МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра теории функций и стохастического анализа

ИСПОЛЬЗОВАНИЕ РЕГУЛЯРИЗАЦИИ ДЛЯ РЕШЕНИЯ ПРОБЛЕМЫ ПЕРЕОБУЧЕНИЯ

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 412 группы
направления 01.03.02 — Прикладная математика и информатика

механико-математического факультета
Георгиева Ивана Владимировича

Научный руководитель
д. ф.-м. н., доцент _____ С. П. Сидоров

Заведующий кафедрой
д. ф.-м. н., доцент _____ С. П. Сидоров

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире машинное обучение приобретает популярность при решении обширного класса задач. Практически в каждой из таких задач возникает проблема переобучения. При переобучении построенная для решения поставленной задачи модель может давать ответы близкие к реальным, на данных, используемых для обучения. Но при этом на остальных данных результаты могут значительно отличаться от ожидаемых.

Для решения этой проблемы используется много различных методов. Одним из наиболее актуальных является регуляризация. Так, в популярных библиотеках, предназначенных для машинного обучения, некоторые модели используют регуляризацию автоматически, при условии, что значение соответствующего параметра не предполагает обратного. Переобучение часто наступает при слишком больших значениях некоторых коэффициентов модели. Регуляризация не позволяет модели чересчур увеличивать коэффициенты, то есть снижает уровень переобучения.

Целью бакалаврской работы является разработка программного кода для решения задачи построения регрессии и методов машинного обучения с регуляризацией.

Для достижения поставленных целей в работе необходимо решить следующие задачи:

— изучить основные понятия машинного обучения;
— показать случаи переобучения на примерах;
— изучить основные виды регуляризации;
— изучить язык программирования Python;
— собрать данные для обучения модели из открытых источников;
— разработать программу с использованием библиотек sklearn, csv, numpy;
— обучить модель с использованием и без использования регуляризации;
— провести тестирование программного кода;
— провести сравнение результатов обучения.

Структура и содержание бакалаврской работы. Работа состоит из введения, пяти разделов, заключения, списка использованных источников, содержащего 20 наименований и двух приложений. Общий объём работы со-

ставляет 44 страницы.

В первом разделе рассматриваются основные понятия машинного обучения:

— общая постановка задачи машинного обучения,
— классы задач машинного обучения,
— классические задачи машинного обучения,
— переобучение,
— основные методы решения проблемы переобучения.

Второй раздел посвящён гребневой регрессии.

— Даётся определение гребневой регрессии.
— Приведён алгоритм градиентного спуска для гребневой регрессии.
— Объясняется реализация алгоритма градиентного спуска для построения гребневой регрессии.

В третьем разделе изучаются виды и источники ошибок, возникающих при применении алгоритмов регрессии

В четвёртом разделе описывается регрессия по методу «лассо».

— Приведены отличия от гребневой регрессии.
— Приведён алгоритм координатного спуска для регрессии методом наименьших квадратов.
— Рассмотрен модифицированный алгоритм для регрессии лассо.
— Разобрана реализация алгоритма координатного спуска для построения регрессии по методу «лассо».

В пятом разделе демонстрируются результаты программной реализации построения регрессий на языке Python.

1 Основные понятия машинного обучения

Приведём общую постановку задачи машинного обучения. Имеется множество объектов и множество ответов. Между ними существует некоторая неизвестная зависимость. Известно лишь конечное число пар "объект-ответ" составляющее обучающую выборку. По имеющимся данным следует построить алгоритм, который достаточно точно отображает неизвестную зависимость. То есть, способный для любых данных выдать ответ близкий к реальному.

Классы задач машинного обучения [1]:

— Обучение с учителем – восстановление зависимости по известным примерам и ответам.

— Обучение без учителя – известно лишь множество объектов. Множество ответов отсутствует. Требуется, например, найти закономерности.

Классические задачи машинного обучения:

— Задача классификации. Множество объектов разделено некоторым образом на классы. Имеется обучающая выборка, содержащая пары "объект, класс". Требуется для произвольного объекта определить, к какому классу он мог бы принадлежать [2].

— Задачи кластеризации. Предназначены как для разработки типологии, так и для проверки гипотез на основе исследования данных.

— Задача регрессии. И множество объектов, и множество ответов являются численными данными. Требуется по конечному числу имеющихся точек восстановить исходную зависимость.

Функция прогноза для регрессии:

$$y_{pred} = w[0]x[0] + w[1]x[1] + \ldots + w[p]x[p] + b.$$

где x – вектор признаков длины p одной точки, w и b – параметры модели, которые находятся её обучением и y_{pred} – предсказание. Для одного признака y_{pred} – линия, для двух – плоскость, для большего числа – гиперплоскость.

Существует множество различных линейных моделей регрессии. Разница между ними в том, как параметры модели узнаются из обучающих данных, и том, как можно контролировать сложность модели.

Линейная регрессия (метод наименьших квадратов) – простейший линейный метод [3]. Модель находит параметры, которые минимизируют среднеквадратичную ошибку между прогнозом и истинным значением. Среднеквадратичная ошибка – это сумма квадратов разностей между прогнозом и истинным значением. Если результаты на тренировочном и тестовом наборе очень близки, это означает, что происходит недообучение.

Обычно при глубоком обучении требуется минимизировать по w следующую функцию потерь:

$$Q(w, X) = \frac{1}{l}||Xw - y||^2,$$

для этой задачи известно точное решение:

$$w^* = (X^T X)^{-1} X^T y.$$

Однако, при его использовании на практике из-за нескольких умножений и взятия обратной матрицы могут быть значительные потери точности. Кроме того, требуется обращать матрицу $X^T X$ размера $n \times n$, что требует n^3 действий, где n – число параметров, может быть достаточно большим [4].

В задаче линейной классификации в качестве функции потерь зачастую используют долю неправильных ответов:

$$Q(a, X) = \frac{1}{l} \sum_{i=1}^{l} [a(x_i) \neq y_i].$$

У данной функции потерь есть несколько важных свойств:

— Она разрывная.

— Она не гладкая.

Отсюда следует, что при её исследовании нельзя использовать методы гладкой оптимизации. Чтобы решить эту проблему, часто вместо $Q(a, X)$ рассматривают некую её гладкую верхнюю оценку, при минимизации которой будет минимизироваться и сама функция потерь [5].

$$Q(a, X) = \frac{1}{l} \sum_{i=1}^{l} [a(x_i) \neq y_i].$$

Одна из основных проблем машинного обучения – переобучение [6]. Это явление, при котором для элементов обучающей выборки модель показывает результат, близкий к корректному, а для любого другого работает гораздо хуже. Это связано с тем, что при обучении модель обнаруживает в выборке случайные закономерности и в итоге "запоминает" все ответы.

Так как обучающая выборка конечна и неполна, а так же достоверно отличить случайные флуктуации от закономерностей невозможно, переобучение будет присутствовать практически всегда.

Один из факторов, способствующих переобучению, – чрезмерная сложность модели. Зачастую большое количество параметров поощряет подгонку под обучающее множество.

Методы борьбы с переобучением [7]:

— Перекрестная проверка – данные разбиваются на k частей. Обучение модели проходит на $k-1$, тестирование на одной.
— Ранняя остановка – как только значение ошибки на тестовых данных начало превышать значение на обучающей выборке, прекращаем обучение.
— Увеличение количества обучающих данных – либо искусственно, специальным образом обрабатывая данные, либо путём сбора дополнительной информации.
— Ансамбли моделей – использование нескольких однотипных моделей параллельным или последовательным образом.
— Регуляризация.

Регуляризация – добавление дополнительных условий к задаче с целью предотвратить переобучение [8].

В данной работе рассматривается применение l_1, l_2 - регуляризаций.

В этих методах используется линейная регрессия, но на этот раз модель добавит дополнительное ограничение на коэффициент w. Было установлено, что при достаточно больших значениях $||w||$ переобучение наступает чаще. Желательно, чтобы величина коэффициентов была как можно меньше, все значения w должны быть близки к нулю.

В случае l_1 регуляризации требуется минимизировать

$$Q(w, X) + \lambda_1 ||w||.$$

В случае l_2 регуляризации:

$$Q(w, X) + \lambda_2 ||w||^2.$$

Иногда l_1 и l_2 регуляризации комбинируют. И минимизируется следующая функция потерь:

$$Q(w, X) + \lambda_1 ||w|| + \lambda_2 ||w||^2.$$

Такая модель называется $ELASTICNET$.

Кроме того существует техника регуляризации исключением.

В отличии от вышесказанных техник, регуляризация исключением не может применяться к линейным моделям и вместо изменения функции ошибки, будет меняться сама сеть.

Во время обучения нейросети часть её нейронов, за исключением входных и выходных, случайным образом временно удаляются. В результате чего получается изменённая сеть с меньшим числом нейронов. Далее берётся небольшое число примеров, на них происходит обучение сети. Обновляются соответствующие веса и смещения [9]. Затем восстанавливаются удалённые нейроны и случайным образом выбирается новая группа для удаления. Одна из возможных реализаций этого алгоритма – вместо удаления фиксированного числа нейронов каждый раз, для каждого нейрона задать вероятность с которым он будет удаляться на новом шаге.

