Московский авиационный институт

(национальный исследовательский университет)

«МАИ»

Факультет №3 — «Системы управления, информатика и электроэнергетика»

Кафедра 316 — «Системное моделирование и автоматизированное проектирование»

Курсовая работа

по дисциплине: **«Программирование Python»**

Выполнил:

студент группы М3О-118М-19

Пономарев Роман

Прогнозирование бутстреп методом

Задача — спрогнозировать дальнейшее развитие параметра, описанного в исходной выборке временного ряда. Прогноз необходимо реализовать, используя метод бутстрепа для размножения исходной выборки.

Продажи у нас по месяцам, в году 12 месяцев, поэтому из ряда мы случайным образом возьмем 12 значений и сделаем это 1 000 раз. Далее по каждому из 1 000 рядов со случайными значениями в выборке рассчитаем среднее значение каждого ряда, и по средним значениям каждого ряда рассчитаем еще раз среднее, получим ср. месячные продажи.

Методы размножения выборок (бутстреп-методы)

Бутстрэп (англ. *bootstrap*) в статистике — практический компьютерный метод исследования распределения статистик вероятностных распределений, основанный на многократной генерации выборок методом Монте-Карло на базе имеющейся выборки[2]. Позволяет просто и быстро оценивать самые разные статистики (доверительные интервалы, дисперсию, корреляцию и так далее) для сложных моделей.

Эконометрика и прикладная статистика бурно развиваются последние десятилетия. Серьезным (хотя, разумеется, не единственным и не главным) стимулом является стремительно растущая производительность вычислительных средств. Поэтому понятен острый интерес к статистическим методам, интенсивно использующим компьютеры. Одним из таких методов является так называемый "бутстреп", предложенный в 1977 г. Б.Эфроном из Станфордского университета (США).

Сам термин "бутстреп" - это "bootstrap" русскими буквами и буквально означает что-то вроде: "вытягивание себя (из болота) за шнурки от ботинок". Термин специально придуман и заставляет вспомнить о подвигах барона Мюнхгаузена.

Метод бутстрэпа заключается в следующем. Пусть имеется выборка **X** размера **N**. Равномерно возьмем из выборки **N** объектов с возвращением. Это означает, что мы будем **N** раз выбирать произвольный объект выборки (считаем, что каждый объект «достается» с одинаковой вероятностью **1N**), причем каждый раз мы выбираем из всех исходных **N** объектов. Можно представить себе мешок, из которого достают шарики: выбранный на каком-то шаге шарик возвращается обратно в мешок, и следующий выбор опять делается равновероятно из того же числа шариков. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через **X1**. Повторяя процедуру **M** раз,

сгенерируем **M** подвыборок **X1,…,XM**. Теперь мы имеем достаточно большое число выборок и можем оценивать различные статистики исходного распределения.

