Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Машинное обучение»

Линейные модели

Студент: Артамонов О. А.
Группа: М8О-308Б-19
Оценка: _____
Подпись: _____

Москва, 2022

Задание

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 (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

Обработка данных

Для начала необходимо подготовить данные для обучения. Категориальные признаи закодируем с помощью One Hot Encoding, количественные признаки приведем к стандартному нормальному распределению (среднее = 0, дисперсия = 1).
Затем посплитим данные на трейн и тест.

Подсчет метрик

Для оценивания качества моделей будем использовать метрики:

- Accuracy
- Precision
- Recall

и будем строить матрицу неточностей.

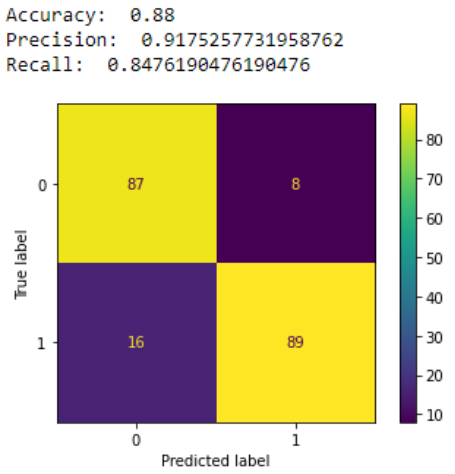
Вычислять все это мы будем с помощью функций Sklearn.

KNN

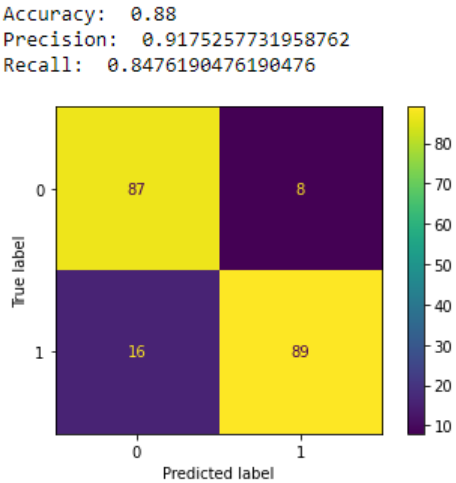
Обучение: запоминаем всю выборку
Предсказание: ищем k ближайших соседей для каждого объекта, выдаем тот класс, соседей которого больше.

```
class MyKNN(BaseEstimator, ClassifierMixin):
    def __init__(self, n_neighbors=5):
        self.n_neighbors = n_neighbors

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

    def predict(self, X):
        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.n_neighbors-1)
            nearest_neighbors = neighbors[:self.n_neighbors]
            labels, cnts = np.unique(self.y[nearest_neighbors],
return_counts=True)
            y[i] = labels[cnts.argmax()]
        return y
```

Результат моей реализации:



Результат реализации из Sklearn:



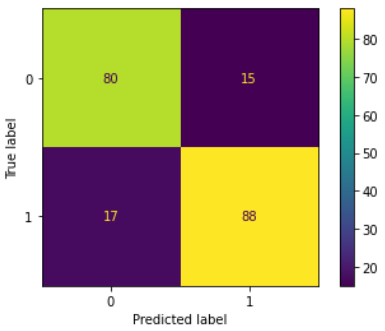
Logistic Regression

Обучение: подбираем коэффициенты линейной комбинации так, чтобы MSE была минимальной (с помощью градиентного спуска)
Предсказание: применяем полученную формулу ко всем объектам, затем навешиваем сигмоиду, чтобы перейти к вероятностям

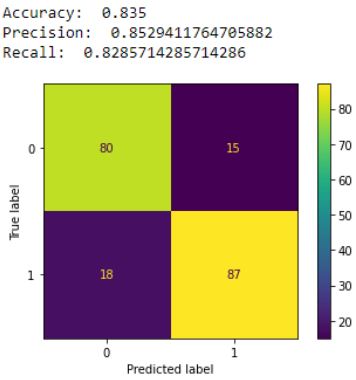
```
def sigmoid(a):
    return 1. / (1 + np.exp(-a))

class MyLogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, epochs=10, lr=0.1, batch_size=64):
        self.w = None
        self.epochs = epochs
        self.lr = lr
        self.batch_size = batch_size

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        n, k = X.shape
        self.w = np.random.randn(k + 1)

        # добавим смещение
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        for i in range(self.epochs):
            for j in range(0, len(X), self.batch_size):
                X_batch = X[j:j+self.batch_size]
                y_batch = y[j:j+self.batch_size]

                # градиентный спуск
                y_pred = sigmoid(np.dot(X_batch, self.w))
                self.w -= self.lr * (X_batch.T @ (y_pred - y_batch))
        return self

    def predict(self, X, threshold=0.5):
        X = check_array(X)
        n = X.shape[0]
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        return sigmoid(np.dot(X, self.w)) > threshold
```

Результат моей реализации:

```
Accuracy:  0.84
Precision:  0.8543689320388349
Recall:  0.8380952380952381
```



Результат реализации из Sklearn:



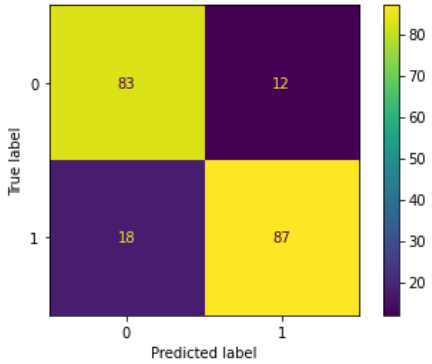
SVM

Обучение: строим разделяющую гиперплоскость
Предсказание: смотрим, с какой стороны от гиперплоскости находятся объекты.

```
class MySVM(ClassifierMixin, BaseEstimator):
    def __init__(self, epochs=10, lr=0.1, alpha=0.1):
        self.w = None
        self.epochs = epochs
        self.lr = lr
        self.alpha = alpha

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        y = np.where(y == 1, 1, -1) # переводим таргеты из 0/1 в -1/1
        n, k = X.shape
        self.w = np.random.randn(k + 1)

        # добавим смещение
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        for i in range(self.epochs):
            for j, x in enumerate(X):
                margin = y[j] * np.dot(self.w, x)
                if margin >= 1:
                    self.w -= self.lr * self.alpha * self.w /
self.epochs
                else:
                    self.w += self.lr * (y[j] * x - self.alpha * self.w
/ self.epochs)
        return self

    def predict(self, X):
        X = check_array(X)
        n, k = X.shape
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        y = np.ndarray((n))

        for i, elem in enumerate(X):
            prediction = np.dot(self.w, elem)
            y[i] = (prediction > 0)
        return y
```

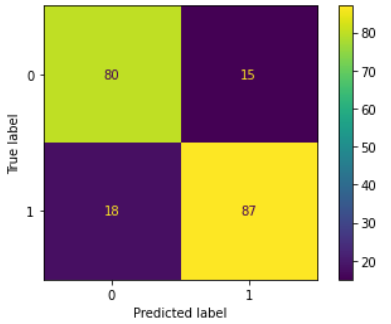
Результат моей реализации:

```
Accuracy:  0.85
Precision:  0.8787878787878788
Recall:  0.8285714285714286
```



Результат реализации из Sklearn:

```
Accuracy:  0.835
Precision:  0.8529411764705882
Recall:  0.8285714285714286
```



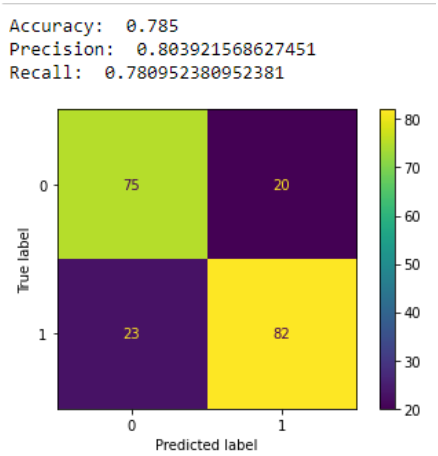
Naive Bayes

Обучение: запоминаем параметры распределений фичей
Предсказание: считаем условные вероятности принадлежности ко всем классам и берем класс с наибольшей вероятностью.

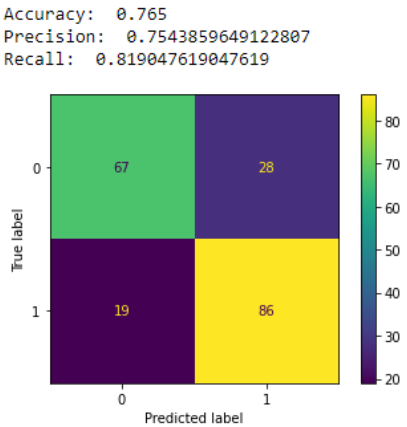
```
class MyNaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        labels, counts = np.unique(y, return_counts=True)
        self.labels = labels

        # запоминаем параметры распределения
        self.freq = np.array([cnt / y.shape[0] for cnt in counts])
        self.means = np.array([X[y == label].mean(axis = 0) for label
in labels])
        self.stds = np.array([X[y == label].std(axis = 0) for label in
labels])
        return self

    def predict(self, X):
        X = check_array(X)
```

```
        y = np.zeros(X.shape[0])
        for i, x in enumerate(X):
            cur_freq = np.array(self.freq)
            for j in range(len(self.labels)):
                # P(класс = j | X)
                p      =      np.array([self.gaussian(self.means[j][k],
self.stds[j][k], x[k]) for k in range(X.shape[1])])
                cur_freq[j] *= np.prod(p)
            y[i] = np.argmax(cur_freq)
        return y

    def gaussian(self, mu, sigma, x0):
        # X ~ N(mu, sigma)
        return np.exp(-(x0 - mu) ** 2 / (2 * sigma)) / np.sqrt(2.0 *
np.pi * sigma) # функция распределения
```

Результат моей реализации:



Результат реализации из Sklearn:



Вывод

В этой работе я релизовал 4 алгоритма машинного обучения: КНН, логистическую регрессию, СВМ и наивного Байеса. Каждую из этих моделей я обучил, провалидировал и сравнил с реализацией из Sklearn.

Результаты всех моих моделей совпали с результатами моделей из Sklearn (ну или незначительно отличаются из-за разных алгоритмов, рандом сидов итд).

Все модели показали примерно одинаковый результат. Лучше всего отработал KNN:
Accuracy: 0.88
Precision: 0.9175257731958762
Recall: 0.8476190476190476

Но остальные алгоритмы были ненамного хуже. Возможно, если бы в качестве порога вероятности мы бы использовали не 0.5, а какое-то другое число, то результаты были бы чуть-чуть получше.