Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Искусственный интеллект»

Студент: В. И. Лобов
Преподаватель: С. Х. Ахмед
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №2

Задача: Требуется реализовать класс на выбранном языке программирования, который реализует один из алгоритмов машинного обучения.

Обязательным является наличия в классе двух методов fit, predict. Необходимо проверить работу вашего алгоритма на ваших данных, произведя необходимую подготовку данных. Также необходимо реализовать алгоритм полиномиальной регрессии, для предсказания значений в таблице. Сравнить результаты с стандартной реализацией sklearn, определить в чем сходство и различие ваших алгоритмов. Замерить время работы алгоритмов.

Номер по списку: 9

Вариант алгоритма: $(9\%6) + 1 = 4$. Рандомизированный лес

1 Описание

Алгоритм построения случайного леса:

1. Сгенерируем случайную подвыборку размером N из обучающей выборки
2. Построим дерево решений, строящее регрессию по образцам данной подвыборки, причём в ходе создания очередного узла дерева будем выбирать наилучший признак, относительно которого будет происходить разбиение. Критерием может выступать среднеквадратическая ошибка.
3. Дерево строится до полного исчерпания подвыборки или до заданной максимальной высоты.

Предсказание значения происходит так:

1. Каждое дерево из леса предсказывает своё значение
2. В качестве результата регрессии берётся среднее значение по всем деревьям

Оптимальное количество деревьев для конкретной задачи подбирается таким образом, чтобы минимизировать ошибку на тестовой выборке.

Чем выше глубина дерева, тем более точно алгоритм может предсказывать значения по выборке, но при неограниченном росте глубины появляется склонность к переобучению.

2 Исходный код

Реализуем класс DecisionTree, строящий регрессию по заданной выборке и класс RandomForest, являющийся ансамблем классов DecisionTree.

```
import numpy as np
import pandas as pd
import random
import time
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor


class RandomForest():
    def __init__(self, max_depth=5, n_trees=3, sample_size=0.1):
        self.trees = []
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.sample_size = sample_size

    def fit(self, X, y):
        self.trees = []
        for i in range(self.n_trees):
            X_sample, y_sample = self.subsample(X, y, self.sample_size)
            tree = DecisionTree(max_depth=self.max_depth)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def predict(self, X):
        predictions = np.zeros(len(X))

        for tree in self.trees:
            pred = tree.predict(X)
            predictions += pred
        predictions /= len(self.trees)

        return predictions

    def subsample(self, X, y, ratio):
        sample = []
        sample_y = []
        n_sample = round(len(X) * ratio)
        while len(sample) < n_sample:
            index = random.randrange(len(X))
            sample.append(X[index])
            sample_y.append(y[index])
        return np.array(sample), np.array(sample_y)

```

```
45
46 class DecisionTree():
47     def __init__(self, max_depth=5):
48         self.feature = None
49         self.label = None
50         self.n_samples = None
51         self.gain = None
52         self.root = None
53         self.left = None
54         self.right = None
55         self.threshold = None
56         self.depth = 0
57         self.max_depth = max_depth
58
59     def fit(self, features, target):
60         self.root = DecisionTree()
61         self.root.build(features, target)
62         self.root.prune(self.max_depth, self.root.n_samples)
63
64     def predict(self, features):
65         return np.array([self.root.predict_feature(feature) for feature in features])
66
67     def predict_feature(self, feature):
68         if self.feature != None:
69             if feature[self.feature] <= self.threshold:
70                 return self.left.predict_feature(feature)
71             else:
72                 return self.right.predict_feature(feature)
73         else:
74             return self.label
75
76     def build(self, features, target):
77         self.n_samples = features.shape[0]
78
79         if len(np.unique(target)) == 1:
80             self.label = target[0]
81             return
82
83         best_gain = 0.0
84         best_feature = None
85         best_threshold = None
86
87         self.label = np.mean(target)
88
89         impurity_node = self.mse(target)
90
91         for col in range(features.shape[1]):
92             feature_level = np.unique(features[:, col])
93             thresholds = (feature_level[:-1] + feature_level[1:]) / 2.0
```

```
94
95              for threshold in thresholds:
96                  target_l = target[features[:, col] <= threshold]
97                  impurity_l = self.mse(target_l)
98                  n_l = float(target_l.shape[0]) / self.n_samples
99
100                 target_r = target[features[:, col] > threshold]
101                 impurity_r = self.mse(target_r)
102                 n_r = float(target_r.shape[0]) / self.n_samples
103
104                 impurity_gain = impurity_node - (n_l * impurity_l + n_r * impurity_r)
105                 if impurity_gain > best_gain:
106                     best_gain = impurity_gain
107                     best_feature = col
108                     best_threshold = threshold
109
110         self.feature = best_feature
111         self.gain = best_gain
112         self.threshold = best_threshold
113         self.split_node(features, target)
114
115     def split_node(self, features, target):
116         features_l = features[features[:, self.feature] <= self.threshold]
117         target_l = target[features[:, self.feature] <= self.threshold]
118         self.left = DecisionTree()
119         self.left.depth = self.depth + 1
120         self.left.build(features_l, target_l)
121
122         features_r = features[features[:, self.feature] > self.threshold]
123         target_r = target[features[:, self.feature] > self.threshold]
124         self.right = DecisionTree()
125         self.right.depth = self.depth + 1
126         self.right.build(features_r, target_r)
127
128     def mse(self, target):
129         return np.mean((target - np.mean(target)) ** 2)
130
131     def prune(self, max_depth, n_samples):
132         if self.feature is None:
133             return
134
135         self.left.prune(max_depth, n_samples)
136         self.right.prune(max_depth, n_samples)
137
138         if self.depth >= max_depth:
139             self.left = None
140             self.right = None
141             self.feature = None
142
```

```
143
144 if __name__ == "__main__":
145     data = pd.read_csv("C:\\Users\\pkmixer\\Downloads\\AI\\Stocks\\aapl.us.txt")
146     X = np.array(data.drop(columns='Date'))
147     y = np.array(data.pop('Close'))
148
149     X, X_test, y, y_test = train_test_split(X, y, test_size=0.3)
150
151     X = np.array(X)
152     X_test = np.array(X_test)
153     y = np.array(y)
154     y_test = np.array(y_test)
155
156     for n_trees in [3, 5, 10, 50, 100]:
157         begin = time.time()
158
159         reg = RandomForest(max_depth=5, n_trees=n_trees)
160         reg.fit(X, y)
161         y_pred = reg.predict(X_test)
162
163         end = time.time()
164
165         time1 = end - begin
166
167         begin = time.time()
168
169         sk_reg = RandomForestRegressor(max_depth=5, n_estimators=n_trees)
170         sk_reg.fit(X, y)
171         sk_y_pred = sk_reg.predict(X_test)
172
173         end = time.time()
174
175         time2 = end - begin
176
177         print("{}{:<25s}".format(n_trees, " trees"), "mine ", "sklearn")
178         print("{:<26s}{} {}".format("time: ", round(time1, 4), round(time2, 4)))
179         print("{:<25s}".format("Explained_variance_score:"), round(metrics.
                explained_variance_score(y_test, y_pred), 4),
180               round(metrics.explained_variance_score(y_test, sk_y_pred), 4))
181         print("{:<25s}".format("Mean absolute error:"), round(metrics.
                mean_absolute_error(y_test, y_pred), 4),
182               round(metrics.mean_absolute_error(y_test, sk_y_pred), 4))
183         print("{:<25s}".format("Mean squared error:"), round(metrics.mean_squared_error
                (y_test, y_pred), 4),
184               round(metrics.mean_squared_error(y_test, sk_y_pred), 4))
185         print("{:<25s}".format("Mean squared log error:"), round(metrics.
                mean_squared_log_error(y_test, y_pred), 4),
186               round(metrics.mean_squared_log_error(y_test, sk_y_pred), 4))
```

```
187         print("{:<25s}".format("Median absolute error:"), round(metrics.
                median_absolute_error(y_test, y_pred), 4),
188               round(metrics.median_absolute_error(y_test, sk_y_pred), 4))
189         print("{:<25s}".format("R2 score:"), round(metrics.r2_score(y_test, y_pred), 4)
                ,
190               round(metrics.r2_score(y_test, y_pred), 4))
```

