НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:
«Лабораторная работа №3»

Выполнил: Осипов Лев,
студент группы 301ПИ (1).

Москва, 2015 г.

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

ЗАДАНИЕ 1

Каждую вершину арностью k можно разбить на дерево глубиной ln(k), если разбить пополам число потомков и присвоить признак принадлежности к какой-либо из этих половин.

ЗАДАНИЕ 2

В задаче регрессии фигурируют не классы, а значения целевой функции. Можно попробовать разделить их на группы. Таким образом, дерево будет определять принадлежность не к классам, а к диапазонам значений целевой функции. Обучив алгоритм, мы сможем примерно предсказать эти значения.

ЗАДАНИЕ 3

Очевидно, что глубина дерева будет 2^n.

Так как на выходе алгоритма может быть K ответов, количество деревьев будет K^{2^n}.

ПРАКТИЧЕСКАЯ ЧАСТЬ

ЗАДАНИЕ 1

Для решения задания были визуализированы разделяющие поверхности для результатов работы двух алгоритмов (метод ближайшего соседа kNN и решающего дерева Tree) на сгенерированных выборках.

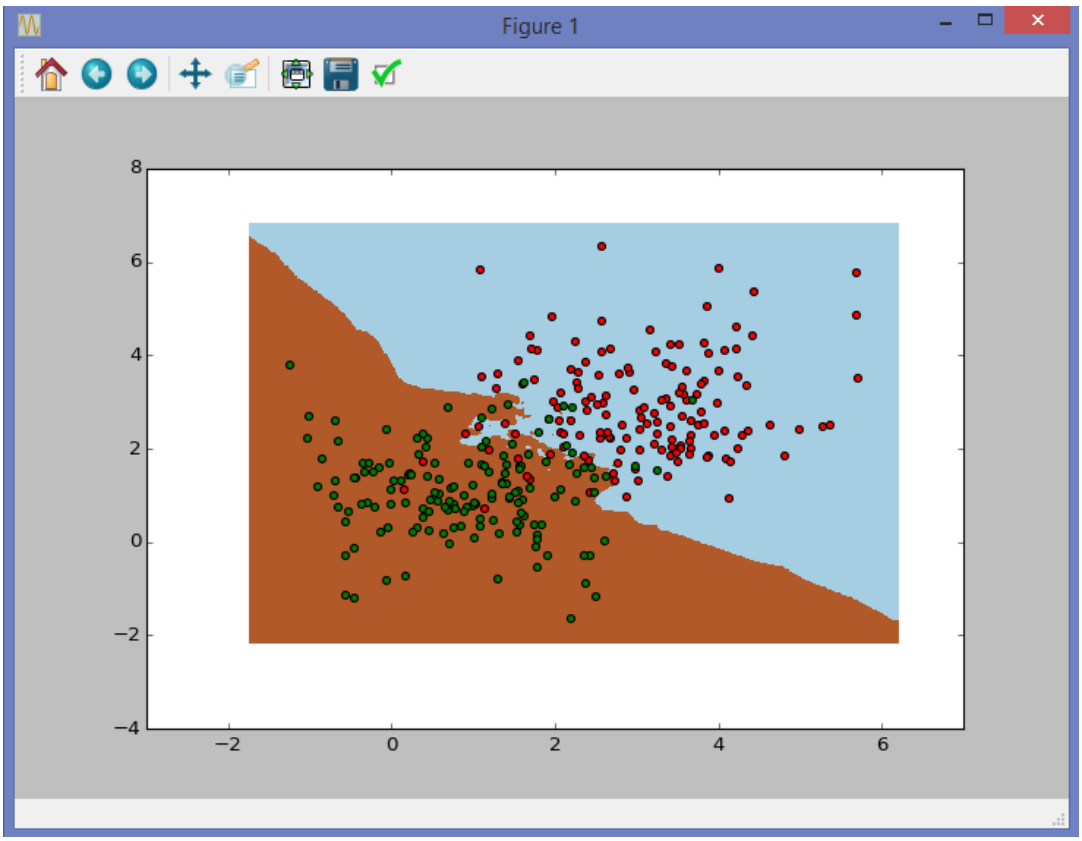

Рис. 1. KNN

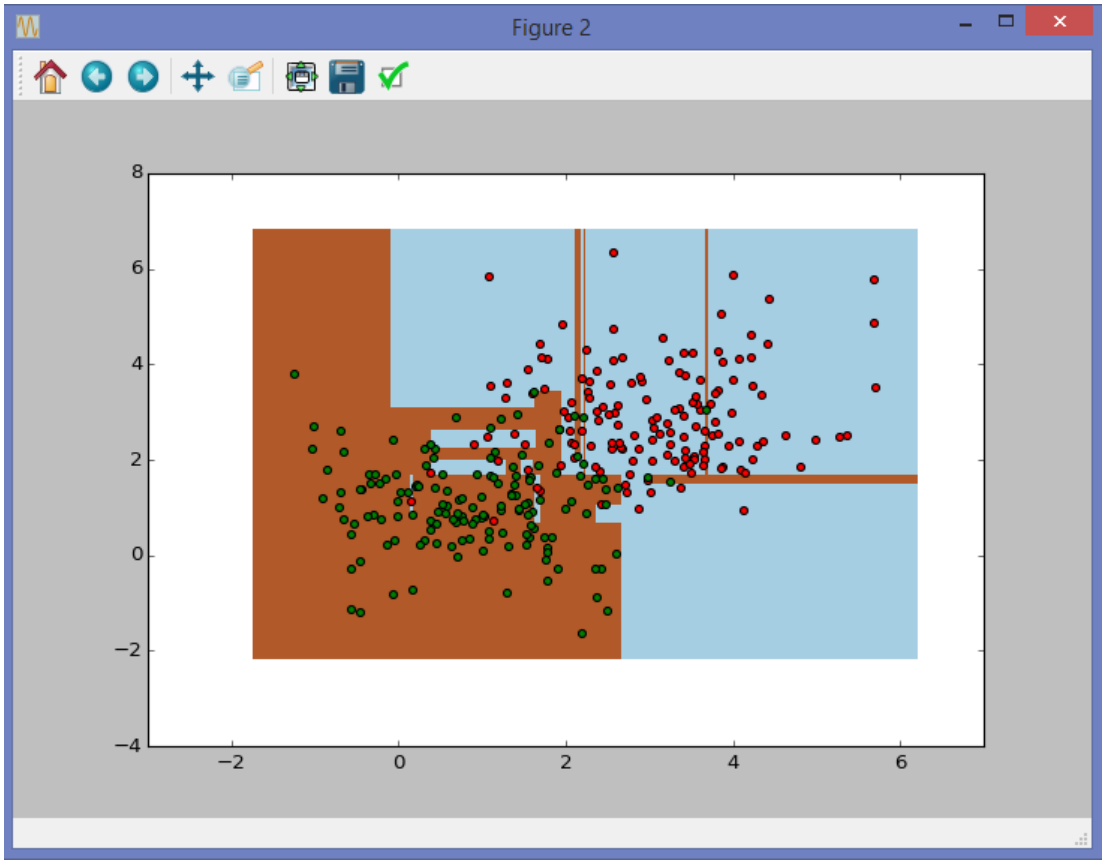

Рис. 2. Tree

По результатам программы видно, что разделяющая поверхность, полученная алгоритмом ближайшего соседа (Рис. 1), имеет произвольную форму, тогда как фигура, полученная алгоритмом решающего дерева (Рис. 2), имеет форму прямоугольного многоугольника.