Получаем, когда удаляются разные нейроны, процесс обучения становится похож на обучение сразу нескольких различных нейросетей. И процедура исключения имеет схожий эффект с усреднением по большому числу нейросетей. Разные сети будут переобучаться по-разному и их общий результат может иметь более низкий уровень переобучения.

2 Гребневая регрессия

Известно, что использование многочленов более высокой степени в уравнении регрессии приводит к переобучению. Переобучение происходит, когда модель слишком хорошо "подходит" для обучающих данных и не обобщается на тестовые данные.

Обратимся к рисункам 1 и 2: переобучение также может произойти, если в уравнении регрессии слишком много независимых переменных или, если наблюдений слишком мало [10]. Переобучение также связано с очень большими оценочными параметрами (весами) $\hat{w}$.

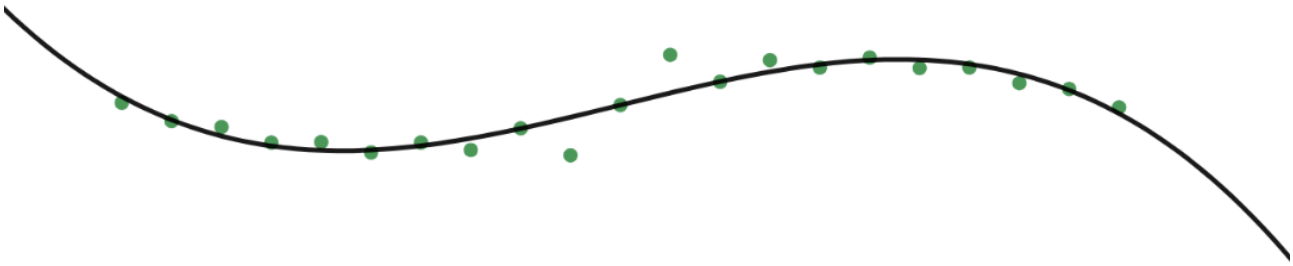

Рисунок 1 – Большое число наблюдений

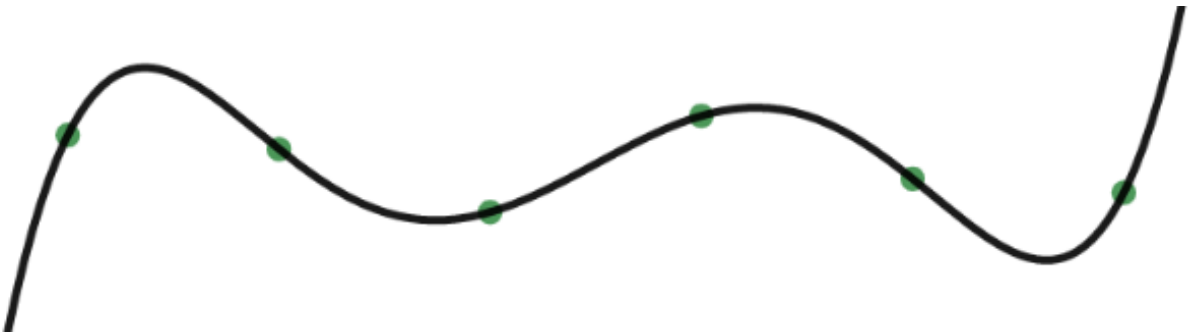

Рисунок 2 – Малое число наблюдений

Поэтому необходимо найти баланс между

— мерой соответствия (насколько хорошо наша модель соответствует данным);
— величиной коэффициентов.

Таким образом, общая оценка модели представляет собой комбинацию меры соответствия и величиной коэффициентов. Мера соответствия представлена суммой квадратов разностей (RSS): небольшая указывает на хорошее соответствие. Мера величины коэффициентов – это сумма абсолютных значений коэффициентов (l_1-норма) или сумма квадратов значений коэффициентов (l_2-норма). Они представлены следующим образом:

$$\|w_0\| + \|w_1\| + \cdots + \|w_n\| = \sum_{j=0}^{n} \|w_j\| = \|w\|_1 \ (l_1 \text{ норма}),$$

$$w_0^2 + w_1^2 + \cdots + w_n^2 = \sum_{j=0}^{n} w_j^2 = \|w\|_2^2 \ (l_2 \text{ норма}).$$

В гребневой регрессии считаем меру величины коэффициентов как l_2-норму. Таким образом, общая оценка

$$Total\ Cost = RSS(w) + \|w\|_2^2 . \tag{1}$$

Цель при построении гребневой регрессии состоит в том, чтобы найти $\hat{w}$, чтобы минимизировать общие затраты в уравнении (1). Баланс между мерой соответствия и величиной коэффициентов достигается путем введения параметра настройки λ следующим образом:

$$Total\ Cost = RSS(w) + \lambda \|w\|_2^2 . \tag{2}$$

Если $\lambda = 0$, то задача суммы квадратов отклонений сводится к минимизации $RSS(w)$, то есть $\hat{w}^{LS}$.

Если $\lambda = \infty$, то общая стоимость равна ∞, когда $(\hat{w} \neq 0)$ и общая стоимость равна 0, когда $(\hat{w} = 0)$.

Если λ между, то $\|\hat{w}\|_2^2 \leq \left\|\hat{w}^{(LS)}\right\|_2^2$.

2.1 Вычисление градиента гребневой регрессии

Закрытая форма решения

Используя метод наименьших квадратов, получаем:

$$RSS(w) = (y - \mathbf{H}w)^T (y - \mathbf{H}w).$$

Общая оценка в случае гребневой регрессии равна

$$\begin{aligned} Total\ Cost &= (y - \mathbf{H}w)^T(y - \mathbf{H}w) + \lambda \|w\|_2^2 = \\ &= (y - \mathbf{H}w)^T(y - \mathbf{H}w) + \lambda w^T w. \end{aligned} \tag{3}$$

Градиент уравнения (3)

$$\begin{gathered} \Delta \left[RSS(w) + \lambda \|w\|_2^2 \right] = \Delta (y - \mathbf{H}w)^T(y - \mathbf{H}w) + \lambda \|w\|_2^2, \\ \Delta Cost(w) = -2\mathbf{H}^t(y - \mathbf{H}w) + \lambda(2w) = \\ = -2\mathbf{H}^t(y - \mathbf{H}w) + 2\lambda I w. \end{gathered} \tag{4}$$

Приравнивая уравнение (4) к 0, получаем

$$\begin{gathered} \Delta Cost(w) = 0, \\ -2\mathbf{H}^T(y - \mathbf{H}w) + 2\lambda I w = 0, \\ -\mathbf{H}^T y + \mathbf{H}^T \mathbf{H}\hat{w} + \lambda I \hat{w} = 0, \\ (\mathbf{H}^T \mathbf{H} + \lambda I)\hat{w} = \mathbf{H}^T y, \\ \hat{w}_{ridge} = (\mathbf{H}^T \mathbf{H} + \lambda I)^{-1} \mathbf{H}^T y. \end{gathered} \tag{5}$$

Если $\lambda = 0$, то $(\hat{w}_{ridge} = \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T y = \hat{w}^{LS}$.

Если $\lambda = 0$, то $(\hat{w}_{ridge} = 0)$.

Тот факт, что в уравнении (3) добавлено слагаемое, оценивающее сложность модели, называется регуляризацией [11]. Сложность модели настраивается параметром λ.

Градиентный спуск Поэлементный алгоритм градиентного спуска гребневой регрессии можно записать следующим образом:

$$\begin{gathered} w_j^{(t+1)} \leftarrow w_j^{(t)} + \Delta Cost(w), \\ w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta(2\mathbf{H}^T(y - \mathbf{H}w) + 2\lambda w), \\ w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \left[-2 \sum_{i=1}^{n} h_j(x_i)(y_i - \hat{y}_i(w^{(t)})) + 2\lambda w_j^{(t)} \right], \\ w_j^{(t+1)} \leftarrow w_j^{(t)}(1 - 2\eta\lambda) + 2\eta \sum_{i=1}^{n} h_j(x_i)(y_i - \hat{y}_i(w^{(t)})). \end{gathered} \tag{6}$$

В уравнении (6) η – размер шага и выражение $(1 - 2\eta\lambda)$ всегда меньше или равно 1 пока $\eta > 0$, $\lambda > 0$. Выражение $2\eta \sum_{i=1}^{n} h_j(x_i)(y_i - \hat{y}_i(w^{(t)}))$ – коэффициент обновления из RSS.

Таким образом, $w_j^{(t)}$ уменьшается в $(1 - 2\eta\lambda)$ на промежуточном этапе, а затем $w_j^{(t+1)}$ увеличивается в коэффициент обновления $2\eta \sum_{i=1}^{n} h_j(x_i)(y_i - \hat{y}_i(w^{(t)}))$.

