# **Оглавление**

[Оглавление 1](#_Toc589845906)

[Задача №1 2](#_Toc538678521)

[Вывод к заданию №1 2](#_Toc227569574)

[Задача №2 3](#_Toc646640874)

[Вывод к заданию №2 7](#_Toc512167882)

[Задача №3 7](#_Toc286210321)

[Вывод к заданию №3 10](#_Toc365063613)

[Задача №4 10](#_Toc1188381584)

[Вывод к задаче №4 12](#_Toc674136467)

[Задача №5 13](#_Toc1311400080)

[Заключение к заданию №5 16](#_Toc838101376)

[Задача №6 17](#_Toc870364849)

[Заключение к заданию №6 24](#_Toc1831745301)

[Список литературы 24](#_Toc676093865)

[Приложение 26](#_Toc1337903338)

[Приложение к задаче №1 26](#_Toc288917046)

[Приложение к задаче №2 29](#_Toc42412325)

[Приложение к задаче №3 30](#_Toc718410004)

[Приложение к задаче №4 30](#_Toc913359889)

[Приложение к задаче №5 30](#_Toc114906053)

[Приложение к задаче №6 30](#_Toc353774471)

# **Задача №1**

Необходимо загрузить данные из указанного набора и произвести следующие действия.

Набор данных: Swiss.

Объясняемая переменная: *Infant.Mortality.*

Регрессоры: *Agriculture, Examination*.

1. Оценить среднее значение, дисперсию и СКО переменных.

Используя встроенную функцию *mean*, *var*, *sd*, получаем все необходимые параметры (Таблица 1).

Таблица 1. Среднее значение, дисперсия и СКО *Agriculture, Examination* и *Infant.Mortality.*

|  |  |  |  |
| --- | --- | --- | --- |
| Переменная | Среднее значение | Дисперсия | СКО |
| *Agriculture* | 50.65957 | 515.7994 | 22.71122 |
| *Examination* | 16.48936 | 63.64662 | 7.977883 |
| *Infant.Mortality* | 19.94255 | 8.483802 | 2.912697 |

1. Построить зависимости вида y = a + bx, где y – объясняемая переменная, x – регрессор. Оценить, насколько «хороша» модель по коэффициенту детерминации R2. Оценить, есть ли взаимосвязь между объясняемой переменной и объясняющей переменной

Строим модель *Infant.Mortality ~ Agriculture.* Получаем уравнение вида

*Infant*.*Mortality* = 20.338 - 0.008 \* *Agriculure.* R2 равен 0.003704 (0,4%), количество звездочек у регрессора равно нулю. Это говорит о том, что между регрессором и объясняемым значением нет зависимости.

Теперь построим модель *Infant.Mortality~Examination.* Получаем уравнение вида

*Infant.Mortality = 20.62899 -0.04163 \* Examination*. R2 равен 0.013 (1,3 %), количество звездочек у регрессора равно нулю. Это указывает на то, что модель также не объясняет нашу переменную

Код решения задачи и сведения о проверенных моделях приведены в [Приложении к задаче 1](#Bookmark1)

# **Вывод к заданию №1**

1. Построенные модели не отображают зависимости детской смертности от процента мужского населения, занимающегося сельским хозяйством, и от процента людей, получивших максимальный балл на экзаменах при поступлении на военную службу.
2. Улучшить модели невозможно -- нужно менять регрессоры.
3. Из модели мы можем понять, что причиной низкой или высокой детской смертности может быть что-то другое, а не данные параметры.

# **Задача №2**

Необходимо загрузить данные из указанного набора и произвести следующие действия.

Набор данных: *attitude.*

Объясняемая переменная: *rating.*

Регрессоры: *complaints, learning , raises.*

1. Проверить регрессоры на линейную зависимость, построить модели для проверки на линейную зависимость и исключить те регрессоры, R^2 моделей которых высокий.

Строим модели *complaints~learning, complaints~raises, raises~learning,* результаты представлены в Таблице 2.

