Решения к заданиям MYTONA

[Дмитрий Пасько](https://github.com/PasaOpasen)

07.09.2020

Table of Contents

[Задание 1 1](#_Toc50409894)

[Задание 2 2](#_Toc50409895)

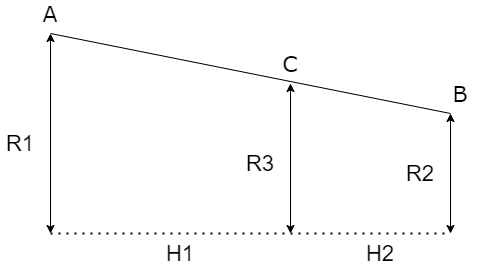
[Задание 3 3](#_Toc50409896)

[Пример 4](#_Toc50409897)

[Решение 5](#_Toc50409898)

# Задание 1

Зафиксируем нашу систему в некоторый момент времени. Схематически она будет выглядеть как-то так:



Здесь пунктирной линией изображен условный горизонт (земля), , - расстояния от земли до левого и правого конца оси. Из условия задачи помним, что эти расстояния взяты из *независимых* нормальных распределений со средним и дисперсиями и соответственно.

Требуется определить положение точки **C**, на которой меньше всего трясёт (дисперсия минимальна).

Сначала выразим через . Поскольку здесь прямоугольная трапеция, это можно сделать через формулу её площади:

Если умножить на 2 и раскрыть скобки, получим:

Известно, что . Поделим последнее полученное уравнение на и обозначим искомое соотношение за :

Найдём дисперсию , используя свойста дисперсии для независимых случайных величин:

Чтобы найти минимум этой функции, требуется найти точки экстремума (нули производной). Я сразу буду избавляться от слагаемых, которые ни на что не влияют:

Нетрудно проверить, что это именно точка минимума.

**Ответ**:

# Задание 2

Чтобы модель хотя бы начала работать для регрессии, нужно

1. Поставить функцию потерь для регрессии (вместо categorical\_crossentropy поставить, например, mean\_squared\_error)
2. Убрать из метрик правильность (acc), так как это метрика для классификаторов.

Сейчас модель выдаёт два числа с суммой 1 (вероятности принадлежности к каждому из двух классов). Для осмысленной регрессии потребуется на последнем слое заменить функцию активации (например, на relu, но зависит от задачи) и установить один нейрон вместо двух (если требуется).

Таким образом, я бы предложил следующий код:

from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers.normalization import BatchNormalization  
  
model = Sequential()  
  
neurons\_count = 32  
model.add(Dense(neurons\_count, input\_dim=X\_train.shape[1], activation='relu'))  
model.add(BatchNormalization())  
model.add(Dense(1, activation='relu'))  
  
model.compile(loss='mean\_squared\_error', optimizer='adam')  
  
model.summary()

# Задание 3

Сразу переведём задачу на язык математики: имеется ансамбль из 5 классификаторов, каждый из которых выдаёт правильный ответ 1 со своей вероятностью. При этом каждый классификатор может иметь целый вес от 0 до ‘5’, причем сумма всех весов равна 5. Требуется подобрать оптимальные веса (чтоб точность ансамбля была максимальной).

Решим эту задачу, проделав большое число симуляций.

Первым делом запишем вероятности:

probs = c(0.8, 0.7, 0.6, 0.5, 0.3)  
  
print(probs)