Итак,

— Для регрессии, которая строится методом наименьших квадратов, будет $w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta^*$ (коэффициент обновления);

— Для гребневой регрессии будет $w_j^{(t+1)} \leftarrow (1-2\eta\lambda)w_j^{(t)} - \eta^*$ (коэффициент обновления).

Важным вопросом является выбор параметра настройки λ. Чтобы найти λ, можно использовать k-кратную перекрестную проверку.

Процесс включает подгонку $\hat{w}_\lambda$ к обучающему набору данных, тестирование производительности модели с $\hat{w}_\lambda$ на тестовом наборе для выбора λ^* и, наконец, оценку ошибки обобщения модели с $\hat{w}_{\lambda^*}$. Средняя ошибка вычисляется следующим образом:

$$CV(\lambda) = \frac{1}{k} \sum_{k=1}^{k} error_k(\lambda). \tag{7}$$

2.2 Реализация алгоритма градиентного спуска для построения гребневой регрессии

В данном подразделе приводится описание приложения, основанного на книге [12], для построения гребневой регрессии, в котором используется набор данных *kc_house_data*. Набор данных содержит сведения о домах, проданных в период с мая 2014 года по май 2015 года в округе Кинг, штат Вашингтон, США. Эти данные охватывают также Сиэтл. Состоит из 21 переменной и 21613 наблюдений Воспользуемся функцией "lm" в R и найдём коэффициенты регрессии. Зависимые переменные – $sqft\_living$ и *bedrooms*, независимая – *price*.

```
lm(price ~ sqft_living + bedrooms, data=house_train_data)$coef
```

(Intercept)	sqft_living	bedrooms
97050.0942	305.2103	-57429.9302

Первая функция в приложении градиентного спуска гребневой регрессии – это создание матрицы данных.

```
data_matrix <- function(data, features, output){
  scaled_feature_matrix <-
    data.frame(scale(data[features]),
      row.names=NULL)
  length <- nrow(data)
  scaled_features <- as.matrix(cbind('Intercept' =
      rep(1, length),
    scaled_feature_matrix[, features]))
  output <- as.matrix(scale(data[output]))
  return(list(scaled_features, output))
}
```

Функция predict_output предсказывает целевые значения.

```
predict_output<-function(feature_matrix, weights){
  predictions=(feature_matrix) %*% (weights)
  return(predictions)
}
```

Согласно источнику [13] штраф l_2 получил свое название, потому что он заставляет веса иметь меньшую l_2 норму, чем в противном случае. Следующая функция принимает начальные веса и матрицу признаков и предсказывает целевые значения с помощью функции predict_output. errors находятся по разнице в предсказанных и фактических значениях. Значение gradient вычисляется с использованием матрицы признаков и errors ($\Delta cost(w) = 2\mathbf{H}^T(y - \mathbf{H}w) + 2\lambda w$). Веса обновляются путем вычитания произведения gradient и step_size (см. уравнение (6)). Норма градиентов рассчитывается, чтобы проверить, меньше ли она tolerance в цикле «while».

```
ridge_regression_gradient_descent =
      function(feature_matrix, output,
    initial_weights, step_size, l2_penalty, tolerance) {
    converged = FALSE
    weights = matrix(initial_weights)
    j = 0
    while (!converged) {
        predictions = predict_output(feature_matrix, weights)
        errors = predictions - output
        # Loop over each feature weight
```

```
        for (i in 1:length(weights)) {
            if (i == 1) {
                gradient =
      2 * (feature_matrix[, i] %*% errors)
                gradient_norm = sqrt(sum(gradient^2))
                weights[1, 1] =
      weights[1, 1] - (step_size * gradient)
            } else {
                gradient =
      2 * (feature_matrix[, i] %*% errors) +
                  2 * (l2_penalty * weights[i, 1])
                gradient_norm = sqrt(sum(gradient^2))
                if (gradient_norm < tolerance) {
                    converged = TRUE
                }
                weights[i, 1] =
      weights[i, 1] - (step_size * gradient)
            }
        }
        j = j + 1
    }
    print(paste0("Gradient descent converged at iteration:",
  j - 1))
    return(weights)
}
```

Присвоим два разных значения параметру настройки $\lambda = 0$ и $\lambda = \infty$ и исследуем, как параметры (веса) корректируются. Рассмотрим модель с двумя функциями, то есть sqft_living и bedrooms.

```
features <- c("sqft_living", "bedrooms")
target <- "price"
train_data <- house_train_data[, c(features, target)]
feature_matrix = data_matrix(train_data,
    features,
    target)[[1]]
output_matrix = data_matrix(train_data,
    features,
    target)[[2]]
initial_weights = c(0, 0, 0)
step_size = 1e-6
tolerance = 1e-8
weights_with_0_penalty =
    ridge_regression_gradient_descent(feature_matrix,
        output_matrix,
        initial_weights,
        step_size,
```

```
        l2_penalty = 0,
        tolerance)
```

[1] "Gradient descent converged at iteration: 3002"

```
print(weights_with_0_penalty)
```

$$
\begin{gathered}
[, 1] \\
[1,] - 5.560047e - 17 \\
[2,] 7.885384e - 01 \\
[3,] - 1.489809e - 01
\end{gathered}
$$

Коэффициенты с $\lambda = 0$ такие же, как для OLS регрессии. Сравним полученные результаты с выводом программы при использовании функции «lm» в R:

```
train_data_features <-
  data.frame(scale(house_train_data[, features]))
train_data_output <-
  data.frame(scale(house_train_data[, target]))
train_data <-
  cbind(train_data_features, train_data_output)
colnames(train_data) <-
  c("sqft_living", "bedrooms", "price")
(OLS_coef <-
  lm(price ~ sqft_living + bedrooms, data =
    train_data)$coef)
```

(Intercept)	sqft_living	bedrooms
2.600634e-17	7.885384e-01	-1.489809e-01

Результаты совпадают. R имеет функцию «glmnet» в пакете glmnet. Характеристики и результат принадлежат классу matrix. Аргумент «alpha» равен 0 для гребневой регрессии гребня и 1 для регрессии лассо. Допуск, определенный в алгоритме, – это «порог», который по умолчанию равен 1e-07. Есть много других параметров, которые можно изучить, набрав help(glmnet). Проверим результаты с помощью функции «glmnet» в R (таблица 1).

Таблица 1 – Сравнение коэффициентов гребневой регрессии с $\lambda = 0$ для приложения, приведённого в данной работе, модели регрессии OLS и glmnet

	Application	OLS	glmnet
(Intercept)	0.00000	0.00000	0.00000
sqft_living	0.78854	0.78854	0.78853
bedrooms	-0.14898	-0.14898	-0.14898

Таблица 2 – Сравнение коэффициентов гребневой регрессии с $\lambda = 100000$ для приложения, приведённого в данной работе, и glmnet

	Application	glmnet
(Intercept)	0.00000	0e+00
sqft_living	0.06679	1e-05
bedrooms	0.02696	0e+00

```
library(glmnet)
ridge_model_lambda_0 <- glmnet(x = feature_matrix,
  y = output_matrix,
  alpha = 0,
  lambda = 0,
```

```
  thresh=1e-8)
glm_ridge_0_coef <- coef(ridge_model_lambda_0)[, 1]
```

Реализованное приложение показывает достаточно высокий уровень точности. Далее используем $\lambda = 100000$

```
initial_weights = c(0, 0, 0)
step_size = 1e-6
tolerance = 1e-8
weights_with_high_penalty =
    ridge_regression_gradient_descent(feature_matrix,
        output_matrix,
        initial_weights,
        step_size,
        l2_penalty = 1e+05,
        tolerance)
```

Эта модель имеет низкую предвзятость [14]. Выполним проверку результатов с помощью функции glmnet в R (таблица 2).

```
ridge_model_lambda_1e5 <- glmnet(x = feature_matrix,
  y = output_matrix,
  alpha = 0,
  lambda = 1e+5,
  thresh=1e-8)
glm_ridge_1e5_coef <- coef(ridge_model_lambda_1e5)[, 1]
```

Действительно, получаем одинаковые результаты, т.е. все коэффициенты очень близки к 0. Покажем графики уравнений гребневой регрессии для двух значений λ на рисунке 3. На оси абсцисс - нормированные значения sqft_living. На оси ординат - нормированные значения price. Модель гребневой регрессии с градиентным спуском переходит в нулевую модель с отсечением 0 (высокое смещение), когда $\lambda = \infty$, и модель регрессии LS, когда $\lambda = 0$. Найдём оптимальное значение λс помощью перекрестной проверки.