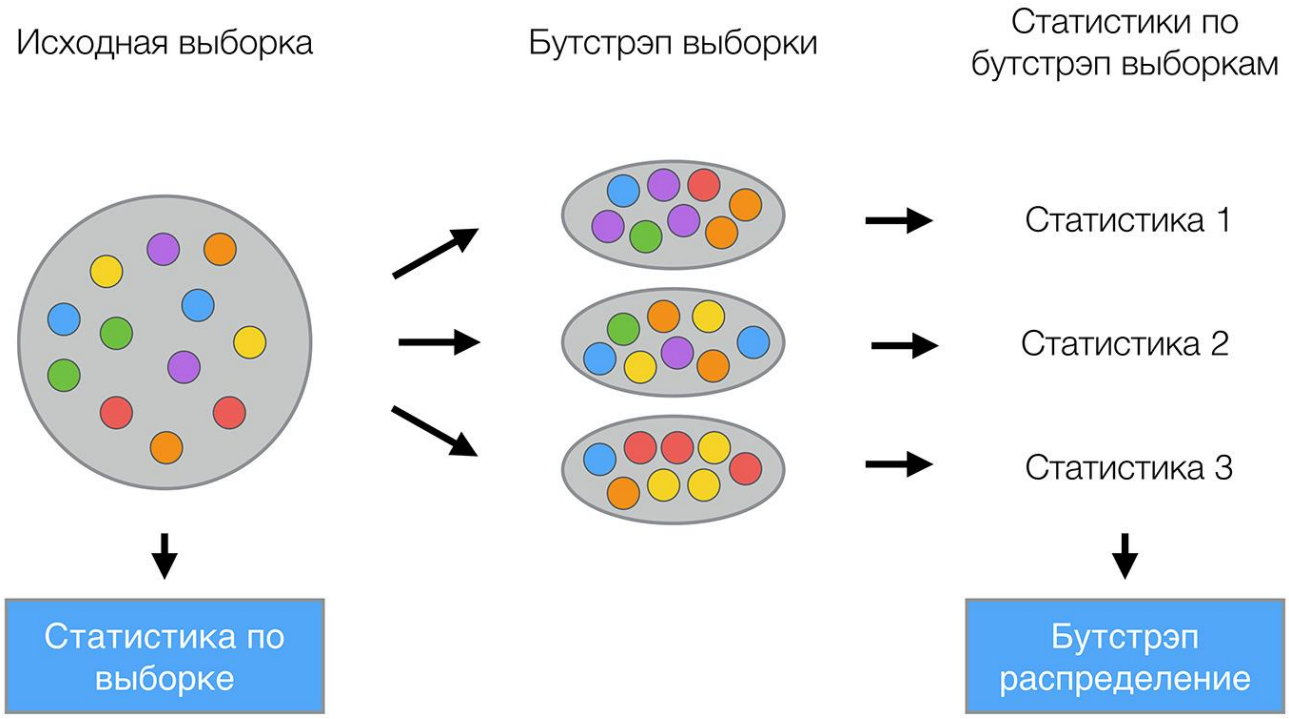

Рисунок 1 - Бутстреп метод

Алгоритм решения задачи:

1. Аппроксимация исходных данных, применяя метод наименьших квадратов, с целью нахождения уравнения прямой и ее построения. Отобразить прямую графически для наглядности.
2. Создать список из значений расстояний **h** между каждой из данных точек и получившейся прямой.
3. Применяя метод бутстрепа увеличить список расстояний до большего значения (например, 1000 значений). Создать, таким образом, несколько списков большего размера (например, 10 списков).
4. Найти среднее значение для каждого из получившихся списков и занести значения в новый список.
5. Найти максимальное и минимальное значения в получившемся списке средних значений. Интервал между максимальным и минимальным значением – доверительный интервал для расстояний от точки до прямой, согласно методу бутстрепа.

6. Теперь, используя полученные значения, можно сделать прогноз на значение Y (цена), подставив в уравнение значение X(месяц). В итоге, получаем прогноз для Y, значение которого лежит в небольшом интервале.