## [1] 0.8 0.7 0.6 0.5 0.3

Определим всевозможные конфигурации весов, подходящие условию (все комбинации 5 чисел от 0 до 5, дающих в сумме 5):

library(tidyr)  
  
facilities = 0:5  
all\_combs = crossing(coef1 = facilities, coef2 = facilities, coef3 = facilities,   
 coef4 = facilities, coef5 = facilities)  
  
is5 = apply(all\_combs, 1, function(row) sum(row) == 5)  
  
all\_combs = all\_combs[is5, ]  
  
all\_combs

## # A tibble: 126 x 5  
## coef1 coef2 coef3 coef4 coef5  
## <int> <int> <int> <int> <int>  
## 1 0 0 0 0 5  
## 2 0 0 0 1 4  
## 3 0 0 0 2 3  
## 4 0 0 0 3 2  
## 5 0 0 0 4 1  
## 6 0 0 0 5 0  
## 7 0 0 1 0 4  
## 8 0 0 1 1 3  
## 9 0 0 1 2 2  
## 10 0 0 1 3 1  
## # ... with 116 more rows

**Алгоритм симуляции** заключается в следующем:

1. для каждого классификатора проводим симуляций, то есть получаем случайных чисел из биномиального распределения с соответствующей вероятностью позитивного исхода
2. для каждой конфигурации весов и всех симуляций проводим суммирование голосов и находим среднее число позитивных исходов у каждой конфигурации (позитивный исход - это когда сумма ответов по ансамблю больше чем 2)
3. выбираем конфигурацию с наибольшим средним числом позитивных исходов

## Пример

Сделаем 10 симуляций с нашими классификаторами:

sim\_results = sapply(probs, function(p) rbinom(10, 1, p))  
  
sim\_results

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 0 1 1 1 0  
## [2,] 1 1 0 0 0  
## [3,] 0 1 1 1 0  
## [4,] 1 1 1 0 0  
## [5,] 1 0 0 1 0  
## [6,] 1 0 0 0 1  
## [7,] 1 1 1 0 1  
## [8,] 1 1 0 1 1  
## [9,] 1 1 0 1 0  
## [10,] 1 0 1 1 0

Возьмем конфигурацию номер 67:

comb = all\_combs[67, ]  
  
comb

## # A tibble: 1 x 5  
## coef1 coef2 coef3 coef4 coef5  
## <int> <int> <int> <int> <int>  
## 1 1 0 2 1 1

Перемножим полученные матрицу и вектор:

ans = sim\_results %\*% t(comb)  
  
ans

## [,1]  
## [1,] 3  
## [2,] 1  
## [3,] 3  
## [4,] 3  
## [5,] 2  
## [6,] 2  
## [7,] 4  
## [8,] 3  
## [9,] 2  
## [10,] 4

Преобразуем ответ в категории правильно-неправильно:

ans = ans > 2  
  
ans

## [,1]  
## [1,] TRUE  
## [2,] FALSE  
## [3,] TRUE  
## [4,] TRUE  
## [5,] FALSE  
## [6,] FALSE  
## [7,] TRUE  
## [8,] TRUE  
## [9,] FALSE  
## [10,] TRUE

Найдём среднее количество верных ответов, это и будет оценка нашей конфигурации:

mean(ans)

## [1] 0.6

## Решение

Симуляцию выполняет следующая функция:

get\_best\_combination = function(count) {  
   
 sim\_results = sapply(probs, function(p) rbinom(count, 1, p))  
   
   
 answers = numeric(nrow(all\_combs))  
   
 for (i in 1:nrow(all\_combs)) {  
   
 comb = all\_combs[i, ]  
   
 nums = (sim\_results %\*% t(comb)) > 2  
   
 answers[i] = mean(nums)  
   
 }  
   
 result = as.numeric(all\_combs[which.max(answers), ])  
   
 cat("best accuracy:", max(answers), "on", result, "combination\n")  
   
 return(result)  
}

Запустим 100 млн симуляций и получим ответ:

get\_best\_combination(10^8)

## best accuracy: 0.800048 on 3 0 0 0 2 combination

## [1] 3 0 0 0 2

Ответ неожиданный, но я получал его много раз. Его можно объяснить тем, что раз пятый классификатор так часто ошибается, его мнение очень необычное и при некотором раскладе весов может быть полезно.

**Ответ**: 3 из первого, 2 из последнего