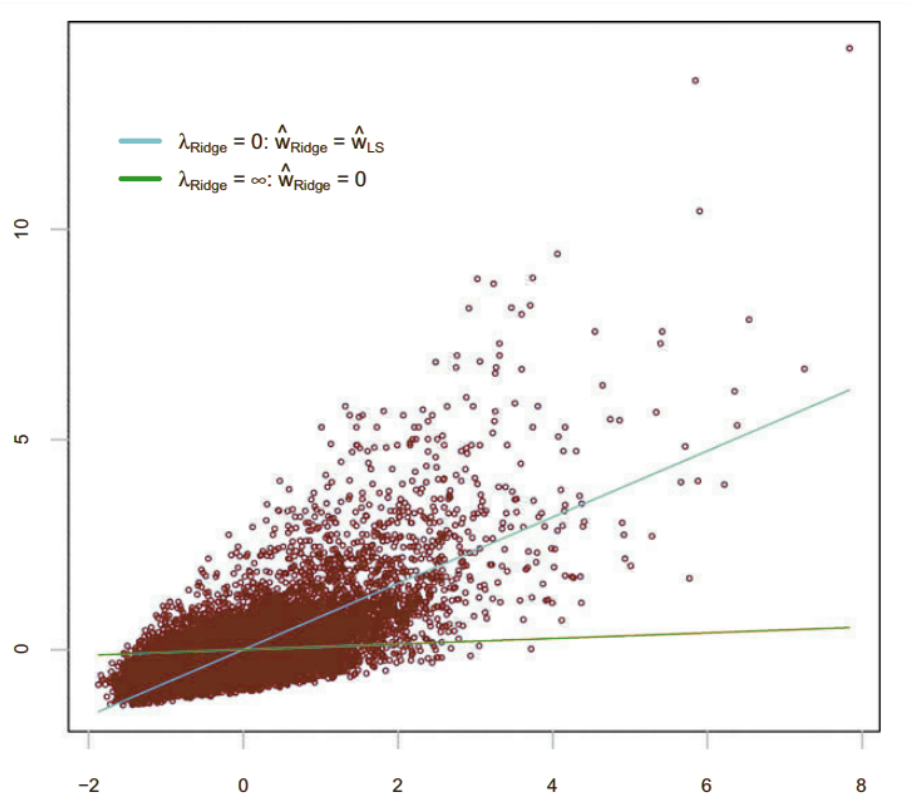

Рисунок 3 – Гребневая регрессия с $\lambda = 0$ и $\lambda = 100000$

3 Оценка производительности

Ранее было упомянуто об оценке модели. Оценка модели – это мера ошибок модели, то есть того, насколько точно модель может предсказать фактическое значение. То есть, если модель не допускает ошибок прогнозирования, стоимость, приписываемая модели, равна нулю, и, следовательно, ее мера точности составляет 100%. Согласно [15] стоимость регрессионной модели измеряется функцией потерь, определяемой как

$$\begin{gathered} Loss(y, f_{\hat{w}}(x)) = (y - f_{\hat{w}}(x))^2 \text{ – ошибка в квадрате,} \\ Loss(y, f_{\hat{w}}(x)) = |y - f_{\hat{w}}(x)| \text{ – модуль ошибки.} \end{gathered} \tag{8}$$

О мере точности модели можно судить по оценке модели на обучающих и тестовых данных. При обучении модели модель пытается соответствовать обучающим данным. Следовательно, с увеличением сложности модели ошибка обучения уменьшается. Очень низкая ошибка обучения не является хорошим показателем точности модели, если обучающие данные не включают «вселенную», то есть все возможные данные, соответствующие данной задаче. Следовательно, эффективность модели можно оценить только по степени ее точности с тестовыми данными. **Ошибка обобщения** – это ошибка модели на данных, не принимавших участие в процессе обучения. Обсуждаемые здесь ошибки модели зависят от сложности и других алгоритмических характеристик модели. Но в самих данных могут быть ошибки. В следующем подразделе приведены возможные способы решения данной проблемы.

3.1 Источники ошибок

Ошибки могут возникать из данных обучения модели и из тестовых данных. Однако существуют также три других источника ошибок в модели – шум, смещение и дисперсия. **Шум** всегда проникает в данные, потому что практически невозможно записать без потери информации, например, «человеческие эмоции», «проблемы во взаимоотношениях», «глобальные проблемы» и так далее. На основании имеющихся данных в модели также проникают **смещения** и **отклонения**. Если обучить модель на основе обучающих данных, в которых нет шума (noise), и оценить ту же модель на тестовых данных, которые содержат шум, получим модель с очень низкой точностью.

Следовательно ошибка прогнозирования в модели состоит из трех компонентов: шума, смещения и дисперсии, которые можно записать как

$$\begin{aligned} Prediction\ Error &= \varsigma^2 + MSE\left[f_{\hat{w}}(x)\right] \\ &= \varsigma^2 + \left[bias(f_{\hat{w}}(x))\right]^2 + var(f_{\hat{w}}(x)). \end{aligned} \tag{9}$$

Подходящая модель выбирается путем выбора параметра настройки λ, который управляет сложностью модели. Это делается с помощью перекрестной проверки. Выбрав подходящую модель, оцениваем ошибку обобщения. Рассмотрим house_train_data и house_test_data, которые были разделены из набора данных kc_house. Наборы данных для обучения и тестирования имеют 10 807 и 10 806 наблюдения соответственно.

```
train_features =
  as.matrix(data.frame(scale(house_train_data[,
features])))
train_output =
  as.matrix(data.frame(scale(house_train_data[,
target])))
test_features =
  as.matrix(data.frame(scale(house_test_data[,
features])))
test_output =
  as.matrix(data.frame(scale(house_test_data[, target])))
```

Обучим модель гребневой регрессии с $\lambda = 0$, используя обучающие данные, то есть реплицируем OLS регрессию и используем модель для прогнозирования тестовых данных с $\lambda = 10$.

```
ridge_model_train <- glmnet(x = train_features,
  y = train_output,
  alpha=0,
  lambda = 0)
ridge_predict = predict(ridge_model_train,
  newx = test_features,
  s = 10,
  exact = TRUE)
MSE_lambda_10 = mean((ridge_predict - test_output)^2)
```

[1] "MSE_lambda_10 = 0.903155"

Среднеквадратичная ошибка для этой модели составляет 0.903155. R имеет встроенную функцию перекрестной проверки cv.glmnet(). По умолча-

нию функция выполняет 10-кратную перекрестную проверку, но это можно изменить с помощью аргумента «nfolds». Выполняем 10-кратную перекрестную проверку данных обучения и находим значение λ, для которого ошибка перекрестной проверки наименьшая.

```
cv_ridge = cv.glmnet(x = train_features,
  y = train_output,
  alpha=0)
ideal_lambda=min(cv_ridge$lambda)
```

[1] "ideal_lambda = 0.076901"

```
ridge_predict = predict(ridge_model_train,
  s=ideal_lambda,
  newx = test_features,
  exact = TRUE)
MSE_ideal_lambda =
  mean((ridge_predict - test_output)^2)
```

[1] "MSE with ideal_lambda = 0.495869"

Среднеквадратичная ошибка при $\lambda = 0.076901$ составляет 0.495869, то есть MSE упала на 45%. Выясним коэффициенты этой модели

```
ridge_train = glmnet(x =
  train_features, y = train_output, alpha = 0)
ridge_ideal_lambda_coef <-
    predict(ridge_train, type = "coefficients",
  s = ideal_lambda)
ridge_ideal_lambda_coef[, 1]
```

(Intercept)	sqft_living	bedrooms
-1.397232e-16	6.998124e-01	-8.978171e-02

Визуализируем рассмотренное выше уравнение гребневой регрессии. Ось абсцисс – sqft_living. Ось ординат – price. На рисунке 4 представлен график уравнения регуляризованной гребневой регрессии.

Покажем, как меняются нормированные коэффициенты при разных значениях λ. Построим график: на оси абсцисс – λ, на оси ординат – значения соответствующих коэффициентов. На рисунке 5 видно, что коэффициенты меняются постепенно от $\lambda = 0$ до $\lambda \to \infty$. Резкого уменьшения до нуля не происходит.

