

Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Лабораторная работа №2 по искусственному интеллекту

6 семестр

Студент: Павлова К.А.

Группа: М8О-406Б-18

Руководитель:

Дата:

Москва, 2021

Оглавление

Условие	3
Логистическая регрессия	4-6
Алгоритм	4
Реализация	5
Обучение и метрики	6
Метод опорных векторов	7-9
Алгоритм	7
Реализация	8
Обучение и метрики	9
Дерево принятия решений	10-14
Алгоритм	10
Реализация	11-13
Обучение и метрики	14

Постановка задачи:

Необходимо реализовать алгоритмы машинного обучения. Применить данные алгоритмы на наборы данных, подготовленных в первой лабораторной работе. Провести анализ полученных моделей, вычислить метрики классификатора. Произвести тюнинг параметров в случае необходимости. Сравнить полученные результаты с моделями реализованными в `scikit-learn`. Аналогично построить метрики классификации. Показать, что полученные модели не переобучились. Также необходимо сделать выводы о применимости данных моделей к вашей задаче. Задачи со звездочкой бьются по вариантам:

N по списку % 2 + 1.

- 1) ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ
- 2) *SVM - ПЕРВЫЙ ВАРИАНТ
- 3) ДЕРЕВО РЕШЕНИЙ
- 4) *RANDOM FOREST - ВТОРОЙ ВАРИАНТ

Логистическая регрессия

Алгоритм

В отличие от обычной регрессии, в методе логистической регрессии не производится предсказание значения числовой переменной исходя из выборки исходных значений. Вместо этого, значением функции является вероятность того, что данное исходное значение принадлежит к определенному классу. Для простоты, давайте предположим, что у нас есть только два класса и вероятность, которую мы будем определять, P_+ вероятности того, что некоторое значение принадлежит классу "+". И конечно $P_- = 1 - P_+$. Таким образом, результат логистической регрессии всегда находится в интервале $[0, 1]$.

Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено линейной границей (т.е. прямой) на две соответствующих классам области. Итак, что же имеется ввиду под линейной границей? В случае двух измерений — это просто прямая линия без изгибов. В случае трех — плоскость, и так далее. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма. Чтобы все работало, точки исходных данных должны разделяться линейной границей на две вышеупомянутых области. Если точки исходных данных удовлетворяют этому требованию, то их можно назвать линейно разделяемыми.

Реализация

Я реализовал логистическую регрессию в виде класса с 2 публичными методами. `fit` - для обучения, `predict` - для предсказания

```
class LogReg:
    #задаем количество итераций при инициализации класса 100000
    def __init__(self,num_iter = 100000):
        self.num_iter=num_iter
        self.beta=1

    #метод обучающий модель
    def fit(self,x,y):
        #задаем матрицу весов в виде единичной матрицы
        self.beta = np.ones(x.shape[1])
        for i in range(self.num_iter):
            h = self._sigmoid(x, self.beta)#считаем сигмоиду
            gradient = self._gradient_spusk(x, h, y)#спускаемся по градиенту
            self.beta =self._weight_update(self.beta, 0.1, gradient)#обновляем веса

    #приватный метод, считающий сигмоиду
    def _sigmoid(self,X, weight):
        z = np.dot(X, weight)
        return 1 / (1 + np.exp(-z))

    #приватная функция для градиентного шага
    def _gradient_spusk(self,X, H, Y):
        return np.dot(X.T, (H - Y)) / Y.shape[0]

    #приватная функция для обновления весов
    def _weight_update(self,weight, learning_rate, gradient):
        return weight - learning_rate * gradient

    def predict(self,test):
        final_result=[]

        #приминаем сигмоиду к тестовым данным
        result = self._sigmoid(test, self.beta)

        #выбираем метки для теста
        for i in result:
            final_result.append(self._onepred(i))

        return final_result

    #приватная функция для одного предсказания
    def _onepred(self,x):
        if x < 0.5:
            return 0
        else:
            return 1
```

Обучение и метрики

```
In [6]: 1 #обучаю свою модель
2 my_lg=LogReg()
3 my_lg.fit(X_train,y_train)
4 #делаем предсказания на трейне и на тесте и смотрим метрики
5 print('Метрики на обучающей выборки ')
6 metrics(my_lg.predict(X_train),y_train)
7 print('Метрики на тестовой выборки ')
8 metrics(my_lg.predict(X_test),y_test)

<ipython-input-5-f7a648a8a967>:20: RuntimeWarning: overflow encountered in exp
return 1 / (1 + np.exp(-z))
```

