during fit
9         self.classes_ = unique_labels(y)
10        self.X_ = X
11        self.y_ = y
12        # Return the classifier
13        return self
14
15    def predict(self, X):
16        # Check is fit had been called
17        check_is_fitted(self, ['X_', 'y_'])
18        # Input validation
19        X = check_array(X)
20        y = np.ndarray((X.shape[0],))
21        for (i, elem) in enumerate(X):
22            distances = euclidean_distances([elem], self.X_)[0]
23            neighbors = np.argpartition(distances, kth = self.k - 1)
24            k_neighbors = neighbors[:self.k]
25            labels, cnts = np.unique(self.y_[k_neighbors], return_counts = True)
26            y[i] = labels[cnts.argmax()]
27        return y
```

Используя 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:

```
1 class Net:
2     def __init__(self, loss_function):
3         self.layers = []
4         self.loss = loss_function()
5
6     def append(self, layer):
7         self.layers.append(layer)
8
9     def forward(self, x):
10        for layer in self.layers:
11            x = layer.forward(x)
12        return x
13
14    def backward(self, z):
15        for layer in self.layers[::-1]:
16            z = layer.backward(z)
17        return z
18
19    def forward_loss(self, x, y):
20        p = self.forward(x)
21        return self.loss.forward(p, y)
22
23    def backward_loss(self, l):
24        dp = None
25        dp = self.loss.backward(l)
26        return self.backward(dp)
27
28    def update(self, step):
29        for layer in self.layers:
30            if "update" in layer.__dir__():
31                layer.update(step)
32
33    def train_epoch(self, x, y, batch_size = 100, step = 1e-7):
34        for i in range(0, len(x), batch_size):
35            xb = x[i:i + batch_size]
36            yb = y[i:i + batch_size]
37            loss = self.forward_loss(xb, yb)
38            dx = self.backward_loss(loss)
39            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:

```
1 class Linear:
2     def __init__(self, n, m):
3         mu = 0.0
4         sigma = 1.0 / np.sqrt(2.0 * n)
5         self.W = np.random.normal(mu, sigma, (m, n))
6         self.b = np.zeros((1, m))
7         self.dW = np.zeros((m, n))
8         self.db = np.zeros((1, m))
9
10    def forward(self, x):
11        self.x = x
12        z = np.dot(x, self.W.T) + self.b
13        return z
14
15    def backward(self, dz):
16        dx = np.dot(dz, self.W)
17        dW = np.dot(dz.T, self.x)
18        db = dz.sum(axis = 0)
19        self.dW = dW
20        self.db = db
21        return dx
22
23    def update(self, step):
24        self.W -= step * self.dW
25        self.b -= step * self.db
```

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

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

```
1 class Sigmoid:
2     def forward(self, x):
3         self.y = 1.0 / (1.0 + np.exp(-x))
4         return self.y
5
6     def backward(self, dy):
7         return self.y * (1.0 - self.y) * dy
```

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

```
1 class BinaryCrossEntropy:
2     def forward(self, p, y):
3         y = y.reshape((y.shape[0], 1))
4         self.p = p
5         self.y = y
6         res = y * np.log(p) + (1 - y) * np.log(1 - p)
7         return -np.mean(res)
8
9     def backward(self, loss):
10        res = (self.p - self.y) / (self.p * (1 - self.p))
11        return res / self.p.shape[0]
```

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

```
1 class LogisticRegression(ClassifierMixin, BaseEstimator):
2     def __init__(self, epoches = 1, batch_size = 10, SGD_step = 0.001):
3         self.epoches = epoches
4         self.batch_size = batch_size
5         self.SGD_step = SGD_step
6         self.Net = Net(BinaryCrossEntropy)
7         self.Net.append(Linear(8, 1))
8         self.Net.append(Sigmoid())
9
10    def fit(self, X, y):
11        # Check that X and y have correct shape
12        X, y = check_X_y(X, y)
13        # Store the classes seen during fit
14        self.classes_ = unique_labels(y)
15        self.X_ = X
16        self.y_ = y
17        for _ in range(self.epoches):
18            self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
19        # Return the classifier
20        return self
21
22    def predict(self, X):
23        # Check is fit had been called
24        check_is_fitted(self, ['X_', 'y_'])
25        # Input validation
26        X = check_array(X)
27        y = self.Net.forward(X)
28        res = np.where(y < 0.5, 0, 1)
29        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:

```
1 class SoftMarginSVM:
2     def __init__(self, n, alpha):
3         self.alpha = alpha
4         mu = 0.0
5         sigma = 1.0 / np.sqrt(n)
6         self.W = np.random.normal(mu, sigma, (1, n + 1))
7
8     def forward(self, x):
9         z = np.dot(x, self.W.T)
10        return z
11
12    def add_ones(self, x):
13        ones = np.ones((x.shape[0], 1))
14        return np.hstack((x, ones))
15
16    def predict(self, x):
17        res = self.forward(self.add_ones(x))
18        return np.where(res < 0, 0, 1)
19
20    def train_epoch(self, x, y, batch_size = 100, step = 1e-7):
21        x = self.add_ones(x)
22        y = np.where(y > 0, 1, -1)
23        for i in range(0, len(x), batch_size):
24            xb = x[i:i + batch_size]
25            yb = y[i:i + batch_size]
26
27            pred = self.forward(xb)
28            grad = self.alpha * self.W
29            for i in range(len(xb)):
30                if (yb[i] * pred[i] < 1):
31                    grad -= yb[i] * xb[i]
32            self.W -= step * grad
```

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

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

```
1 class SVM(ClassifierMixin, BaseEstimator):
2     def __init__(self, epochs = 1, batch_size = 10, SGD_step = 0.001, alpha = 0.1):
3         self.epochs = epochs
4         self.batch_size = batch_size
5         self.SGD_step = SGD_step
6         self.alpha = alpha
7         self.Net = SoftMarginSVM(8, alpha)
8
9     def fit(self, X, y):
10         # Check that X and y have correct shape
11         X, y = check_X_y(X, y)
12         # Store the classes seen during fit
13         self.classes_ = unique_labels(y)
14
15         self.X_ = X
16         self.y_ = y
17         for _ in range(self.epochs):
18             self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
19         # Return the classifier
20         return self
21
22     def predict(self, X):
23         y = self.Net.predict(X)
24         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:

```
1 class NaiveBayes(ClassifierMixin, BaseEstimator):
2     def __init__(self):
3         None
4
5     def fit(self, X, y):
6         # Check that X and y have correct shape
7         X, y = check_X_y(X, y)
8
9         self.X_ = X
10        self.y_ = y
11
12        labels, cnts = np.unique(self.y_, return_counts = True)
13        self.labels = labels
14        self.p_of_y = np.array([elem / self.y_.shape[0] for elem in cnts])
15        self.means = np.array([self.X_[self.y_ == elem].mean(axis = 0) for elem in
16                               labels])
17        self.stds = np.array([self.X_[self.y_ == elem].std(axis = 0) for elem in labels
18                               ])
19        # Return the classifier
20        return self
21
22    def gaussian(self, mu, sigma, x0):
23        return np.exp(-(x0 - mu) ** 2 / (2 * sigma)) / np.sqrt(2.0 * pi * sigma)
24
25    def predict(self, X):
26        # Check is fit had been called
27        check_is_fitted(self, ['X_', 'y_'])
28
29        # Input validation
30        X = check_array(X)
31
32        res = np.zeros(X.shape[0])
33        for (i, elem) in enumerate(X):
34            p = np.array(self.p_of_y)
35            for (j, label) in enumerate(self.labels):
36                p_x_cond_y = np.array([self.gaussian(self.means[j][k], self.stds[j][k],
37                                                       elem[k]) for k in range(X.shape[1])])
38                p[j] *= np.prod(p_x_cond_y)
39            res[i] = np.argmax(p)
40        return res
```