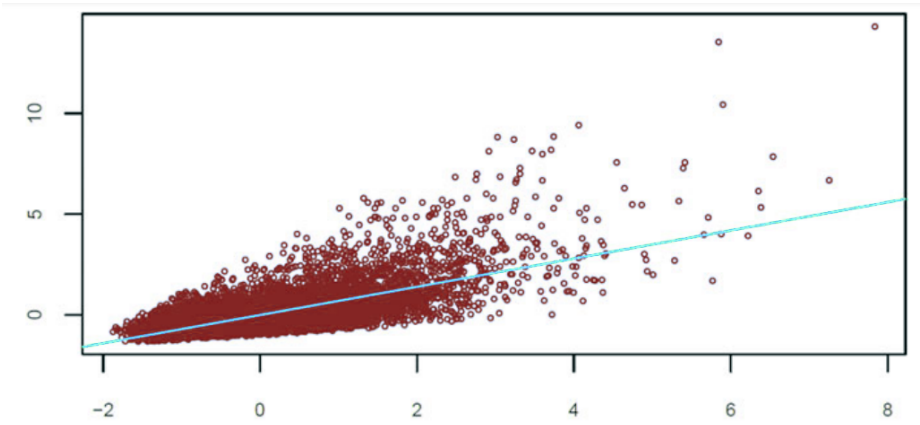

Рисунок 4 – Гребневая регрессия с $\lambda = 0.076901$

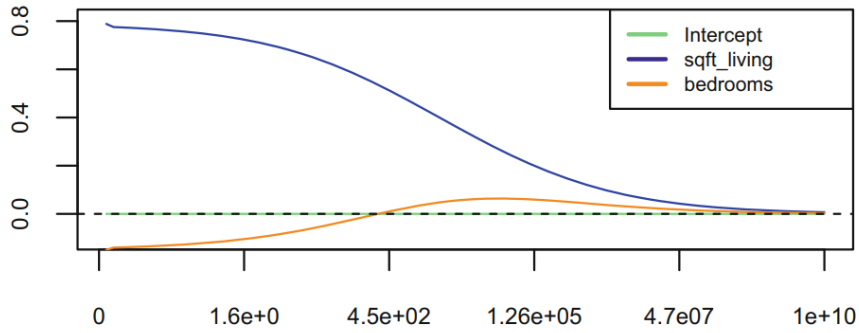

Рисунок 5 – Нормированные коэффициенты гребневой регрессии при изменении λ. Коэффициенты не падают резко до 0 при $\lambda \to \infty$

3.2 Дилемма смещения – дисперсии в гребневой регрессии

Большое значение λ приведет к высокому смещению (простая модель) и, следовательно, к низкой дисперсии ($\hat{w} = 0$, для $\lambda = \infty$). Небольшое значение λ приведет к низкому смещению и, как следствие, большой дисперсии [16].

4 Регрессия по методу «лассо»

Если имеется «широкий» набор функций (скажем, $1e + 10$), гребневая регуляризация может создать вычислительные проблемы, поскольку выбраны все функции. Алгоритм лассо отбрасывает менее важные или избыточные характеристики, переводя их коэффициенты в ноль. Это позволяет более качественно интерпретировать функции, а также сокращает время вычислений. Регрессия по методу «лассо» (регуляризованная регрессия l_1) использует норму l_1 в качестве штрафа, вместо нормы l_2 в гребневой регрессии. Перед тем, как перейти к целевой функции регрессии лассо, вернёмся к алгоритму гребневой регрессии . Гребневая регрессия выбирает параметры β с минимальной RSS при условии, что норма l_2 параметров $\beta_1^2 + \beta_2^2 + \cdots + \beta_n^2 \leq t$. На рисунке 6 показаны изолинии RSS и ограничения параметров для гребневой регрессии и регрессии лассо. Контуры RSS представляют собой эллипсы, центрированные по методу наименьших квадратов (точка красного цвета). Ограничение построено для различных значений параметров β. В случае гребневой регрессии $\beta_1^2 + \beta_2^2 \leq t$, ограничение гребня. Ограничение принимает форму круга для двух параметров и становится сферой с большим количеством параметров. Первая точка соприкосновения контура RSS с окружностью – это точка, описывающая параметры гребня β_1 и β_2. Значения β в большинстве случаев оказываются ненулевыми. Если t мало, параметры будут маленькими, а если велико, будет стремиться к решению по методу наименьших квадратов.

В случае регрессии лассо параметры β выбираются такими, что $|\beta_1| + |\beta_2| \leq t$, ограничение лассо, для минимального RSS. Как показано на правом графике рисунка 6 для регрессии лассо с двумя параметрами, ограничение имеет форму ромба с четырьмя углами; для более чем двух элементов контур становится ромбовидным с множеством углов. Если контур RSS касается угла, он заставляет одну из β стать нулевой. Можно переписать общую оценку в формуле (3) для регрессии лассо как

$$Total\ Cost = RSS(w) + \lambda \|w\|_1 . \tag{10}$$

Если $\lambda = 0, \hat{w}^{lasso} = \hat{w}^{LS}$. Если $\lambda = \infty, \hat{w}^{lasso} = 0$. Если λ между, то $0 \leq \hat{w}^{lasso} \leq \hat{w}^{LS}$.

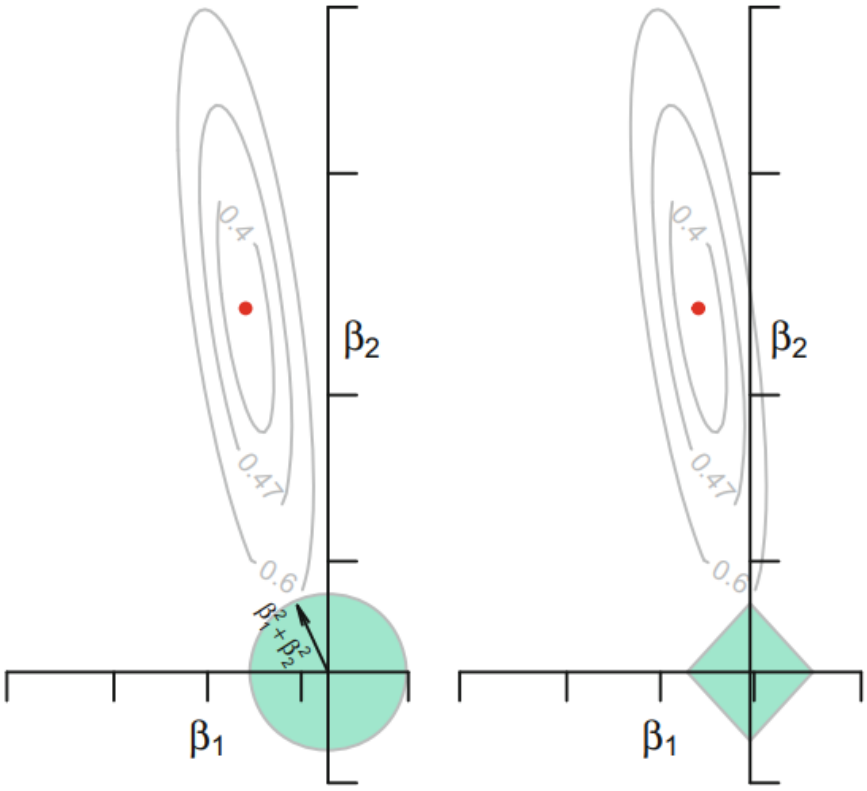

Рисунок 6 – Левый график показывает контуры RSS гребневой регрессии и ограничение $\beta_1^2 + \beta_2^2 \leq t$ (ограничение гребня). Правый график показывает RSS-контуры регрессии лассо и ограничение на коэффициенты лассо $|\beta_1| + |\beta_2| \leq t$ (ограничение лассо)

Лассо выбирает параметры β с минимальным RSS, при условии, что l_1 норма параметров $(|\beta_1| + |\beta_2| + \cdots + |\beta_n|) \leq tolerance$. Ранее было показано, что оптимальное решение задачи минимизации лассо находится в начале координат, и поэтому градиент вычислить нельзя. Поэтому решением является выпуклый алгоритм оптимизации, называемый координатным спуском [17]. Этот алгоритм пытается минимизировать

$$f(w) = f(w_0, w_1, \cdots, w_n). \tag{11}$$

Алгоритм координатного спуска можно описать следующим образом:

$$\begin{gathered} \text{Initialize } \hat{w}, \\ \text{while not converged, pick a coordinate j,} \\ \hat{w}_j \leftarrow \min_{f} (\hat{w}_0.\hat{w}_1, \cdots, w, \hat{w}_{j+1}, \cdots, \hat{w}_n). \end{gathered} \tag{12}$$

Если выбирать следующую координату случайным образом, этот алго-

ритм станет алгоритмом стохастического координатного спуска. При координатном спуске не нужно выбирать размер шага.

4.1 Координатный спуск для регрессии наименьших квадратов

Ниже приведен алгоритм координатного спуска для регрессии наименьших квадратов, по одной координате за раз.

$$RSS(w) = \sum_{i=1}^{n}(y_i - \sum_{j=0}^{n} w_j h_j(x_i))^2.$$

Зафиксируем все координаты w_{-j} и возьмем частную производную по w_j:

$$\frac{\partial}{\partial w_j} = -2\sum_{i=1}^{n} h_j(x_i)(y_i - \sum_{j=0}^{n} w_j h_j(x_i))^2,$$

$$= -2\sum_{i=1}^{n} h_j(x_i)(y_i - \sum_{k\neq j} w_k h_k(x_i) - w_j h_j(x_i)),$$

$$= -2\sum_{i=1}^{m} h_j(x_i)(y_i - \sum_{k\neq j} w_k h_k(x_i)) + 2w_j \sum_{i=1}^{n} h_j(x_i)^2,$$

Пояснения:

по определению нормированных функций $\sum_{i=1}^{n} h_j(x_i)^2 = 1,$

$$\sum_{i=1}^{n} h_j(x_i)(y_i - \sum_{k\neq j} w_k h_k(x_i) = \rho_j = -2\rho + 2w_j.$$

Приравняв частичные производные к 0, получим

$$w_j = \rho_j. \tag{13}$$

4.2 Координатный спуск для регрессии лассо

Ниже приведен псевдокод спуска координат для регрессии лассо, по одной координате за раз.

$$\begin{gathered}
\text{Initialize } \hat{w}, \\
\text{while not converged,} \\
\text{for j in } 0, 1, 2, \cdots, n \\
\text{compute: } \rho_j = \sum_{i=1}^{n} h_j(x_i)(y_i - \hat{y}_i(\hat{w}_j)), \\
\text{set: } w_j = \begin{cases} \rho_j + \frac{\lambda}{2} \text{ if } \rho_j < -\frac{\lambda}{2}, \\ 0 \text{ if } \rho_j \text{ in } \left[-\frac{\lambda}{2}, \frac{\lambda}{2}\right], \\ \rho_j - \frac{\lambda}{2} \text{ if } \rho_j > \frac{\lambda}{2}. \end{cases}
\end{gathered} \tag{14}$$

4.3 Написание приложения для координатного спуска регрессии лассо

На основе книги [12] приведём алгоритм регрессии лассо.