Код Bootstrap forecast:

```
#МНК - Метод наименьших квадратов
#У нас есть много экспериментальных точек. Через них надо провести прямую,
которая как можно ближе проходила к этим точкам.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
#X - МЕСЯЦ
#У - ЦЕНА
#Проведем прямую y = kx + b через экспериментальные точки
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
y = np.array([150, 141, 155.5, 147.3, 161.45, 148.45, 168, 170, 166.78,
170.1, 168, 159.96, 162.49, 178.76])
data = dict(zip(x, y))
print(data)

#Перепишем линейное уравнение y = mx + c как y = Rp, где A = [[ x 1 ]] и p =
[[m], [c]]
#Построим R по x :
R = np.vstack([x, np.ones(len(x))]).T
#�спользуем lstsq для решения его относительно вектора p
k, b = np.linalg.lstsq(R, y)[0]
#print(k, b)
def show_graf(x, y, k, b):
    #Построим график полученной прямой и укажем на нем точки
    plt.plot(x, y, 'o', label='Original data', markersize=10)
    plt.plot(x, k*x + b, 'r', label='Fitted line')
    plt.legend()
    plt.show()
#show_graf(x, y, k, b)
#Рассчитаем кратчайшие расстояния от исходных данных(точек) до полученной
прямой
h = abs(k*x-1*y+b)/((k**2 + (-1)**2)**0.5)
#Рассчитаем кратчайшие расстояния от исходных данных(точек) до полученной
прямой
h = abs(k*x-1*y+b)/((k**2 + (-1)**2)**0.5)
print("Список полученных расстояний:","\n", h)

#Функция для отображения гистограммы для передаваемого списка(h, y(цены))
def show_hist(h):
    x1 = range(len(h))
    ax = plt.gca()
    ax.bar(x1, h, align='edge') # align='edge' - выравнивание по границе, а
не по центру
    ax.set_xticks(x1)
    #ax.set_xticklabels(('first', 'second', 'third', 'fourth'))
    plt.show()


#�дея применения бутсрэпа в том, что у нас есть выборка небольшого размера и
нам надо оценить, например, среднее.
```

```
# ◆'место подсчета среднего самой этой выборки, мы извлекаем n_samples
выборок с возвращением (то есть элементы могут повторяться) из исходной.
# У полученных выборок считаем среднее. Его уже оцениваем, вместо оценки
среднего исходной выборки.
def get_bootstrap_samples(data, n_samples):
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples
#Получим новую выборку из 1000 значений:
n_samples = 1000
number = 10
#Функция расчета доверительного интервала бутстреп методом(прогноз расстояния
h):
def bootstrap_forecast(h, number, n_samples):
    #number = 10
    spisok =[]
    for i in range(0,number):
        sample = get_bootstrap_samples(h, n_samples)
        spisok1 = spisok.append(np.mean(sample))
    #print("spisok =", spisok)
    global h_min
    global h_max
    h_min = min(spisok)
    h_max = max(spisok)
    print("Минимум доверительного интервала для расстояния =", h_min)
    print("Максимум доверительного интервала для расстояния =", h_max)
#bootstrap_forecast(h, number, n_samples)
#Прогноз будущих точек:
# Так, например, двигаясь по оси Ox, значение У будет принадлжать интервалу:
#h = abs(k*x-1*y+b)/((k**2 + (-1)**2)**0.5)
#Функция для расчета цены на определенный месяц:
#◆"ля x=15:
#x=15
def forecast(x, h_min, h_max):
    #◆"ля x=15:
    #x=15
    #◆'Выше прямой МНК:
    y_max1 = -h_min*((k**2 + (-1)**2)**0.5) + k*x + b
    y_min1 = -h_max*((k**2 + (-1)**2)**0.5) + k*x + b
    #Ниже прямой МНК:
    y_min2 = h_min*((k**2 + (-1)**2)**0.5) - k*x - b
    y_max2 = h_max*((k**2 + (-1)**2)**0.5) - k*x - b
    #y = h*((k**2 + (-1)**2)**0.5) - k*x - b
    print(f"Максимальная прогнозируемая цена для {x} месяца = ",y_max1)
    print(f"Минимальная прогнозируемая цена для {x} месяца = ",y_min1)
    #print("y_min2 = ",y_min2)
    #print("y_max2 = ",y_max2)

#Отображение исходных данных и прямой МНК:
show_graf(x, y, k, b)
#Отображение гистограммы известных цен:
show_hist(y)
#Отображение гистограммы известных расстояний точек до прямой:
show_hist(h)
#Прогноз бутстрап методом, где h - исходные расстояния, number - количество
генерируемых выборок
# для нахождения среднего значения выборки, n_samples - размер выборки:
bootstrap_forecast(h, number, n_samples)
#Прогноз будущих цен для определенного месяца(вместо 15 - нужный месяц по
порядку, на выходе интервал возможных значений прогноза
forecast(17, h_min, h_max)
```

https://github.com/romanponomarew/Python1/blob/master/Kursach/BUTSTR.py

Домашние задания

- KNN по станциям Метро
- Центральный по посредничеству социального графа VK
- Построение многоугольника по точкам
- ID3 алгоритм

