Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:

«Машинное обучение»

По лабораторной работе №8

«Выявление аномалий»

Выполнил: Сенькович Дмитрий Сергеевич

магистрант кафедры информатики

группы №858642

Проверил: Стержанов Максим Валерьевич

доцент, кандидат технических наук

Минск 2019

**Оглавление**

[1 Введение 2](#_Toc25849337)

[2 Выявление аномалий в работе сервера 5](#_Toc25849338)

[2.1 Постановка задачи 5](#_Toc25849339)

[2.2 Чтение данных 5](#_Toc25849340)

[2.3 График исходных данных 5](#_Toc25849341)

[2.4 Представление данных в виде независимых нормально распределенных случайных величин 6](#_Toc25849342)

[2.5 Оценка параметров случайных величин 7](#_Toc25849343)

[2.6 График плотности распределения 8](#_Toc25849344)

[2.7 Реализация алгоритма выявления аномалий. Поиск оптимального ε 9](#_Toc25849345)

[2.8 Визуализация аномальных значений выборки 11](#_Toc25849346)

[3 Алгоритм выявления аномалий для n-мерной случайной величины 13](#_Toc25849347)

[3.1 Постановка задачи 13](#_Toc25849348)

[3.2 Чтение данных 13](#_Toc25849349)

[3.3 Представление данных в виде 11-мерной нормально распределенной величины 13](#_Toc25849350)

[3.4 Поиск лучшего ε 14](#_Toc25849351)

# 1 Введение

Еще одна задача машинного обучения - выявление аномалий. Алгоритм чем-то схож с логистической регрессией: нужно отнести экземпляр выборки к 0 или 1, где - аномалия. Но важным отличием является то, что в выборке для задачи выявления аномалий, как правило, очень мало положительных меток - аномалий. Такая задачи - несбалансированная, требует особой оценки качества.

Базовый алгоритм состоит в следующим: каждая из фич моделируется как случайная независимая от других фич величина с нормальным распределением, а наши данные оцениваются как случайная величина со следующей функцией плотности распределения:



Рисунок 1.1. Функция плотности распределения полученной случайной величины

Параметры функции плотности распределения для каждой случайной величины вычисляются по выборке. Во время обучения выбирается некоторый threshold ε. Этот параметр подбирается таким образом, чтобы получить наилучший результат на валидационной выборке - F-score. F-score используется ввиду несбалансированности задачи.

Далее, для предсказания аномалии, для произвольного вектора размерности n подсчитывается значение функции плотности распределения выше. Если получившееся значение ниже ε - найдена аномалия.

Такой алгоритм достаточно быстро работает и отлично распараллеливается на нескольких параллельных вычислительных компонентах. Однако в алгоритме есть следующая проблема: он не учитывает возможные связи между фичами, моделируя их как независимые случайные величины.

Есть другой подход: моделирование данных в виде n-мерной нормально распределенной случайной величины. В это подходе используются не разбросы каждой фичи как случайной величины, а ковариационная матрица, способная отражать связи между фичами, что позволяет моделировать более сложные зависимости. Например, если обычные, не аномальные данные, расположены в эллипсе, вытянутом по диагонали, то первый подход не сможет выявить аномалии вне эллипса но внутри круга, для которого сравнение с ε актуально:

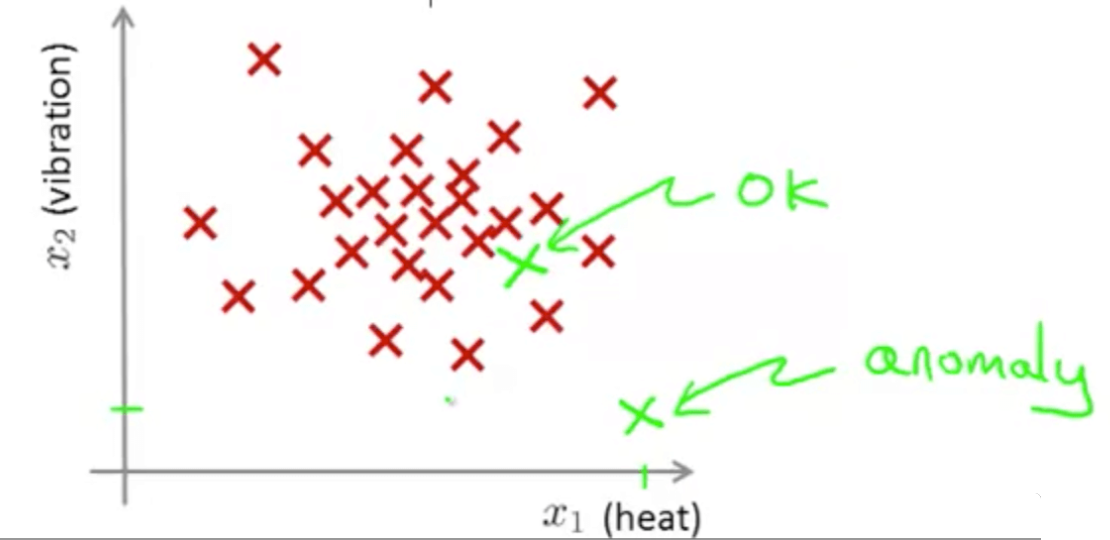


Рисунок 1.2. Пример данных, для которых первых подход даст хорошие результаты

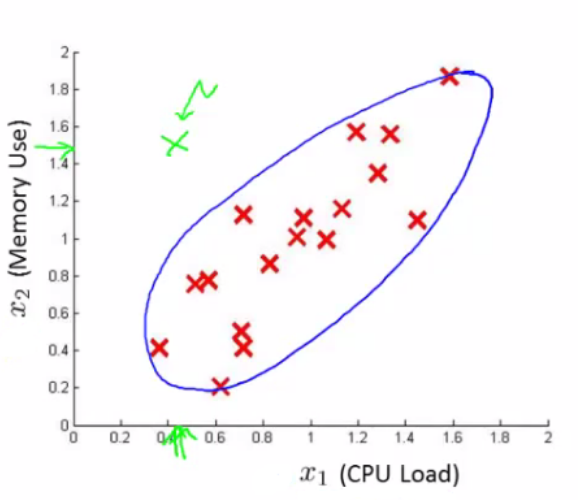


Рисунок 1.3. Пример данных, для которых первых подход может ошибаться

Более наглядное влияние матрицы ковариации (то есть, наличие ненулевых элементов вне диагонали в первой случае) на функцию распределения величины: чем холоднее цвет, тем больше шанс, что алгоритм посчитает экземпляр аномалией:

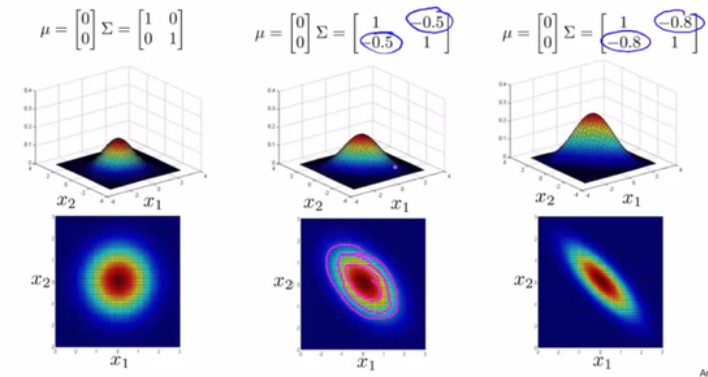


Рисунок 1.4. Влияние учитывания связей между фичами

Во втором случае функция плотности распределения имеет следующий вид:

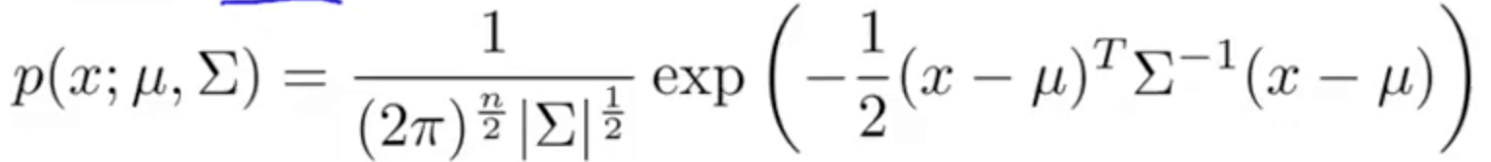


Рисунок 1.5. Формула функции плотности распределения n-мерной нормальной распределенной случайной величины

где параметры - мат ожидание и ковариационная матрицы - выглядят следующим образом:

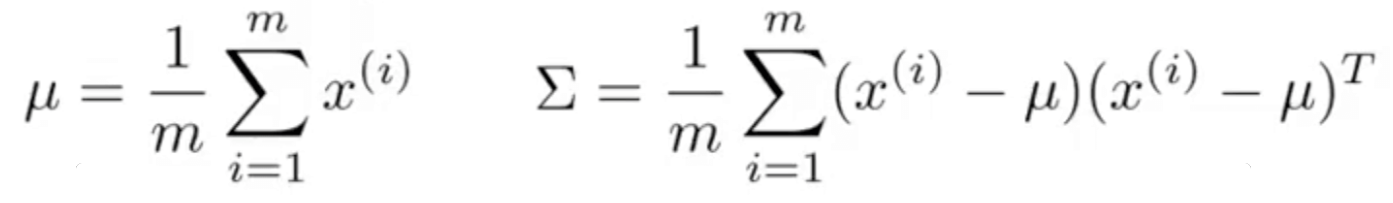


Рисунок 1.6. Формулы для вычислений параметров функции плотности распределения n-мерной нормальной распределенной случайной величины

Выявление аномалий имеет широкое практическое значение: например, выявление неисправностей в оборудованиях, датацентрах, выявление хакерских взломов, подозрительного поведения банковских пользователей и т.д..

# 2 Выявление аномалий в работе сервера

### 2.1 Постановка задачи

Дан набор двух параметров работы сервера, требуется реализовать алгоритм выявления аномалий в работе сервера.

### 2.2 Чтение данных

Считаем данные:

data = loadmat('ex8data1.mat')

x = data['X']

x\_val = data['Xval']

y\_val = data['yval']

### 2.3 График исходных данных

Построим график исходных данных в виде диаграммы рассеяния:

def plot\_data(x):

plt.scatter(x[:, 0], x[:, 1])

plot\_data(x)

plt.show()

Полученный график:

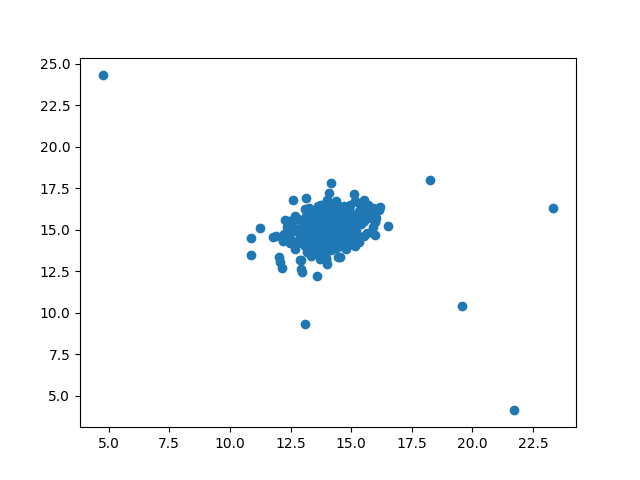


Рисунок 2.3.1. График рассеяния для исходных данных

### 2.4 Представление данных в виде независимых нормально распределенных случайных величин

Построим гистограммы распределения наших параметров:

def plot\_histogram(x):

plt.hist(x, bins=70)

plot\_histogram(x[:, 0])

plt.show()

plot\_histogram(x[:, 1])

plt.show()

Полученные гистограммы:

|  |  |
| --- | --- |
|  |  |

Рисунок 2.4.1. Гистограммы распределения параметров

Видим, что оценивание распределения фич как нормального имеет смысл для обоих параметров.

### 2.5 Оценка параметров случайных величин

Оценим параметры распределения фич как независимых нормально распределенных случайных величин:

def estimate(x):

means = [np.mean(x[:, i]) for i in range(x.shape[1])]

stds = [np.std(x[:, i]) for i in range(x.shape[1])]

return means, stds

means, stds = estimate(x)

Здесь подсчитываются мат ожидания и cреднеквадратические отклонения случайных величин, так как формула нормального распределения случайной величины имеет следующую формулу:

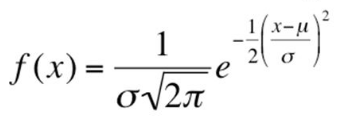


Рисунок 2.5.1. Формула функции нормального распределения

### 2.6 График плотности распределения

Для большей наглядности создадим график плотности распределения получившейся случайной величины вместе с исходными данными:

def p(x, means, stds):

probability = 1.0

for i in range(x.shape[0]):

mean = means[i]

std = stds[i]

xi = x[i]

probability \*= (1.0/(math.sqrt(2\*math.pi)\*std))\*np.exp(-((xi - mean)\*\*2)/(2\*(std\*\*2)))

return probability

def plot\_distribution(means, stds):

x1, x2 = np.meshgrid(np.arange(0, 30, 0.1), np.arange(0, 30, 0.1))

points\_count = len(x1)

z = np.zeros(shape=(points\_count, points\_count))

for i in range(points\_count):

for j in range(points\_count):

z[i, j] = p(np.array([x1[i, i], x2[j, j]]).reshape(-1, 1), means, stds)

levels = [10 \*\* exp for exp in range(-60, 0, 3)]

plt.contour(x1, x2, z, levels=levels)

plot\_data(x)

plot\_distribution(means, stds)

plt.show()