Функция get_features подбирает данные на основе выбранных функций.

```
get_features <- function(data, features){
  selected_features <- matrix(nrow = nrow(data), ncol = length(features))
  for (i in 1: length(features)){
    selected_features[,i] <- as.numeric(data[[features[[i]]]])
  }
  selected_features
}
```

Функция add_constant_term добавляет столбец единиц к матрице признаков.

```
add_constant_term <- function(data_features){
  length <- nrow(data_features)
  combined <- cbind(rep(1, length), data_features)
  combined
}
```

Функция construct_matrix принимает имена функций и выходные данные из data.frame. Затем он выбирает значения функций и выходные значения, вызывая функцию get_features, и возвращает их в виде списка.

```
construct_features_matrix <- function(features, outputs, data) {
  features <- as.list(features)
  data_features <- get_features(data, features)
  output_data <- get_features(data, outputs)
  features_data <- add_constant_term(data_features)
```

```
  list(features_data, output_data)
}
```

Функция normalize_features вычисляет норму для каждой функции и нормализует feature_matrix.

```
normalize_features <- function(feature_Matrix) {
  norms <- as.numeric(vector(length = ncol(feature_Matrix)))
  normalized_features <- matrix(nrow = nrow(feature_Matrix),
    ncol = ncol(feature_Matrix))
  for (i in 1:ncol(feature_Matrix)) {
    v <- feature_Matrix[, i]
    norms[i] <- sqrt(sum(v^2))
    normalized_features[, i] <- feature_Matrix[, i]/norms[i]
  }
  list(normalized_features, norms)
}
```

Функция predict_output предсказывает выходные значения.

```
predict_output <- function(feature_matrix, weights){
  predictions<-feature_matrix[[1]]%*%weights
  predictions
}
```

В уравнении (13) было показано, что $\rho_j = \sum_{i=1}^{n} h_j(x_i)(y_i - \sum_{k \neq j} w_k h_k(x_i))$. Для понимания последующего кода, в котором вычисляется ρ, рассмотрим набор данных с тремя функциями, в котором требуется найти ρ_2. Это можно записать как

$$\begin{gathered} \rho_2 = h_2(x_2)\left[(y_1 + y_2 + y_3) - (\hat{w}_1 h_1(x_1) + \hat{w}_3 h_3(x_3))\right] = \\ = h_2(x_2)\left[(y_1 + y_2 + y_3) - (\hat{y}_1 + \hat{y}_3)\right] = \\ = h_2(x_2)\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2) + (y_3 - \hat{y}_3) + \hat{y}_2\right] = \\ = h_2(x_2)\left[(target - predicted) + \hat{w}_2 h_2(x_2)\right]. \end{gathered} \tag{15}$$

```
get_ro <- function(feature_matrix, output, weights, i) {
  prediction = predict_output(feature_matrix, weights)
  feature_i = feature_matrix[[1]][, i]
  ro_i = sum(feature_i * (output - prediction + weights[i] * feature_i))
  return(ro_i)
}
```

Далее реализуем функцию координатного спуска, которая минимизирует функцию стоимости для одного признака i. Функция возвращает новый

вес для признака i.

```
lasso_coordinate_descent_step <- function(i, data_matrix, weights,
  l1_penalty) {
  normalized_features <- normalize_features(feature_Matrix = data_matrix
    [[1]])
  rho <- get_ro(normalized_features, data_matrix[[2]], weights,
    i)
  if (i == 1) {
    new_weight_i <- rho
  } else if (rho < -l1_penalty/2) {
    new_weight_i <- rho + l1_penalty/2
  } else if (rho > l1_penalty/2) {
    new_weight_i <- rho - l1_penalty/2
  } else {
    new_weight_i <- 0
  }
  return(new_weight_i)
}
```

Итак, при наличии функции, которая оптимизирует функцию стоимости по одной координате, реализуем циклический спуск координат, где отсортируем координаты от 1 до n по порядку и повторяем.

```
lasso_cyclical_coordinate_descent <- function(data_matrix, initial_weights,
  l1_penalty, tolerance) {
  normalized_features <- normalize_features(feature_Matrix = data_matrix
    [[1]])
  norms <- normalized_features[[2]]
  converged <- FALSE
  weights <- initial_weights
  new_weights <- vector(length = length(weights))
  while (!converged) {
    old_weights <- weights
    for (i in 1:length(weights)) {
      weights[i] <- lasso_coordinate_descent_step(i = i,
        data_matrix = data_matrix, weights = weights,
        l1_penalty = l1_penalty)
    }
    delta <- vector()
    for (i in 1:length(weights)) {
      delta[i] <- old_weights[i] - weights[i]
    }
    if (max(abs(delta)) < tolerance) {
      converged <- TRUE
    }
  }
```

```
  weights/norms
}
```

4.4 Реализация координатного спуска

При обучении будет использоваться набор данных KC house, состоящий из 21613 наблюдений. Разделим его на тренировочные и тестовые данные, состоящие из 19451 и 2162 наблюдений соответственно. Будем искать параметры для «sqft_living» и «bedrooms», и выполним функцию lasso_cyclical_coordinate_descent со штрафом $l_1 = 0$, что аналогично регрессии LS.

```
features = c("sqft_living", "bedrooms")
target = "price"
data_matrix <- construct_features_matrix(features = features, outputs =
    target,
  data = house_train_data)
weights <- lasso_cyclical_coordinate_descent(data_matrix = data_matrix,
  initial_weights = c(0, 0, 0), l1_penalty = 0, tolerance = 1e-07)
weights
```

[1] 97050.0942 305.2103 -57429.9302

Сравним результаты с функциями glm и lm в R.

```
x = model.matrix(price ~ sqft_living + bedrooms, data = house_train_data)[,
  -1]
y = house_train_data$price
lasso_model = glmnet(x, y, alpha = 1, lambda = 0, thresh = 1e-07)
predict(lasso_model, type = "coefficients", s = 0, exact = T)[1:3]
```

[1] 97052.1826 305.1896 -57417.7511

```
lm(price ~ sqft_living + bedrooms, data = house_train_data)$coef
```

(Intercept)	sqft_living	bedrooms
97050.0942	305.2103	-57429.9302

После выполнения алгоритма для регрессии лассо получаем схожие результаты.

В регрессии лассо, когда $\lambda = 0$, решение находится методом наименьших квадратов, а когда λ становится достаточно большим, регрессия лассо дает нулевую модель, в которой все оценки коэффициентов равны нулю. На рисунке 7 построены графики семи коэффициентов для регрессии лассо при изменении λ от 0 до $1e+10$. Ось абсцисс – натуральный логарифм λ. Ось ординат – значения соответствующих параметров. Можно заметить, что между двумя крайностями подгонки LS и нулевой модели, в зависимости от значения λ, лассо создает модели с любым количеством переменных, и большинство из них отбрасывается по мере увеличения значения λ.