Метрики на обучающей выборки
Accuracy: 0.851652056641942
Precision: 0.851652056641942
Recall: 0.851652056641942
F1: 0.851652056641942
Метрики на тестовой выборки
Accuracy: 0.839622641509434
Precision: 0.839622641509434
Recall: 0.839622641509434
F1: 0.839622641509434

```
In [7]: 1 #обучаю модель из sklearn
2 sk_lg=LogisticRegression(max_iter=100000)
3 sk_lg.fit(X_train,y_train)
4 #делаем предсказания на трейне и на тесте и смотрим метрики
5 print('Метрики на обучающей выборки ')
6 metrics(sk_lg.predict(X_train),y_train)
7 print('Метрики на тестовой выборки ')
8 metrics(sk_lg.predict(X_test),y_test)
```

Метрики на обучающей выборки
Accuracy: 0.8577208361429535
Precision: 0.8577208361429535
Recall: 0.8577208361429535
F1: 0.8577208361429536
Метрики на тестовой выборки
Accuracy: 0.8443396226415094
Precision: 0.8443396226415094
Recall: 0.8443396226415094
F1: 0.8443396226415094

Выводы о моделях по метрикам

- Моя модель и модель из sklearn не переобучились, т к разница на метриках между трейном и тестом минимальна.
- Разница метрик на трейне и тесте между моей моделью и моделью из sklearn минимальна
- Обе модели показали достаточно неплохой результат примерно 0.85 по метрикам

Метод опорных векторов

Алгоритм

Алгоритм Главная цель SVM как классификатора — найти уравнение разделяющей гиперплоскости в пространстве, которая бы разделила два класса неким оптимальным образом. После настройки весов алгоритма (обучения), все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс.

Реализация

Я реализовал SVM в виде класса MYSVM с двумя публичными методами `fit`- для обучения, `predict` - для предсказания

```
class MYSVM(object):
    # при инициализации класса задается сразу _etha - шаг градиентного спуска, _alpha - коэффициент скорости
    # пропорционального уменьшения весов, _epochs - количество эпох обучения
    def __init__(self, etha=0.1, alpha=0.2, epochs=990):
        self._epochs = epochs
        self._etha = etha
        self._alpha = alpha
        self._w = None

    # метод для обучения модели
    def fit(self, X_train, Y_train):

        for i in range(len(Y_train)):
            if Y_train.iloc[i] == 0:
                Y_train.iloc[i] = -1

        # добавляем в конец каждого вектора число 1
        X_train = self._add_bias_feature(X_train)
        self._w = np.random.normal(loc=0, scale=0.05, size=X_train.shape[1]) # задаем первые веса

        for epoch in range(self._epochs):

            for i, x in enumerate(X_train):
                margin = Y_train.iloc[i] * np.dot(self._w, X_train[i])
                if margin >= 1: # классифицируем верно
                    self._w = self._w - self._etha * self._alpha * self._w / self._epochs
                else: # классифицируем неверно или попадаем на полосу разделения при 0 < margin < 1
                    self._w = self._w + \
                        self._etha * (Y_train.iloc[i] * X_train[i] - self._alpha * self._w / self._epochs)

            for i in range(len(Y_train)):
                if Y_train.iloc[i] == -1:
                    Y_train.iloc[i] = 0

        # приватный метод, добавляющий в конец каждого вектора число 1
        def _add_bias_feature(self, a):

            a_extended = np.zeros((a.shape[0], a.shape[1] + 1))
            a_extended[:, :-1] = a
            a_extended[:, -1] = int(1)
            return a_extended

        # метод для предсказания
        def predict(self, X):
            y_pred = []
            # X_extended = self._add_bias_feature(X)
            for i in range(len(X)):
                y_pred.append(np.sign(1 + np.dot(self._w[1:], X.iloc[i])))
            for i in range(len(y_pred)):
                if y_pred[i] == -1:
                    y_pred[i] = 0

            return y_pred
```