Мы добавили функцию плотности распределения и функция построения графика плотности распределения. Получившийся график представлен ниже:

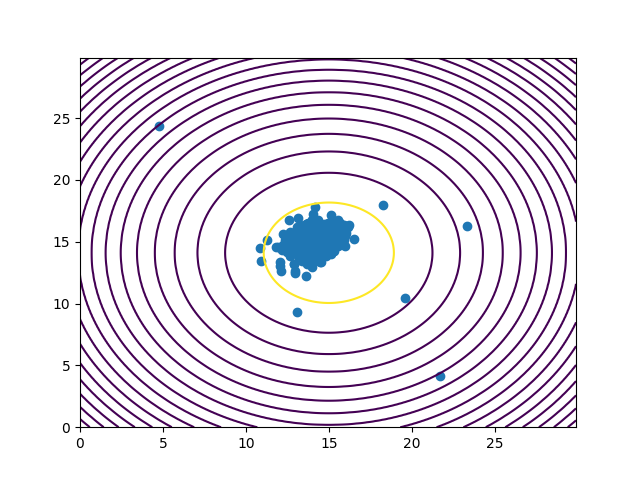


Рисунок 2.6.1. График плотности распределения получившийся случайно величины

Желтым цветом показаны наиболее высокие участки функции плотности распределения на 3d графике. Эти участки соответствуют наиболее вероятному месту расположения экземпляров выборки, не являющимися аномалиями.

### 2.7 Реализация алгоритма выявления аномалий. Поиск оптимального ε

Алгоритм выявления аномалий, по сути, будет просто проверять значение функции плотности распределения для очередного вектора выборки, сравнивать его с некоторым ε и возвращать 1, если получившееся значение меньше этого ε:

def is\_outlier(x, means, stds, epsilon):

return p(x, means, stds) < epsilon

Далее, реализуем функцию оценки работы нашего алгоритма - F-score. Данная оценка качества алгоритма позволяет более точно оценивать accuracy классификации алгоритма, не позволяя алгоритм “обмануть” исследователя: F-score метрика является метрикой, составленной из precision и recall таким образом, чтобы оба параметра были высоки. Максимальное значения F-score единица достигается лишь в случае, когда обе метрики 1. Первая метрика является часть правильно классифицированных положительных результатов среди всех предсказанных положительных меток. Вторая - часть правильно классифицированных положительных результатов среди всех положительных меток в выборке:

def f\_score(res):

predicted\_positive\_total = res[0][0] + res[0][1]

precision = res[0][0] / (predicted\_positive\_total if predicted\_positive\_total > 0 else 1)

actual\_positive\_total = res[0][0] + res[1][0]

recall = res[0][0] / (actual\_positive\_total if actual\_positive\_total > 0 else 1)

return 2\*precision\*recall/((precision + recall) if (precision + recall) > 0 else 1)

Искать лучшую ε будем путем перебора 100 различных значений в интервале между минимальных и максимальным значением функции плотности распределения величины на обучающей выборке. Для каждого такого значения ε будем вычислять f-score на валидационной выборке. Выберем ε, максимизирующий f-score:

def find\_best\_epsilon(x\_val, y\_val, means, stds):

best\_f\_score = 0

best\_epsilon = -1

p\_values = [p(x\_val[i], means, stds) for i in range(len(x\_val))]

step = (np.max(p\_values) - np.min(p\_values)) / 1000

epsilons = np.arange(np.min(p\_values), np.max(p\_values), step)

for epsilon in epsilons:

true\_positives = 0.0

false\_positives = 0.0

false\_negatives = 0.0

true\_negatives = 0.0

is\_outliers = p\_values < epsilon

for i in range(len(x\_val)):

yi = y\_val[i, 0]

is\_positive = 1 if is\_outliers[i] else 0

if is\_positive == 1 and yi == 1:

true\_positives += 1

elif is\_positive == 1 and yi == 0:

false\_positives += 1

elif is\_positive == 0 and yi == 1:

false\_negatives += 1

elif is\_positive == 0 and yi == 0:

true\_negatives += 1

score = f\_score(np.array([

[true\_positives, false\_positives],

[false\_negatives, true\_negatives]

]))

if score > best\_f\_score:

best\_f\_score = score

best\_epsilon = epsilon

print(best\_f\_score)

print(np.array([

[true\_positives, false\_positives],

[false\_negatives, true\_negatives]

]))

return best\_epsilon, best\_f\_score

best\_epsilon, best\_f\_score = find\_best\_epsilon(x\_val, y\_val, means, stds)

print('best f score: %s' % best\_f\_score)

print('best epsilon: %s' % best\_epsilon)

В нашем случае лучший ε = 8.990852779269492e-05, при котором лучший f-score: 0.875.