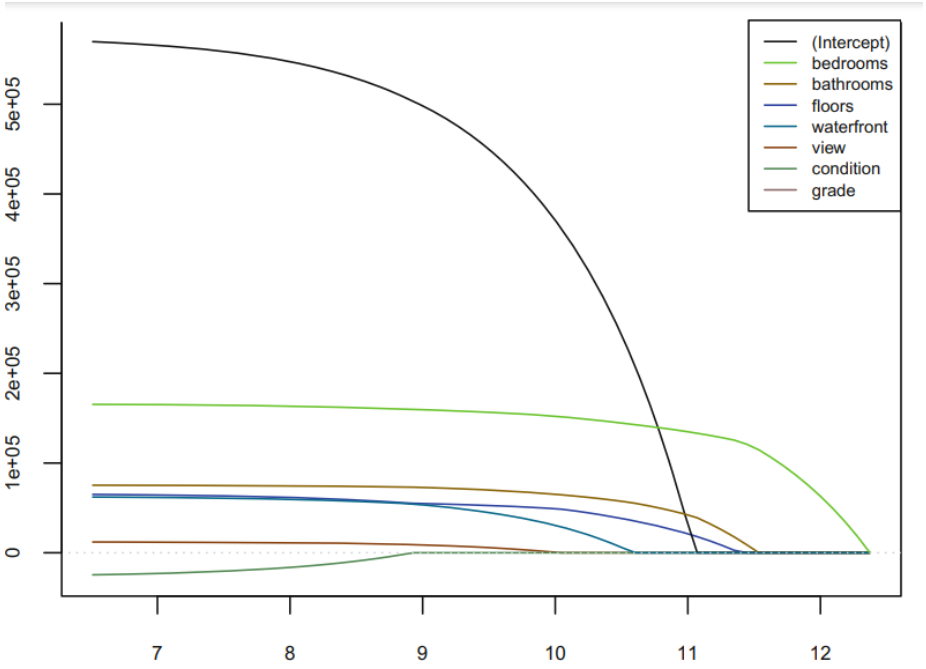

Рисунок 7 – Коэффициенты регрессии Лассо при изменении λ. Коэффициенты резко уменьшаются почти до 0 при $\lambda \to \infty$

4.5 Компромисс отклонения в регрессии лассо

Большое значение λ приведет к высокому смещению (простая модель) и, следовательно, к низкой дисперсии ($\hat{w} = 0$, для $\lambda = \infty$). Небольшое значение λ приведет к низкому смещению и, как следствие, большой дисперсии.

5 Разработка программной реализации построения регрессий с регуляризацией

Выполним реализацию построения регрессий на языке программирования Python.

В алгоритмах используется SGDClassifier [18]. SGDClassifier – линейный классификатор из библиотеки sklearn. Реализует обучение с помощью стохастического градиентного спуска. Поддерживает l_1 и l_2 регуляризации.

5.1 Постановка диагноза по симптомам

Приведём описание алгоритма применения регуляризации при построении модели, показывающей диагнозы пациентов.

1. Импортируем нужные библиотеки.
2. Поиск в открытых источниках данных по диагнозам. В работе используются данные с платформы kaggle.
3. Предварительная подготовка данных – чтение всех симптомов и диагнозов. Для каждого диагноза набор симптомов формируется следующим образом: создаётся словарь, где ключом является симптом, а значением – 0 или 1 в зависимости от наличия симптома для соответствующего диагноза.
4. Обучение модели без регуляризации и двух моделей с использованием регуляризации с помощью SGDClassifier.
5. Сравнение полученных результатов.

Используемые данные по диагнозам хранятся в двух файлах:

1. в файле "*Symptom−severity.csv*" хранится список всевозможных симптомов, которые могут встречаться в основном наборе данных и численная оценка их серьёзности;
2. в файле "*dataset.csv*" находится список диагнозов и соответствующих им симптомом.

Теперь приведём подробное описание действий программы.

С помощью библиотеки *csv* считываем из файла "*Symptom−severity.csv*" список симптомов и сохраняем его в словарь *symptoms*

```
import csv
symptoms = dict()
with open('Symptom-severity.csv', newline='') as csvfile:
```

```
    symptomsreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in symptomsreader:
        symptom, weight = row[0].split(',')
        if (symptom != "Symptom"):
            symptoms[symptom] = int(weight)
```

Определяем функцию, которая принимает на вход список симптомов *old_list*, а возвращает вектор, в котором на местах, соответствующих симптомам из *old_list* стоят положительные числа, взятые из словаря *symptoms*, а на всех остальных стоят нули.

```
def to_vec(old_list):
    new_list = list()
    for symptom in symptoms.keys():
        if old_list.count(symptom) > 0:
            new_list.append(symptoms[symptom])
        else:
            new_list.append(0)
    return new_list
```

Из файла "*dataset.csv*" построчно берутся диагнозы и их симптомы. Диагнозы помещаются в список *disease*, а список симптомов, преобразованный с помощью функции *to_vec* в *diseaseSymptoms*.

```
diseaseSymptoms = list()
disease = list()

with open('dataset.csv', newline='') as csvfile:
    datasetreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in datasetreader:
        new_row = list()
        for smth in row:
            new_smth = smth.strip()
            if (new_smth != ''):
                new_row.append(new_smth)
        if (new_row[0] != 'Disease'):
            disease.append(new_row[0])
            diseaseSymptoms.append(to_vec(new_row[1:]))
```

С помощью функции *train_test_split* из библиотеки *sklearn* разбиваем данные, полученные на предыдущем шаге на обучающую и тестовую выборки

```
from sklearn.model_selection import train_test_split

features_train, features_test, target_train, target_test = train_test_split(
    diseaseSymptoms, disease, test_size=0.98)
```

С помощью $SGDClassifier$ обучаем 3 модели: clf – без регуляризации, $clf1$ – l_1-регуляризации, $clf2$ – l_2-регуляризации.

```
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

clf = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty =
    'none'))
clf.fit(features_train, target_train)

clf2 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
   = 'l1'))
clf2.fit(features_train, target_train)

clf3 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
   = 'l2'))
clf3.fit(features_train, target_train)
```

Для проверки вводим случайный тест и выводим результат применения к нему моделей. С помощью функции $accuracy\_score$ из библиотеки $sklearn$ считаем точность каждой модели на обучающей и тестовой выборке:

```
from sklearn.metrics import accuracy_score

mytest = ['fatigue', 'loss_of_smell', 'obesity', 'excessive_hunger', '
    increased_appetite', 'cough']
print(clf.predict([to_vec(mytest)]))
print(clf2.predict([to_vec(mytest)]))
print(clf3.predict([to_vec(mytest)]))

print(accuracy_score(clf.predict(features_train), target_train))
print(accuracy_score(clf.predict(features_test), target_test))

print(accuracy_score(clf2.predict(features_train), target_train))
print(accuracy_score(clf2.predict(features_test), target_test))

print(accuracy_score(clf3.predict(features_train), target_train))
print(accuracy_score(clf3.predict(features_test), target_test))
```

Результат выполнения:

```
['Diabetes']
['Diabetes']
['Diabetes']
```

```
1.0
0.9253421816673579
1.0
0.9464952301949399
1.0
0.9477395271671506
```

Все три модели показали идеальный результат на обучающей выборке. На тестовой результаты соответствуют ожиданиям: без применения регуляризации точность ниже, чем с её применением.

5.2 Выявление предрасположенности к болезням сердца

Краткий алгоритм работы программы:

1. Импортируем нужные библиотеки.
2. С платформы Kaggle получаем набор данных *HeartDiseaseDataset.csv* [19].
3. С помощью библиотеки csv производим чтение данных. Каждая модель должна подобрать значения столбца с названием target, используя остальные столбцы.
4. Обучение модели без регуляризации и двух моделей с использованием регуляризации с помощью SGDClassifier.
5. Сравнение полученных результатов.

В файле *HeartDiseaseDataset.csv* содержатся следующие столбцы:

1. age – возраст человека в годах;
2. sex – пол человека (1 – если мужской, 0 – женский);
3. cp – тип боли в груди, приведённый к числовому формату;
4. trestbps – артериальное давление в покое;
5. chol – уровень холестерина в крови;
6. fbs – уровень сахара в крови натощак;
7. restecg – результаты электрокардиографии в покое;
8. thalach – максимальный уровень пульса;
9. exang – наличие стенокардии, вызванная физической нагрузкой;
10. oldpeak – депрессия сегмента ST при нагрузке в сравнении с состоянием покоя;

11. slope – наклон сегмента ST при пиковой нагрузке;
12. ca – количество крупных сосудов, окрашенных флурозопией;
13. target – наличие предрасположенности к болезням сердца.

Всего в описанном наборе данных имеется 303 строки.

С помощью библиотеки csv из имеющегося набора данных считываем значения. В списке disease храним значения для столбца target. В списке diseaseSymptoms – все остальные значения из соответствующей строки.

```
import csv

diseaseSymptoms = list()
disease = list()

with open('Heart Disease Dataset.csv', newline='') as csvfile:
    datasetreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in datasetreader:
        new_row = list()
        for smth in row:
            new_smth = smth.strip()
            if (new_smth != ''):
                new_row.append(new_smth)
        if (new_row[0] != 'age'):
            disease.append(new_row[-1])
            diseaseSymptoms.append(new_row[:-2])
```

Разделяем данные на обучающие и тестовые:

```
from sklearn.model_selection import train_test_split

features_train, features_test, target_train, target_test = train_test_split(
    diseaseSymptoms, disease, test_size=0.2)
```

Обучаем модель без использования регуляризаций и с применением l_1, l_2 регуляризаций.

```
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

clf = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty =
    'none'))
clf.fit(features_train, target_train)

clf2 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
   = 'l1'))
```

Таблица 3 – Матрица для модели без использования регуляризации

29.5%	14.75%
14.75%	41%

Таблица 4 – Матрица для модели с использованием l_1 регуляризации

27.9%	16.4%
8.2%	47.5%

```
clf2.fit(features_train, target_train)

clf3 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
    = 'l2'))
clf3.fit(features_train, target_train)
```

Считаем точность полученных моделей на тестовых данных с помощью функции *accuracy_score*.

```
from sklearn.metrics import accuracy_score


print(accuracy_score(clf.predict(features_test), target_test))

print(accuracy_score(clf2.predict(features_test), target_test))

print(accuracy_score(clf3.predict(features_test), target_test))
```

Итак, на тестовом наборе данных модель без использования регуляризации – 70.5%. С использованием l_1 регуляризации – 75.4%, l_2 – 77%.