Таблица 2. проверка на линейную зависимость регрессоров путем перебора моделей.

|  |  |  |  |
| --- | --- | --- | --- |
| Регрессор 1 | Регрессор 2 | R^2 | Pr |
| *complaints* | *learning* | 0.3561 | 0.00050 |
| *complaints* | *raises* | 0.4478 | 5.27e-05 |
| *raises* | *learning* | 0.41 | 0.000138 |

Второй способ: в Таблице 3 представлен результат использования функцию *vif* для модели *rating ~ complaints + learning + raises*

Таблица 3. результат функции *vif*

|  |  |  |
| --- | --- | --- |
| *complaints* | *learning* | *raises* |
| 1.983336 | 1.856191 | 2.164556 |

Устанавливаем, что между *complaints* и *raises* присутствует небольшая линейная зависимость, поэтому мы можем исключить *raises* из расмотрения

1. Построить модель линейной регрессии между объясняемой переменной и регрессорами. Построить модели с использованием логарифмов и произведений регрессоров.

Строим модели между *rating* и различными комбинациями регрессоров, их произведения и логарифмов.

Таблица 4. результаты перебора моделей

|  |  |
| --- | --- |
| зависимость | R^2 |
| *rating ~ learning + complaints + raises* | 0.7083 |
| *rating ~ complaints* | 0.6813 |
| *rating ~ learning + complaints* | 0.708 |
| *rating ~ complaints + raises* | 0.6839 |
| *rating ~ learning + raises* | 0.4507 |
| *rating ~ I(log(complaints)) + learning* | 0.7047 |
| *rating ~ I(log(learning)) + complaints* | 0.7018 |
| *rating ~ I(log(complaints + learning))* | 0.6496 |
| *rating ~ I(log(complaints \* learning))* | 0.6439 |
| *rating ~ I(log(complaints)) + complaints + learning* | 0.7088\* |
| *rating ~ I(complaints \* learning)* | 0.6626 |
| *rating ~ I(learning \* raises) + complaints* | 0.7022 |
| *rating ~ I(complaints\*raises\*learning)* | 0.6057 |

\*есть линейная зависимость между регрессорами

Лучшей моделью, среди описанных в Таблице 4, по значению R^2 является зависимость *rating ~ learning + complaints* (Подробнее представлена в Таблице 5)*.* К тому же мы исключили *raises*, который был частично линейно зависим от *complaints.*

Таблица 5. модель *rating ~ learning + complaints.*

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Estimate* | *Std. Error* | *Pr(>|t|)* |
| *Intercept* | 9.8709 | 7.0612 | 0.174 |
| *learning* | 0.2112 | 0.1344 | 0.128 |
| *complaints* | 0.6435 | 0.1185 | 9.57e-06 |

1. Построить доверительный интервал для каждого регрессора лучшей модели и проанализировать его. Посчитать интервал для одного прогноза.

Находим критерий Стьюдента при помощи встроенной функции *qt*

Получаем критерий Стьюдента равный 2.048

Умножаем значение стандартной ошибки регрессора на значение критерия Стьюдента и прибавляем / вычитаем из значения коэффициента.

Получаем 2 промежутка:

[-0.0628;0.4852] - для *learning* (0 входит в промежуток, поэтому мы не можем отвергнуть статистическую гипотезу о том, что коэффициент равен 0)

[0.399788;0.887212] - для *complaints* (0 не входит в промежуток, так что мы можем отвергнуть статистическую гипотезу о том, что коэффициент равен 0)

Теперь посчитаем доверительный интервал для прогноза. Возьмем *complaints* = 60, *learning* = 55. Результаты приведены в Таблице 6.

Таблица 6. доверительный интервал

|  |  |  |
| --- | --- | --- |
| нижняя граница | прогноз модели | верхняя граница |
| 57.17933 | 60.09749 | 63.01565 |

Код и решение задания приведены в [Приложении к заданию 2.](#mark2)

# **Вывод к заданию №2**

1. Построенные модели отображают зависимость рейтинга от работы с жалобами сотрудников. Скорее всего, чем больше финансовая организация работает с жалобами своих сотрудников, тем лучше ее оценивают. Соответственно, чем лучше организация слушает своих сотрудников, тем больше и повышений, это объясняет линейную зависимость между *complaints* и *raises* (данные приведены в таблице 5)*.*
2. Для улучшения модели возможно необходимо использовать еще другие регрессоры или же попробовать иные функции от уже имеющихся регрессоров (степень, деление и т.д)
3. Построен доверительный интервал (результаты приведены в Таблице 6), позволяющие довольно точно предсказать рейтинг организации.