5 Подбор гиперпараметров

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

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

1 | rscv = RandomizedSearchCV(Pipeline([("SVM", SVM())]),
2 |                            {"SVM__epoches" : [1, 2, 4],
3 |                             "SVM__batch_size" : [5, 10, 20],
4 |                             "SVM__SGD_step" : [0.01, 0.05, 0.1],
5 |                             "SVM__alpha" : [1.0, 0.1, 0.01, 0.0]})
6 | rscv.fit(train_X, train_y)
7 | 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

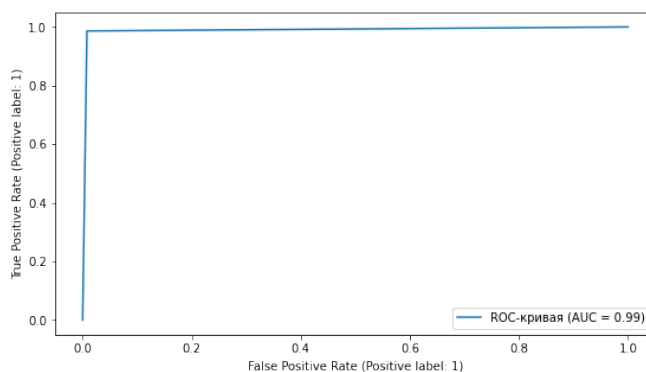
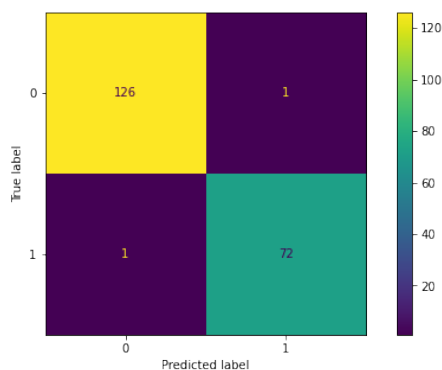
Лучшие гиперпараметры модели: 'knn__k': 3

Лучший счёт модели: 0.9950000000000001

Accuracy: 0.99

Recall: 0.9863013698630136

Precision: 0.9863013698630136



1.2 sklearn.neighbors.KNeighborsClassifier

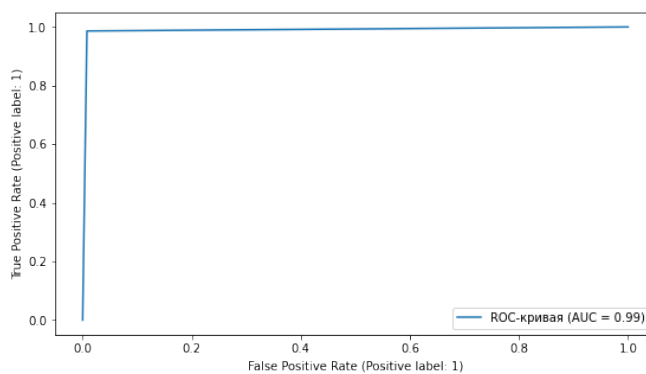
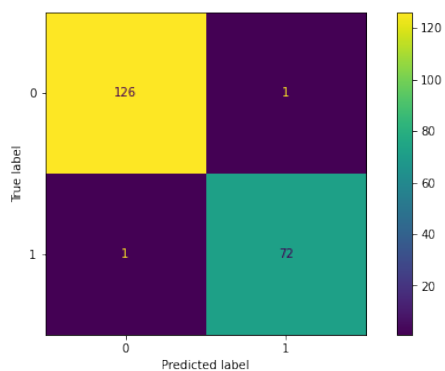
Лучшие гиперпараметры модели: 'knn__n_neighbors': 3