Построим матрицу несоответствий для каждой модели [20]. С помощью функции *confusion_matrix* посчитаем для каждой модели вероятности дать истинно положительный, ложно положительный, ложно отрицательный и истинно отрицательный ответ:

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(target_test, clf.predict(features_test))/len(
    target_test))
print(confusion_matrix(target_test, clf2.predict(features_test))/len(
    target_test))
print(confusion_matrix(target_test, clf3.predict(features_test))/len(
    target_test))
```

Оформим вывод в виде таблиц 3, 4 и 5.

Таблица 5 – Матрица для модели с использованием l_2 регуляризации

34.4%	9.9%
13.1%	42.6%

Сумма диагональных значений для каждой модели равняется значениям, полученным с помощью функции accuracy_score как сумма истинно положительных и истинно отрицательных ответов.

ЗАКЛЮЧЕНИЕ

Поставленные задачи были успешно выполнены. Были рассмотрены основные понятия машинного обучения, осуществлено знакомство с гребневой регрессией, регрессией по методу "лассо". Успешно освоен язык программирования Python. Проведена разработка программы для решения задачи построения регрессии с регуляризацией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Машинное обучение — это легко [Электронный ресурс]: [сайт]. - URL: https://habr.com/ru/post/319288/ (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

2 Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. / Пер. с польского И.Д. Рудинского - М.: Горячая линия - Телеком, – 2006. - 452 с.

3 Machine Learning. Linear Models. Part 1. [Электронный ресурс]: [сайт]. - URL: https://medium.com/pharos-production/machine-learning-linear-models-part-1-312757aab7bc (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

4 Головко В. А. Нейронные сети: обучение, организация и применение. / В. А. Головко, В. В. Краснопрошин. – Минск: БГУ, – 2017. – 263 с.

5 Галушкин А.И. Нейрокомпьютеры и их применение. Книга 3. Нейрокомпьютеры / А.И. Галушкин М.: ИПРЖР, – 2000. - 528 с.

6 Машинное обучение (курс лекций, К.В.Воронцов) [Электронный ресурс]: [сайт]. - URL: www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_%28курс_лекций%2C_К.В.Воронцов%29 (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

7 Нейросети и глубокое обучение, глава 3, ч.2: почему регуляризация помогает уменьшать переобучение? [Электронный ресурс]: [сайт]. - URL: https://habr.com/ru/post/459816/ (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

8 Регуляризация - Линейные модели: классификация и практические аспекты | Coursera [Электронный ресурс]: [сайт]. - URL: https://ru.coursera.org/lecture/supervised-learning/rieghuliarizatsiia-sR94Q (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

9 Галушкин А.И. Нейрокомпьютеры и их применение. Книга 1. Теория нейронных сетей / А.И. Галушкин М.: ИПРЖР, – 2000. - 416 с.

10 L1 и L2-регуляризация для логистической регрессии [Электронный ресурс]: [сайт]. - URL: https://craftappmobile.com/l1-и-l2-регуляризация-для-логистической-р/ (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

11 Осовский С. Нейронные сети для обработки информации / Пер. с польского И.Д. Рудинского М.:Финансы и статистика, – 2002. - 344 с.

12 Ghatak, A. Machine Learning with R / A. Ghatak. - Singapore: Springer Nature, – 2017. - 210 pp.

13 The difference between L1 and L2 regularization [Электронный ресурс]: [сайт]. - URL: https://explained.ai/regularization/L1vsL2.html (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

14 How to Improve a Neural Network With Regularization [Электронный ресурс]: [сайт]. - URL: https://towardsdatascience.com/how-to-improve-a-neural-network-with-regularization-8a18ecda9fe3 (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

15 Neural Networks and Deep Learning [Электронный ресурс]: [сайт]. - URL: http://neuralnetworksanddeeplearning.com/chap3.html (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

16 Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. / А.Н. Горбань, Д.А. Россиев. - Новосибирск: Наука. Сибирская издательская фирма РАН, – 1996. - 276 с.

17 Простыми словами о методах решения проблем с переобучением [Электронный ресурс]: [сайт]. - URL: https://newtechaudit.ru/overfitting/ (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

18 sklearn.linear_model.SGDClassifier – scikit-learn 0.24.2 documentation [Электронный ресурс]: [сайт]. - URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

19 Heart Disease Dataset | KaggleHeart Disease Dataset | Kaggle [Электронный ресурс]: [сайт]. - URL: https://www.kaggle.com/zeeshanmulla/heart-disease-dataset (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.англ.

20 Понимание метрик классификации Data Science в Scikit-Learn в Python [Электронный ресурс]: [сайт]. - URL: https://www.machinelearningmastery.ru/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019/ (дата обращения: 04.06.2021). - Загл. с экрана. - Яз.рус.

ПРИЛОЖЕНИЕ А

Программный код для задачи постановки диагноза по симптомам

```
import csv

symptoms = dict()

with open('Symptom-severity.csv', newline='') as csvfile:
    symptomsreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in symptomsreader:
        symptom, weight = row[0].split(',')
        if (symptom != "Symptom"):
            symptoms[symptom] = int(weight)

diseaseSymptoms = list()
disease = list()

def to_vec(old_list):
    new_list = list()
    for symptom in symptoms.keys():
        if old_list.count(symptom) > 0:
            new_list.append(symptoms[symptom])
        else:
            new_list.append(0)
    return new_list

with open('dataset.csv', newline='') as csvfile:
    datasetreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in datasetreader:
        new_row = list()
        for smth in row:
            new_smth = smth.strip()
            if (new_smth != ''):
                new_row.append(new_smth)
        if (new_row[0] != 'Disease'):
            disease.append(new_row[0])
            diseaseSymptoms.append(to_vec(new_row[1:]))

from sklearn.model_selection import train_test_split

features_train, features_test, target_train, target_test = train_test_split(
    diseaseSymptoms, disease, test_size=0.98)

import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
```

```
44 from sklearn.pipeline import make_pipeline
45
46 clf = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty =
       'none'))
47 clf.fit(features_train, target_train)
48
49 clf2 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
      = 'l1'))
50 clf2.fit(features_train, target_train)
51
52 clf3 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
      = 'l2'))
53 clf3.fit(features_train, target_train)
54
55 from sklearn.metrics import accuracy_score
56
57 mytest = ['fatigue', 'loss_of_smell', 'obesity', 'excessive_hunger', '
      increased_appetite', 'cough']
58 print(clf.predict([to_vec(mytest)]))
59 print(clf2.predict([to_vec(mytest)]))
60 print(clf3.predict([to_vec(mytest)]))
61
62 print(accuracy_score(clf.predict(features_train), target_train))
63 print(accuracy_score(clf.predict(features_test), target_test))
64
65 print(accuracy_score(clf2.predict(features_train), target_train))
66 print(accuracy_score(clf2.predict(features_test), target_test))
67
68 print(accuracy_score(clf3.predict(features_train), target_train))
69 print(accuracy_score(clf3.predict(features_test), target_test))
```

ПРИЛОЖЕНИЕ Б

Программный код для задачи выявления предрасположенности к болезням сердца

```
import csv

diseaseSymptoms = list()
disease = list()

with open('Heart Disease Dataset.csv', newline='') as csvfile:
    datasetreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in datasetreader:
        new_row = list()
        for smth in row:
            new_smth = smth.strip()
            if (new_smth != ''):
                new_row.append(new_smth)
        if (new_row[0] != 'age'):
            disease.append(new_row[-1])
            diseaseSymptoms.append(new_row[:-2])


from sklearn.model_selection import train_test_split

features_train, features_test, target_train, target_test = train_test_split(
    diseaseSymptoms, disease, test_size=0.2)


import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

clf = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty =
    'none'))
clf.fit(features_train, target_train)

clf2 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
    = 'l1'))
clf2.fit(features_train, target_train)

clf3 = make_pipeline(StandardScaler(), SGDClassifier(max_iter=1000, penalty
    = 'l2'))
clf3.fit(features_train, target_train)


```

```
40 from sklearn.metrics import accuracy_score
41
42 print(accuracy_score(clf.predict(features_test), target_test))
43
44 print(accuracy_score(clf2.predict(features_test), target_test))
45
46 print(accuracy_score(clf3.predict(features_test), target_test))
47
48
49 from sklearn.metrics import confusion_matrix
50
51 print(confusion_matrix(target_test, clf.predict(features_test))/len(
       target_test))
52 print(confusion_matrix(target_test, clf2.predict(features_test))/len(
       target_test))
53 print(confusion_matrix(target_test, clf3.predict(features_test))/len(
       target_test))
```