# Задача №3

Необходимо загрузить данные из указанного набора и произвести следующие действия.

Набор данных: *r27i\_os\_31.csv*

Объясняемая переменная: *wj13.2 (salary)*

Регрессоры: *w\_age, wh5, w\_educ, status, wj6.2, w\_marst* + по собственному усмотрению *region, wj1.1.2,w\_occup08,wj23*.

1. Построить линейную регрессию зарплаты на все параметры, оценить коэффициент вздутия дисперсии *VIF*.

Производим нормализацию данных:

Обрезаем набор данных в 2 раза из-за поврежденности файла. Убираем все поля, не имеющие значений (*NaN* и специальные поля (пример 99999998)). Приводим данные к числовому виду (выделяем регрессоры, принимающие либо 1, либо 0). Переименуем регрессоры. Полученные регрессоры представлены в Таблице 7.

Таблица 7. регрессоры.

|  |  |
| --- | --- |
| регрессоры | значение |
| *salary* | зарплата |
| *age* | возраст |
| *sex* | пол |
| *higher\_educ* | Наличие законченного высшего образования |
| *city\_status* | Является ли населенный пункт городом или административным центром |
| *wed* | Семейный статус |
| *dur* | Продолжительность рабочего дня |
| *rich\_region* | Принадлежит ли населенный пункт к Москве, Санкт-Петербургу, Московской области, Ленинградской Области. |
| *enjoy* | Удовлетворенность работой |
| *prof* | Область деятельности |
| *gos* | Является ли государство владельцем предприятия |

Исследуем на линейную зависимость регрессоры. Исключаем зависимые.

1. Поэкспериментировать с функциями вещественных параметров, используя логарифмы, степени (хотя бы от 0.1 до 2 с шагом 0.1), произведения вещественных регрессоров, и выделить наилучшие модели из построенных.

Перебираем все комбинации логарифмов и степеней.

Таблица 8. Лучшая полученная модель(*R^2* = 0.2802)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *Estimate* | *Std. Error* | *t value* | *Pr(>|t|)* |
| *(Intercept)* | -0.73577 | 0.27731 | -2.653 | 0.009008 \*\* |
| *I(sign(age) \* log(abs(age)))* | 0.12610 | 0.27731 | -2.653 | 0.009008 \*\* |
| *I(sign(dur) \* (abs(dur)^1.1))* | 0.23852 | 0.07862 | 3.034 | 0.002937 \*\* |
| *sex* | 0.29120 | 0.19796 | 1.471 | 0.143809 |
| *higher\_educ* | 0.67349 | 0.18893 | 3.565 | 0.000517 \*\*\* |
| *city\_status* | 0.12209 | 0.19347 | 0.631 | 0.529155 |
| *wed2* | -0.26967 | 0.19612 | -1.375 | 0.171588 |
| *wed3* | -0.10005 | 0.22653 | -0.442 | 0.659502 |
| *rich\_region* | 0.76207 | 0.30957 | 2.462 | 0.015191 \* |
| *enjoy* | 0.47663 | 0.18881 | 2.524 | 0.012843 \* |
| *prof3* | -0.38554 | 0.19856 | -1.942 | 0.054431 |
| *prof4* | 0.34780 | 0.24279 | 1.433 | 0.154490 |
| *gos* | -0.02974 | 0.16809 | -0.177 | 0.859864 |

Проанализировав м сравнив все модели, получим лучшую, представленную в Таблице 8. Все зависимости от регрессоров положительные, кроме *wed2, wed3, prof3, gos.*

Есть слабая зависимость от того, в "богатом" ли регионе живет человек и наслаждается ли он работой; зависимость от степени 1.1 времени работы и сильная зависимость от наличия высшего образования.

1. Сделайте вывод о том, какие индивиды получают наибольшую зарплату.