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,

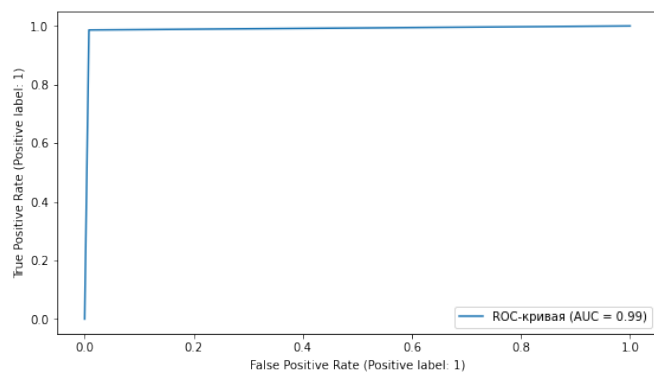
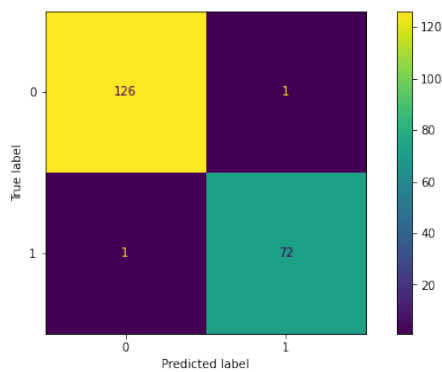
'logreg__batch_size': 5, 'logreg__epoches': 4

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',

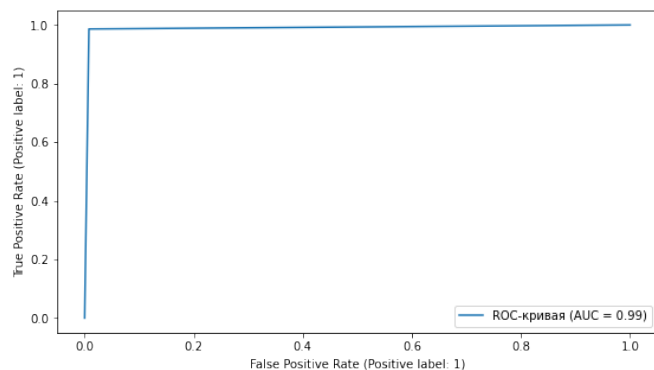
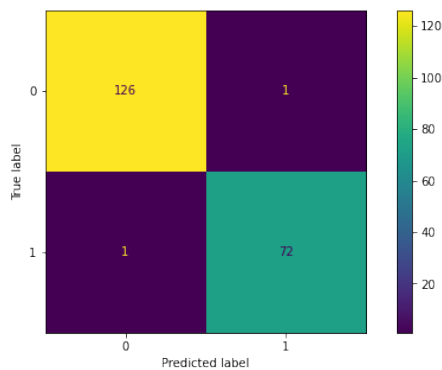
'logreg__solver': 'newton-cg'

Лучший счёт модели: 0.9950000000000001

Accuracy: 0.99

Recall: 0.9863013698630136

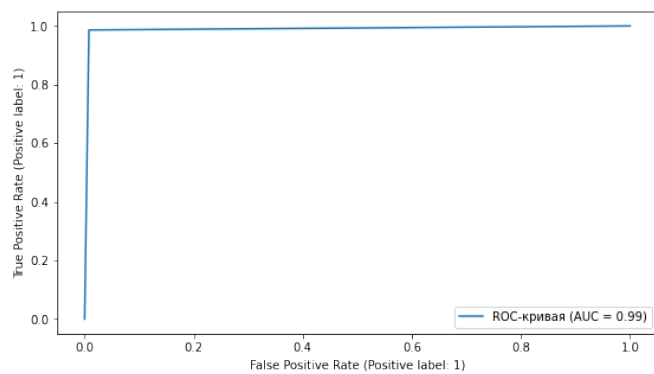
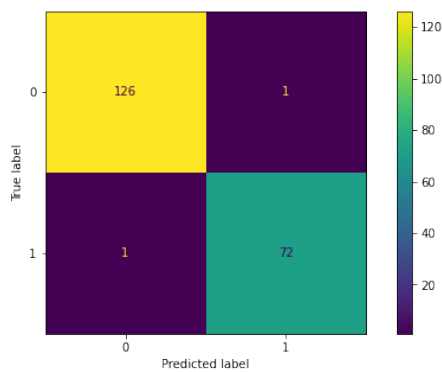
Precision: 0.9863013698630136