В качестве датасета будем использовать котировки акций компании Apple. По значениям цены открытия, минимальной цены, максимальной цены и обороте за день попробуем предсказать цену на момент закрытия.

```
Date,Open,High,Low,Close,Volume,OpenInt
1984-09-07,0.42388,0.42902,0.41874,0.42388,23220030,0
1984-09-10,0.42388,0.42516,0.41366,0.42134,18022532,0
1984-09-11,0.42516,0.43668,0.42516,0.42902,42498199,0
1984-09-12,0.42902,0.43157,0.41618,0.41618,37125801,0
1984-09-13,0.43927,0.44052,0.43927,0.43927,57822062,0
```

Сравним результаты работы данного алгоритма с алгоритмом, реализованным в модуле sklearn. Для оценки будем использовать различные критерии, такие как среднеквадратическая ошибка, R^2, коэффициент вариаций и т.д.. Также сравним время работы данных алгоритмов. Будем использовать разное количество деревьев в лесу. Отношение размера тестовой выборки к обучающей 3:7.

```
3 trees                    mine  sklearn
time:                     2.674 0.0199
Explained_variance_score: 0.9994 0.9995
Mean absolute error:      0.4981 0.481
Mean squared error:       0.8714 0.7225
Mean squared log error:   0.0088 0.0098
Median absolute error:    0.2509 0.2749
R2 score:                 0.9994 0.9994
5 trees                    mine  sklearn
time:                     4.4157 0.0339
Explained_variance_score: 0.9995 0.9995
Mean absolute error:      0.4728 0.4799
Mean squared error:       0.7861 0.6821
Mean squared log error:   0.0084 0.0096
Median absolute error:    0.239 0.2672
R2 score:                 0.9995 0.9995
10 trees                    mine  sklearn
time:                     8.8158 0.0663
```

```
Explained_variance_score: 0.9995 0.9996
Mean absolute error:      0.413 0.4351
Mean squared error:       0.6737 0.5331
Mean squared log error:   0.0082 0.0096
Median absolute error:    0.2107 0.2541
R2 score:                 0.9995 0.9995
50 trees                    mine  sklearn
time:                     44.2026 0.3248
Explained_variance_score: 0.9996 0.9997
Mean absolute error:      0.38 0.3976
Mean squared error:       0.5465 0.4365
Mean squared log error:   0.008 0.0095
Median absolute error:    0.2172 0.2477
R2 score:                 0.9996 0.9996
100 trees                    mine  sklearn
time:                     87.8184 0.6451
Explained_variance_score: 0.9997 0.9997
Mean absolute error:      0.3855 0.3939
Mean squared error:       0.5125 0.4363
Mean squared log error:   0.0084 0.0095
Median absolute error:    0.2309 0.2484
R2 score:                 0.9997 0.9997
```

Как видно из результатов тестирования, в среднем оценки предсказанных значений приблизительно равны у моей реализации и реализации в sklearn. Значительно отличается время работы: в том и другом случае оно линейно зависит от числа деревьев, но время построения одного дерева отличается примерно в 130 раз. Это можно объяснить тем, что некоторая часть кода в данном модуле написана на Cython. В целом, результат довольно приемлимый.