Код KNN по станциям Метро:

```
import numpy as np
import pandas as pd
metro = pd.read_csv('METRO(3).csv', delimiter=';')
print(metro,"\n")

coffee = metro[metro["coffee"]!=0]
#print(coffee)
print("Станции на которых пьют кофе:")
print(coffee[["name", "coffee", "tea" ]], "\n")

tea = metro[metro["tea"]!=0]
#print(tea)
print("Станции на которых пьют чай:")
print(tea[["name", "tea", "coffee"]])
```

Данные: METRO(3).csv

https://github.com/romanponomarew/Python1/tree/master/METRO

Код Центральный по посредничеству социального графа VK:

```
import networkx as nx

# для визуализации
import matplotlib.pyplot as plt
#%config InlineBackend.figure_format = 'svg'
plt.rcParams['figure.figsize'] = (10, 6)
#Создание пустого графа
G = nx.Graph()
#�"обавление узлов
G.add_nodes_from(["Маша", "Саша", "Сергей", "�"аша", "�'аня","Таня", "Рома",
"Кирилл", "Коля", "�'ова", "Андрей", "Лена", "Света", "Лера"])
```

```
#G.nodes()
G.add_edge("Маша", "Саша")
G.add_edge("Сергей", "Саша")
G.add_edge("�"аша", "Саша")
G.add_edge("�"аша", "Сергей")
G.add_edge("�'аня", "Сергей")
G.add_edge("�'аня", "Маша")
G.add_edge("Коля", "�"аша")
G.add_edge("Коля", "Лера")
G.add_edge("Лена", "Лера")
G.add_edge("Лена", "Рома")
G.add_edge("Маша", "Рома")
G.add_edge("Саша", "Рома")
G.add_edge("Лена", "Света")
G.add_edge("Сергей", "Лера")
G.add_edge("�"аша", "Кирилл")
G.add_edge("Таня", "Маша")
G.add_edge("�'ова", "Андрей")
G.add_edge("Андрей", "Саша")
G.add_edge("Таня", "�'ова")
G.add_edge("Кирилл", "Рома")
G.add_edge("Сергей", "Кирилл")
G.add_edge("Таня", "Сергей")

nx.draw(G, with_labels=True, font_weight='bold')
plt.show();
print("Количество узлов в графе =", G.number_of_nodes())
print("Количество связей в графе =", G.number_of_edges())
bet_centr = nx.betweenness_centrality(G)
print(bet_centr)

for k,v in bet_centr.items():
    if v == max(bet_centr.values()):
        a = k
print(a)
```

https://github.com/romanponomarew/Python1/blob/master/Metro(Betweenness%20centrality%2C%20%D0%B8%D0%BB%D0%B8%20%D0%BF%D0%BE%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE)/GRAF.py

Код построение многоугольника по точкам:

Алгоритм Джарвиса (или алгоритм обхода Джарвиса, или алгоритм заворачивания подарка) определяет последовательность элементов множества, образующих выпуклую оболочку для этого множества. Метод можно представить как обтягивание верёвкой множества вбитых в доску гвоздей. Алгоритм работает за время {\displaystyle O(nh)} O(nh), где {\displaystyle n} n — общее число точек на плоскости, {\displaystyle h} h — число точек в выпуклой оболочке

Псевдокод: Jarvis(P)