Обучение и метрики

```
In [9]: 1 my_svm=MYsvm()  
2 my_svm.fit(X_train,y_train)  
3 print('метрики на обучении')  
4 metrics(my_svm.predict(X_train),y_train)  
5 print('метрики на тесте')  
6 metrics(my_svm.predict(X_test),y_test)
```

```
метрики на обучении  
Accuracy: 0.8479433580579906  
Precision: 0.8479433580579906  
Recall: 0.8479433580579906  
F1: 0.8479433580579906  
метрики на тесте  
Accuracy: 0.839622641509434  
Precision: 0.839622641509434  
Recall: 0.839622641509434  
F1: 0.839622641509434
```

```
In [10]: 1 sk_svm = svm.SVC()  
2 sk_svm.fit(X_train, y_train)  
3 print('метрики на обучении')  
4 metrics(sk_svm.predict(X_train),y_train)  
5 print('метрики на тесте')  
6 metrics(sk_svm.predict(X_test),y_test)
```

```
метрики на обучении  
Accuracy: 0.851652056641942  
Precision: 0.851652056641942  
Recall: 0.851652056641942  
F1: 0.851652056641942  
метрики на тесте  
Accuracy: 0.839622641509434  
Precision: 0.839622641509434  
Recall: 0.839622641509434  
F1: 0.839622641509434
```

Выводы

- Моя модель не переобучилась, т к разница на метриках между трейном и тестом минимальна.
- Моделт из sklearn не переобучилась, т к разница на метриках между трейном и тестом минимальна.
- Моя модель показывает себя хуже по метрикам на трейне чем модель из sklearn на трейне, но при этом моя модель показывает такие же метрики как и sklearn на тесте.

Дерево принятия решений

Алгоритм

Дерево решений представляет собой иерархическую древовидную структуру, состоящую из правила вида «Если ..., то ...». За счет обучающего множества правила генерируются автоматически в процессе обучения. Правила генерируются за счет обобщения множества отдельных наблюдений (обучающих примеров), описывающих предметную область. Поэтому их называют индуктивными 10 правилами, а сам процесс обучения — индукцией деревьев решений. В обучающем множестве для примеров должно быть задано целевое значение, так как деревья решений — модели, создаваемые на основе обучения с учителем

Реализация

Я реализовать дерево решений в виде класса MyDT с двумя публичными методами , fit - для обучения, predict - для предсказания остальные методы приватные и используются в публичных

```
class MyDT():  
    # объявляем характеристики класса  
    def __init__(self, max_depth=3, min_size=10):  
        self.max_depth = max_depth  
        self.min_size = min_size  
        self.value = 0  
        self.feature_idx = -1  
        self.feature_threshold = 0  
        self.left = None  
        self.right = None
```