Вывод по полученной модели: зарплата выше у тех, кто живет в Москве, Санкт-Петербурге или Московской области, имеет высшее образование и имеет много рабочих часов (ключевые факторы), также играет роль удовлетворение работой.

1. Оцените лучшие модели для подмножества индивидов: “Городские жители, женщины, не состоявшие в браке” и “разведенные женщины, с высшим образованием”

Выделяем подмножество данных и пропускаем через лучшую модель. Результаты представлены в Таблице 9.

Таблица 9. Срезы данных и R^2 моделей построенных на этих срезах.

|  |  |
| --- | --- |
| подмножество | *R^2* |
| Городские жители, женщины, не состоявшие в браке | 0.1432 |
| разведенные женщины, с высшим образованием | 0.4407 |

Получаем, что

1. Подмножество индивидов “Городские жители, женщины, не состоявшие в браке” крайне плохо описывается моделью.
2. Подмножество индивидов “разведенные женщины, с высшим образованием” очень хорошо описываются моделью.

Код и решение задания приведены в [Приложение к заданию №3.](#zad3)

# **Вывод к заданию №3**

1. Построенные модели пытаются отражать зависимость зарплаты от региона проживания, от удовлетворенности работой, от наличия высшего образования, от продолжительности рабочего дня и возраста.
2. Чем больше человек работает, чем больше у него рабочий стаж, тем больше он получает. Если есть высшее образование, то человек может устроиться на более высокооплачиваемую работу. Скорее всего, удовлетворенность, наоборот, зависит от зарплаты, чем выше зарплата, тем больше удовлетворенность. И в Москве платят больше. Данный вывод сделан по результатам Таблицы 8.
3. Для улучшения модели можно попробовать взять другие регрессоры, а также попробовать другие функции от регрессоров или другие степени.

# **Задача №4**

Необходимо загрузить данные из указанного набора и произвести следующие действия.

Набор данных: *kc\_house\_data.csv*

Классификация по столбцу: *bedrooms*

Классификатор: *Logistic regression* и *Random Forest*

1. Обработать набор данных набор данных. Разделить набор данных на тестовую и обучающую выборку. Построит классификатор и оценить точность построенного классификатора с помощью метрик *precision, recall* и *F1* на тестовой выборке.

Загружаем данные, анализируем набор данных и проводим предобработку:

* Столбец *bedrooms* если *<=* 3 присваиваем 1 иначе 0.
* Столбец *view* если == 0 присваиваем 0 иначе 1.
* Столбец *condition* если> = 4 присваиваем 1 иначе 0.
* Столбец yr\_*renovated* если> 100 присваиваем 1 иначе 0.
* Столбец *sqft*\_*lot15* если>10000 присваиваем 1 иначе 0.

Разделяем набор данных на 2 поднабора -- тренировочный и тестовый наборы данных. Выборка производится случайным образом в соотношении 3 к 7 соответственно.

Строим классификатор логистической регрессии и обучаем его на выборке.

Производим предсказание для тестовой выборки. Полученные результаты сопоставляем с реальными данными для расчета метрик вручную и встроенными функциями *python* (представлено в Таблице 10)

Таблица 10. Метрики для модели логистической регрессии.

|  |  |  |
| --- | --- | --- |
| метрика | Подсчет вручную | Функции |
| *precision* | 0.7392651615759186 | 0.7414352437999796 |
| *recall* | 0.6357061286638751 | 0.6243157704146297 |
| *F1* | 0.683585755218993 | 0.6776567973236963 |

Как видим, значения идентичны с точностью до сотой.

Точность модели довольно высокая, но далеко не идеальная.

* 74% того, что модель отнесет к 1 группе будет действительно принадлежать 1 группе.
* Из всего, что относится к 1 группе, будет найдено 63%.

1. Построить классификатор типа Случайный Лес и оценить его качество с помощью метрик *precision, recall* и *F1* на тестовой выборке. С помощью *GridSearch* перебрать различные комбинации гиперпараметров: на первой итерации задайте большие шаги (50 или 100) по числу деревьев *n\_estimators*. На следующих итерациях определите лучшее количество деревьев *n\_estimators* с точностью до 10. Выделить лучший классификатор.