3 Метод опорных векторов

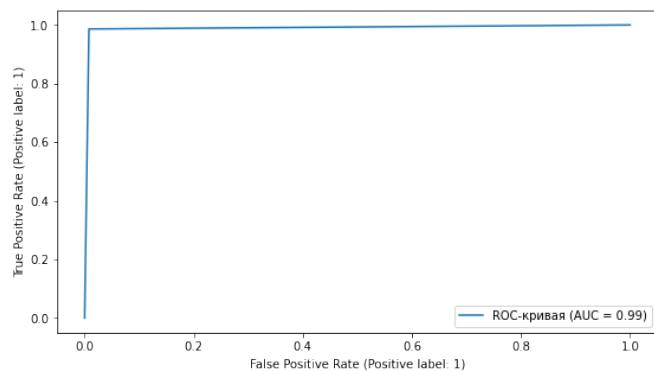
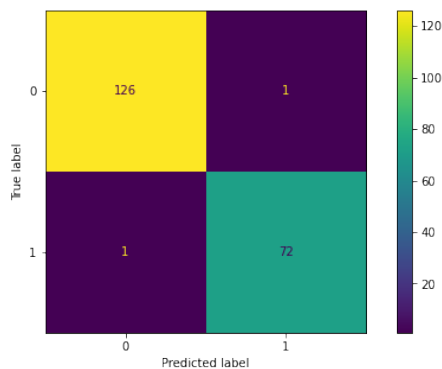
3.1 SVM

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.LinearSVC

Лучшие гиперпараметры модели: 'svc__loss': 'hinge', 'svc__max_iter': 10000.0
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136



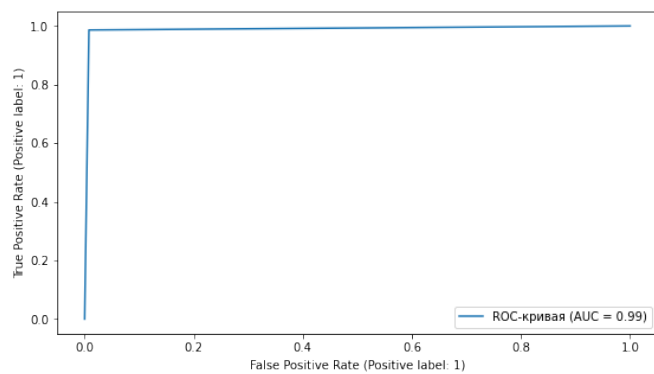
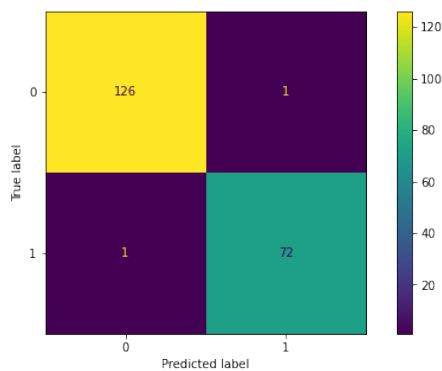
4 Наивный байесовский классификатор