Реализация метода fit

```

# процедура обучения - сюда передается обучающая выборка
def fit(self, X, y):

    for i in range(len(y)):
        if y.iloc[i] == 0:
            y.iloc[i] = -1

    # начальное значение - среднее значение y
    self.value = y.mean()
    # начальная ошибка - mse между значением в листе (пока нет
    # разбиения, это среднее по всем объектам) и объектами
    base_error = ((y - self.value) ** 2).sum()
    error = base_error
    flag = 0

    # пришли в максимальную глубину
    if self.max_depth <= 1:
        return

    dim_shape = X.shape[1]

    left_value, right_value = 0, 0

    for feat in range(dim_shape):

        prev_error1, prev_error2 = base_error, 0
        if feat==0:
            idxs = np.argsort(X[:, feat])

        # переменные для быстрого переброса суммы
        mean1, mean2 = y.mean(), 0
        sm1, sm2 = y.sum(), 0

        N = X.shape[0]
        N1, N2 = N, 0
        thres = 1

        while thres < N - 1:
            N1 -= 1
            N2 += 1

            idx = idxs[thres]
            x = X[int(idx), feat]

            # вычисляем дельты - по ним в основном будет делаться переброс
            delta1 = (sm1 - y.iloc[idx]) * 1.0 / N1 - mean1
            delta2 = (sm2 + y.iloc[idx]) * 1.0 / N2 - mean2

            # увеличиваем суммы
            sm1 -= y.iloc[idx]
            sm2 += y.iloc[idx]

            # пересчитываем ошибки за O(1)
            prev_error1 += (delta1**2) * N1
            prev_error1 -= (y.iloc[idx] - mean1)**2
            prev_error1 -= 2 * delta1 * (sm1 - mean1 * N1)
            mean1 = sm1/N1

            prev_error2 += (delta2**2) * N2
            prev_error2 += (y.iloc[idx] - mean2)**2
            prev_error2 -= 2 * delta2 * (sm2 - mean2 * N2)
            mean2 = sm2/N2

            # пропускаем близкие друг к другу значения
            if thres < N - 1 and np.abs(x - X[idxs[thres + 1], feat]) < 1e-5:
                thres += 1
                continue

            # 2 условия, чтобы осуществить сплит - уменьшение ошибки
            # и минимальное кол-о эл-в в каждом листе
            if (prev_error1 + prev_error2 < error):
                if (min(N1,N2) > self.min_size):

                    # переопределяем самый лучший признак и границу по нему
                    self.feature_idx, self.feature_threshold = feat, x
                    # переопределяем значения в листах
                    left_value, right_value = mean1, mean2

                    # флаг - значит сделали хороший сплит
                    flag = 1
                    error = prev_error1 + prev_error2

            thres += 1

    # ничего не разделили, выходим
    if self.feature_idx == -1:
        return

    self.left = MyDT(self.max_depth - 1)
    # print ("Левое поддерево с глубиной %d"%(self.max_depth - 1))
    self.left.value = left_value
    self.right = MyDT(self.max_depth - 1)
    # print ("Правое поддерево с глубиной %d"%(self.max_depth - 1))
    self.right.value = right_value

    idxs_l = (X[:, self.feature_idx] > self.feature_threshold)
    idxs_r = (X[:, self.feature_idx] <= self.feature_threshold)

    self.left.fit(X[idxs_l, :], y[idxs_l])
    self.right.fit(X[idxs_r, :], y[idxs_r])

    for i in range(len(y)):
        if y.iloc[i]==-1:
            y.iloc[i]=0

```

Реализация метода predict и приватный методов __predict ,
_prediction , используемых в predict

```
def __predict(self, x):
    if self.feature_idx == -1:
        return self.value

    if x[self.feature_idx] > self.feature_threshold:
        return self.left.__predict(x)
    else:
        return self.right.__predict(x)

#метод для финального расставления меток
def _prediction(self, x):
    if x < 0:
        return 0
    else:
        return 1

#Метод для предсказания
def predict(self, X):
    y = np.zeros(X.shape[0])

    for i in range(X.shape[0]):
        y[i] = self.__predict(X[i])

    for i in range(len(y)):
        y[i] = self._prediction(y[i])
    return y
```

Обучение и метрики

```
In [12]: 1 my_dt=MyDT()
2 my_dt.fit(X_train.values,y_train)
3 print('метрики на обучении')
4 metrics(my_dt.predict(X_train.values),y_train)
5 print('метрики на тесте')
6 metrics(my_dt.predict(X_test.values),y_test)
```

```
метрики на обучении
Accuracy: 0.851652056641942
Precision: 0.851652056641942
Recall: 0.851652056641942
F1: 0.851652056641942
метрики на тесте
Accuracy: 0.839622641509434
Precision: 0.839622641509434
Recall: 0.839622641509434
F1: 0.839622641509434
```

```
In [13]: 1 dt = DecisionTreeClassifier(max_depth=3, min_samples_leaf=10)
2 dt.fit(X_train,y_train)
3 print('метрики на обучении')
4 metrics(dt.predict(X_train.values),y_train)
5 print('метрики на тесте')
6 metrics(dt.predict(X_test.values),y_test)
```

```
метрики на обучении
Accuracy: 0.8550236008091706
Precision: 0.8550236008091706
Recall: 0.8550236008091706
F1: 0.8550236008091706
метрики на тесте
Accuracy: 0.8388364779874213
Precision: 0.8388364779874213
Recall: 0.8388364779874213
F1: 0.8388364779874213
```

Выводы

- Моя модель практически не переобучилась, т к разница на метриках между трейном и тестом минимальна.
- Моделт из sklearn практически не переобучилась, т к разница на метриках между трейном и тестом минимальна.
- Моя модель показывает себя хуже по метрикам на трейне чем модель из sklearn на трейне(0.004) и на тесте (0.001)