ЗАДАНИЕ 2

Для решения задания были произведены сравнения результатов работы двух алгоритмов (метод ближайшего соседа kNN и решающего дерева Tree) на выборке с учетом номинальных признаков и без них.

```
KNN:
Maximum score: 0.764444444444 ( neighbours count = 17 )
Scores:
Score: 0.763333333333 ( neighbours count = 15 )
Score: 0.761111111111 ( neighbours count = 16 )
Score: 0.764444444444 ( neighbours count = 17 )
Score: 0.761111111111 ( neighbours count = 18 )
Score: 0.762222222222 ( neighbours count = 19 )
Tree:
Maximum score: 0.807777777778 ( maximum depth = 7 , minimum leafs = 8 )
Scores:
Score: 0.797777777778 ( maximum depth = 6 , minimum leafs = 7 )
Score: 0.795555555556 ( maximum depth = 6 , minimum leafs = 8 )
Score: 0.804444444444 ( maximum depth = 6 , minimum leafs = 9 )
Score: 0.805555555556 ( maximum depth = 7 , minimum leafs = 7 )
Score: 0.807777777778 ( maximum depth = 7 , minimum leafs = 8 )
Score: 0.807777777778 ( maximum depth = 7 , minimum leafs = 9 )
Score: 0.791111111111 ( maximum depth = 8 , minimum leafs = 7 )
Score: 0.786666666667 ( maximum depth = 8 , minimum leafs = 8 )
Score: 0.795555555556 ( maximum depth = 8 , minimum leafs = 9 )
```

Рис. 3. Результаты без номинальных признаков.

По результатам работы алгоритмов на выборке без номинальных признаков (Рис. 3) видно, что при небольшом изменении параметров алгоритма значения точности различаются относительно слабо (в пределах десятых долей процента) в обоих алгоритмов. Также стоит отметить, что алгоритм решающего дерева показал себя лучше.

```
One-hot encoding score (KNN):  0.813333333333
One-hot encoding score (tree): 0.786666666667
```

Рис. 4. Результаты с номинальными признаков.

По результатам работы алгоритмов на выборке с номинальными признаками (Рис. 4) видно, что метод ближайшего соседа сработал существенно быстрее (относительно, разумеется). Так как номинальных признаков у объектов немало, очевидно, что для решения этой задачи именно этот алгоритм лучше.

СПИСОК ЛИТЕРАТУРЫ

1) **Анализ данных (Программная инженерия)** – http://wiki.cs.hse.ru/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%28%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F%29#.D0.9E.D1.84.D0.BE.D1.80.D0.BC.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5_.D0.BF.D0.B8.D1.81.D0.B5.D0.BC