4.1 NaiveBayes

Accuracy: 0.99

Recall: 0.9863013698630136

Precision: 0.9863013698630136

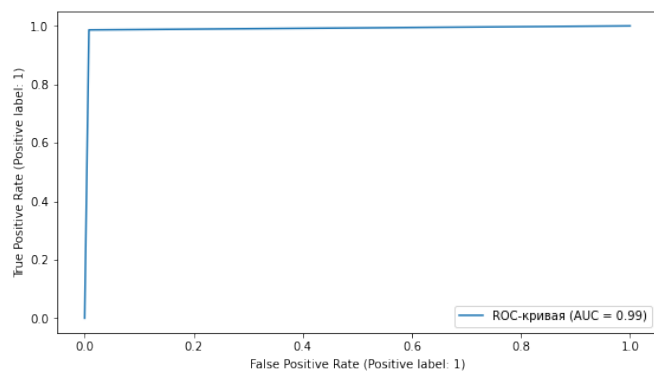
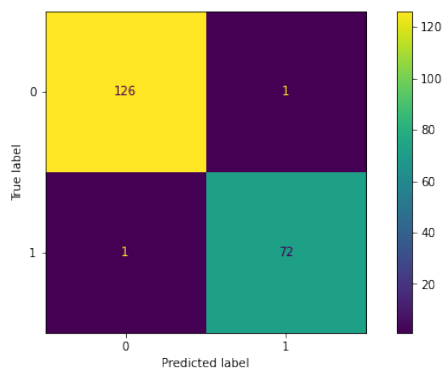