### 2.8 Визуализация аномальных значений выборки

Выделить аномальные значения на обучающей выборке с учетом лучшего значения ε:

def is\_outlier(x, means, stds, epsilon):

return p(x, means, stds) < epsilon

def plot\_outliers(x, means, stds, epsilon):

for xi in x:

if is\_outlier(xi, means, stds, epsilon):

plt.plot(xi[0], xi[1], 'ro', c='r')

plot\_data(x)

plot\_distribution(means, stds)

plot\_outliers(x, means, stds, best\_epsilon)

plt.show()

Получим следующий результат:

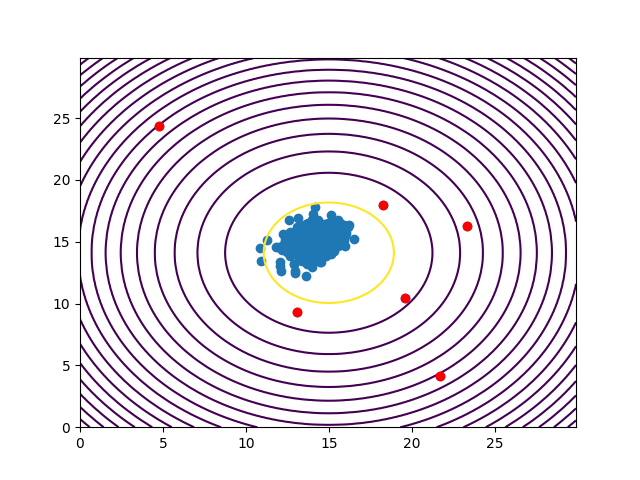


Рисунок 2.8.1. График данных с аномальными значениями

# 3 Алгоритм выявления аномалий для n-мерной случайной величины

### 3.1 Постановка задачи

Смоделировать данные в виде 11-мерной нормально распределенной случайной величины и реализовать алгоритм выявления аномалий с лучшим ε.

### 3.2 Чтение данных

Считаем данные:

data = loadmat('ex8data2.mat')

x = data['X']

x\_val = data['Xval']

y\_val = data['yval']

### 3.3 Представление данных в виде 11-мерной нормально распределенной величины

Реализуем функции оценки величины:

def estimate(x):

mu = np.array([np.mean(x[:, i]) for i in range(x.shape[1])]).reshape(1, -1)

sigma = compute\_sigma(x, mu)

return mu, sigma

def compute\_sigma(x, mu):

m = x.shape[0]

features\_count = x.shape[1]

sum = np.zeros(shape=(1, features\_count))

for xi in x:

diff = xi - mu

sum += diff \*\* 2

return sum / m

Реализуем функцию плотности распределения получившейся величины:

def p(x, mu, sigma):

sigma = np.diag(sigma.ravel())

sigma\_det = np.linalg.det(sigma)

n = mu.shape[1]

first\_part = (((2\*math.pi)\*\*(n/2))\*(sigma\_det\*\*0.5))\*\*-1

sigma\_inv = np.linalg.pinv(sigma)

diff = x - mu

second\_part = np.exp(-0.5 \* np.sum(diff.dot(sigma\_inv) \* diff, 1)).reshape(-1, 1)

return first\_part \* second\_part

### 3.4 Поиск лучшего ε

Поиск будет организован аналогично предыдущему пункту, единственная разница - по-другому смоделированная функция плотности распределения случайной величины:

best\_epsilon, best\_f\_score, best\_confusion\_matrix = find\_best\_epsilon(x\_val, y\_val, mu, sigma)

print('best f score: %s' % best\_f\_score)

print('best epsilon: %s' % best\_epsilon)

p\_vals = p(x, mu, sigma)

anomalies\_count = sum(p\_vals < best\_epsilon)

print('anomalies\_count: %s' % anomalies\_count)

В результате поиска лучшего порогового значения и подсчета аномальных значений получим 117 аномалий при лучшем значении ε 1.3772288907613627e-18, дающем максимальный f-score 0.6153846153846154.