1. p[1] = самая левая нижняя точка множества P;
2. p[2] = соседняя точка от p[1] справа (находится через минимальный положительный полярный угол)
3. i = 2;
4. do: (a)for для каждой точки j от 1 до |P|, кроме уже попавших в выпуклую оболочку, но включая p[1] p[i+1] = point_with_min_cos(p[i-1], p[i], P[j]); //точка, образующая минимальный косинус с прямой p[i-1]p[i], (b)i = i + 1; while p[i] != p[1]
5. return p;

```
Алгоритм ?"жарвиса - поиск следующей точки по минимальному углу
# Класс Point для координат x,y точек
class Point:
      def __init__(self, x, y):
            self.x = x
            self.y = y


def Left_index(points):
      # Функция поиска самой левой точки, точки начала построения
      minn = 0
      for i in range(1, len(points)):
            if points[i].x < points[minn].x:
                  minn = i
            elif points[i].x == points[minn].x: #При равных x левой точки
ищем самую левую, верхнюю точку
                  if points[i].y > points[minn].y:
                        minn = i
      return minn


def orientation(p, q, r):
      '''
      Расчет правой тройки векторов, направление построения по алгоритму -
против часовой стрелки (p, q, r).
      Функция возвращает 0, 1 или 2:
      0 --> p, q и r - параллельны
      1 --> левая тройка векторов (по часовой стрелке)
      2 --> правая тройка векторов (против часовой стрелки)
      '''
      val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y)

      if val == 0:
            return 0
      elif val > 0:
            return 1
      else:
            return 2

def convexHull(points, n):
      # ?"олжно быть по крайней мере 3 точки
```

```
	if n < 3:
		return

	# Находим самую левую точку для начала построения
	l = Left_index(points)

	hull = []

	'''
	Начинаем с левой точки, двигаясь против часовой стрелки пока не попадем
в начальную точку
	'''
	p = l
	q = 0
	while (True):

		# ?"обавляем текущую точку к результату
		hull.append(p)
		'''
		?щем точку q, которая правее текущей точки. Если точка i правее,
чем q - изменяем q
		'''
		q = (p + 1) % n

		for i in range(n):
			# If i is more counterclockwise
			# than current q, then update q
			if (orientation(points[p], points[i], points[q]) == 2):
				q = i
		'''
		Now q is the most counterclockwise with respect to p
		Set p as q for next iteration, so that q is added to
		result 'hull'
		'''
		p = q
		# Пока не вернемся в начальную точку
		if (p == l):
			break
	# ?'Вывод результата
	for each in hull:
		print(points[each].x, points[each].y)

	# ?'Исходные данные
points = []
points.append(Point(1, 2))
points.append(Point(2, 2))
points.append(Point(2, 0))
points.append(Point(0, 1))
points.append(Point(3, 4))
points.append(Point(0, 3))
points.append(Point(3, 3))
points.append(Point(3, 5))
convexHull(points, len(points))
# Python1
```

https://github.com/romanponomarew/Python1/blob/master/Mnogougolnik_DZ.py

Код ID3 алгоритм:

```
import math
import csv
students_list = list()
with open('output.csv', encoding="utf-8") as csvfile:
        students_csv = csv.reader(csvfile, delimiter=';')
        students_list = list(students_csv)

header = students_list.pop(0)
print(header)
print(students_list)
#print(students_list[0])   #студент
#print(students_list[0][1]) #редактор студента

redaktor = list()
for stroka in students_list:
    redaktor.append(stroka[1])
#print(redaktor) # Список со значениями признака редактора для всех студентов
redak = set(redaktor) #Множество с неповторяющимися редакторами
#print(redak)
spisok = list()
for i in redak:
    p = redaktor.count(i) / len(redaktor) #Частота значения
    m = p * math.log(p) #Энтропия для одного значения
    spisok.append(m) #Список с энтропиями значений
print(spisok)
#entropia = sum(spisok)
print(sum(spisok)) #Общая энтропия для признака
```

https://github.com/romanponomarew/Python1/tree/master/Entropia