ТЕКСТ ПРОГРАММЫ

```
__author__ = 'Lev Osipov'

import numpy as np
import pylab as pl
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split


# Task 1

n = 150
covariance = [[1, 0], [0, 1]]
expected1 = (3, 3)
expected2 = (1, 1)
sample1 = np.random.multivariate_normal(expected1, covariance,
n)
sample2 = np.random.multivariate_normal(expected2, covariance,
n)
all_values = np.concatenate((sample1, sample2), axis=0)
labels = [0]*n + [1]*n

h = .02
x_min, x_max = all_values[:, 0].min() - .5, all_values[:,
0].max() + .5
y_min, y_max = all_values[:, 1].min() - .5, all_values[:,
1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))

# KNN decision
knn = KNeighborsClassifier()
knn.fit(all_values, labels)
knn_prediction = knn.predict(np.c_[xx.ravel(), yy.ravel()])
knn_prediction = knn_prediction.reshape(xx.shape)

pl.figure(1, figsize=(9, 6))
pl.set_cmap(pl.cm.Paired)
pl.pcolormesh(xx, yy, knn_prediction)
pl.scatter(sample1[:, 0], sample1[:, 1], c='r')
pl.scatter(sample2[:, 0], sample2[:, 1], c='g')

# Tree decision
tree = DecisionTreeClassifier()
tree.fit(all_values, labels)
tree_prediction = tree.predict(np.c_[xx.ravel(), yy.ravel()])
tree_prediction = tree_prediction.reshape(xx.shape)

pl.figure(2, figsize=(9, 6))
pl.set_cmap(pl.cm.Paired)
```

```
pl.pcolormesh(xx, yy, tree_prediction)
pl.scatter(sample1[:, 0], sample1[:, 1], c='r')
pl.scatter(sample2[:, 0], sample2[:, 1], c='g')

pl.show()

# Task 2


# Divides class labels and attributes
def class_division(all_adults, index):
    class_labels = []
    attributes = []
    for adult in all_adults:
        class_labels.append(adult[index])
        if index == -1:
            attributes.append(adult[:-1])
        else:
            attributes.append(np.delete(adult, index))
    return class_labels, attributes

# Finds optional params for knn algorithm


def find_knn_params(tr_class, tr_attributes, te_class,
te_attributes):
    neighbours_count = 50
    scores = np.zeros(neighbours_count)
    for k in xrange(1, neighbours_count):
        knn_test = KNeighborsClassifier(n_neighbors=k)
        knn_test.fit(tr_attributes, tr_class)
        scores[k - 1] = knn_test.score(te_attributes, te_class)
    max_score = scores.max()
    max_index = scores.argmax()
    print "KNN:\nMaximum score:", max_score, "( neighbours count
=", max_index + 1, ")"

    # Deviation
    print "Scores:"
    for k in xrange(max_index - 2, max_index + 3):
        print "Score:", scores[k], "( neighbours count =", k +
1, ")"


# Finds optional params for tree algorithm

def find_tree_params(tr_class, tr_attributes, te_class,
te_attributes):
    leaf_count = 10
    depth_count = 10
    scores = np.zeros((depth_count, leaf_count))
    for depth in xrange(1, depth_count):
        for leaf in xrange(1, leaf_count):
```

```
            tree_test = DecisionTreeClassifier(max_depth=depth,
min_samples_leaf=leaf)
            tree_test.fit(tr_attributes, tr_class)
            scores[depth - 1][leaf - 1] =
tree_test.score(te_attributes, te_class)
    max_score = scores.max()
    max_index = np.unravel_index(scores.argmax(), scores.shape)
    print "Tree:\nMaximum score:", max_score, \
        "( maximum depth =", max_index[0] + 1, ", minimum leafs
=", max_index[1] + 1, ")"

    # Deviation
    print "Scores:"
    for depth in xrange(max_index[0] - 1, max_index[0] + 2):
        for leaf in xrange(max_index[1] - 1, max_index[1] + 2):
            if 0 <= depth < depth_count and 0 <= leaf <=
leaf_count:
                print "Score:", scores[depth][leaf], "( maximum
depth =", depth + 1, ", minimum leafs =", leaf + 1, ")"


# Reading file
adults = pd.read_csv('adult.data', names=['age', 'workclass',
'fnlwgt', 'education', 'education-num', 'marital-status',
                                          'occupation',
'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
                                          'hours-per-week',
'native-country', 'income'])

# Deleting nominal attributes
adults_without_nominal = adults.drop(['workclass', 'education',
'marital-status', 'occupation', 'relationship', 'race',
                                      'sex', 'native-country'],
axis=1)

train, test = train_test_split(adults_without_nominal,
test_size=.3)
train_class, train_attributes = class_division(train, -1)
test_class, test_attributes = class_division(test, -1)

find_knn_params(train_class, train_attributes, test_class,
test_attributes)
find_tree_params(train_class, train_attributes, test_class,
test_attributes)
print "\n\n\n"


# Processing with nominal attributes

# One-hot encoding
for column in ['workclass', 'education', 'marital-status',
'occupation', 'relationship', 'race', 'sex', 'native-country']:
    column_dummy = pd.get_dummies(adults[column])
```

```
    if ' ?' in column_dummy:
        column_dummy = column_dummy.drop(' ?', axis=1)
    concatenated = pd.concat([adults, column_dummy], axis=1)
    adults = concatenated.drop(column, axis=1)

income_index = np.nonzero(adults.columns.values ==
'income')[0][0]
train, test = train_test_split(adults, test_size=.3)
train_class, train_attributes = class_division(train,
income_index)
test_class, test_attributes = class_division(test, income_index)

# Testing with KNN
knn = KNeighborsClassifier()
knn.fit(test_attributes, test_class)
print "One-hot encoding score (KNN): ",
knn.score(test_attributes, test_class)

# Testing with tree
tree = DecisionTreeClassifier()
tree.fit(train_attributes, train_class)
print "One-hot encoding score (tree):",
tree.score(test_attributes, test_class)
```