4.2 sklearn.naive_bayes.GaussianNB

Accuracy: 0.99

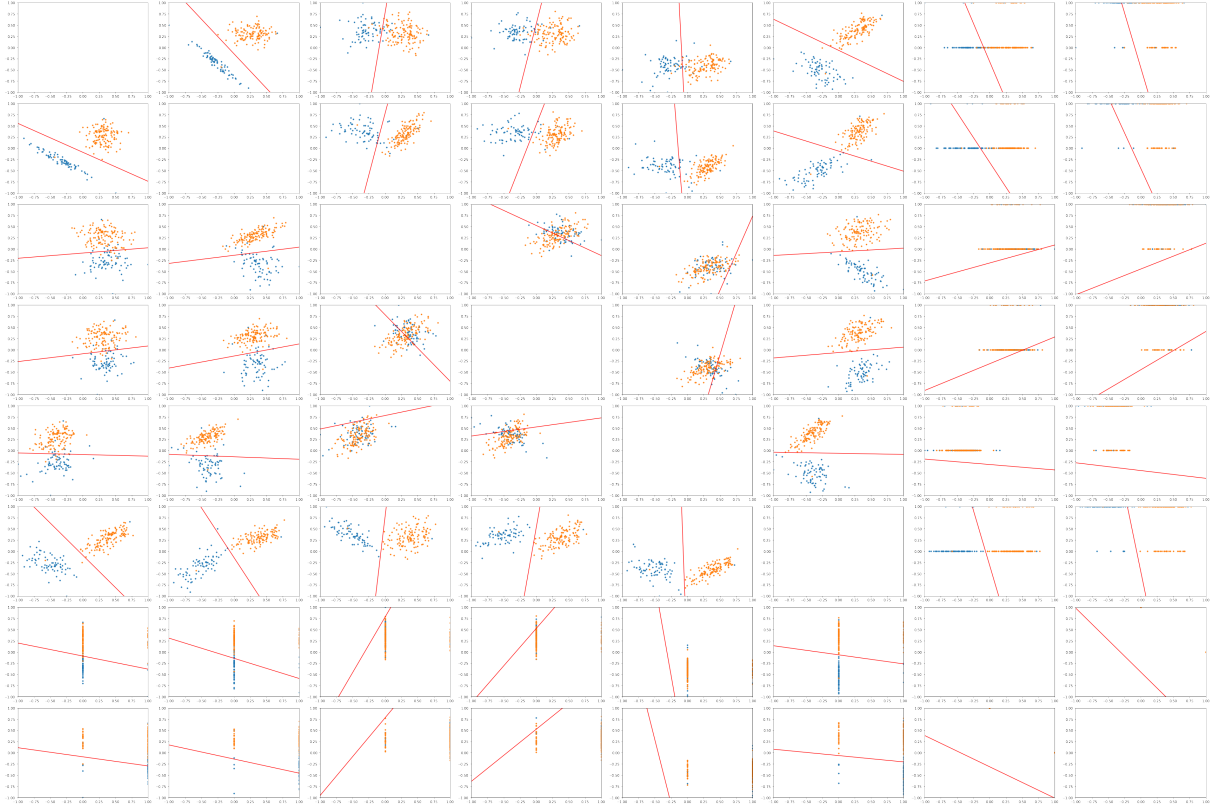
Recall: 0.9863013698630136

Precision: 0.9863013698630136

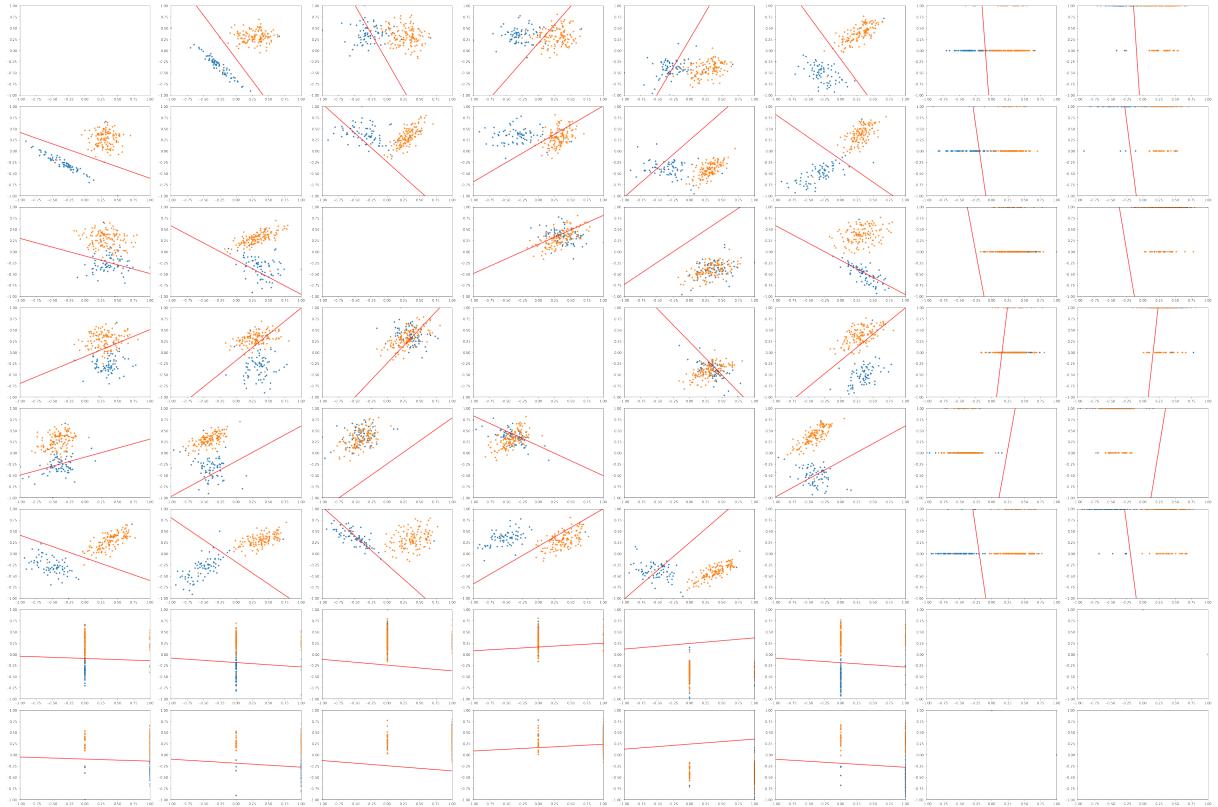


5 Разделяющая прямая для метода опорных векторов

5.1 Моя реализация



5.2 sklearn.svm.LinearSVC



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 всего 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 перемешаны 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).