Данные уже преобразованы, поэтому можем сразу строить модель случайного леса для случайного числа деревьев. Пусть это будет 210. (результаты представлены в Таблице 11).

Глубина составляет от 1 до 20. При оценивании используется критерий Джини.

Теперь перебираем модели с числом деревьев от 100 до 300 с шагом 50. Промежуток делаем не слишком большим, чтобы не было переобучения. Выделяем лучшую модель. (результаты представлены в Таблице 11).

Теперь перебираем модели с числом деревьев от 100 до 350 с шагом 10. Выделяем лучшую модель.

Таблица 11. Метрики моделей случайного леса.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *n\_estimators* | *max\_depth* | *precision* | *recall* | *F1* |
| 210 | *none* | 0.7368109172 | 0.7453521636 | 0.7387856818 |
| 250 | 19 | 0.7350434070 | 0.7518240086 | 0.7416167155 |
| 320 | 18 | 0.7368172063 | 0.7499243165 | 0.7426074271 |

Качество, судя по метрикам, даже у первой модели выше, чем у модели логистической регрессии.

Даже с шагом в 10 деревьев качество моделей остается примерно одинаковым в районе 0.73 -- 0.75.

Вывод: случайный лес дал модель лучше, чем логистическая регрессия, но при этом, для получения лучшей модели из все возможных вариантов, потребуется крайне много времени на обучение.

Код и решение задания приведены в [Приложение к заданию №4.](#_Приложение_к_задаче)

# **Вывод к задаче №4**

1. Удалось построить модели Логистической регрессии и Случайного лес, которые довольно хорошо способны классифицировать дома по количеству спален. Примерная точность составляет около 70%.
2. Для улучшения показателей можно попробовать иначе пред обработать набор данных. Также можно попробовать изменить соотношение тренировочной и тестовой выборки, хотя это может привести к переобучению, или наоборот данных будет недостаточно для наилучшего обучения.

# **Задача №5**

Найти имплементацию алгоритма Чена для двумерного случая

Алгоритм Чена — алгоритм построения выпуклой оболочки конечного множества точек на плоскости. Является комбинацией двух более медленных алгоритмов (сканирование по Грэхему O(nlogn) и заворачивание по Джарвису O(nh), где n - количество точек и h - количество точек в оболочке). Недостатком сканирования по Грэхему является необходимость сортировки всех точек по полярному углу, что занимает достаточно много времени O(nlogn). Заворачивание по Джарвису требует перебора всех точек для каждой из h точек выпуклой оболочки, что в худшем случае занимает O(n^2). Назван по имени Тимоти М. Чена.

Описание алгоритма:

Идея алгоритма Чена заключается в изначальном делении всех точек на группы по m

штук в каждой. Соответственно, количество групп равно r =nm. Для каждой из групп строится выпуклая оболочка сканированием по Грэхему за O(mlogm) ,то есть для всех групп понадобится O(rmlogm) = O(nlogm) времени. Затем, начиная с самой левой нижней точки, для получившихся в результате разбиения оболочек строится общая выпуклая оболочка по Джарвису. При этом следующая подходящая для выпуклой оболочки точка находится за O(rlogm), так как для того, чтобы найти точку с максимальным тангенсом по отношению к рассматриваемой точке в m-угольнике достаточно затратить O(logm) (точки в m-угольнике были отсортированы по полярному углу во время выполнения алгоритма сканирования Грэхема). В итоге, на обход требуется O(h\*r\*logm) = O((h\*n/m)logm) времени. То есть алгоритм Чена работает за O((n + h\*n/m)logm), при этом, если получить m = h, то за O(nlogh).

(Описание алгоритма и его сложности взято с [Wikipedia, State: Chan's algorithm).](#_Список_литературы)

Алгоритм Чена для построения выпуклой оболочки

|  |
| --- |
| std::vector<Point> convexHull(Point\* const points, const long long& len) {  if (points == NULL || len < 1) throw "wrong input";  if (len < 3)  return std::vector<Point>(points,points + len);  long long n = (long long)ceil(sqrt(len) - 1);  std::cout << std::endl << "n = " << n << std::endl;  std::vector<Point> tmp\_vect;  std::vector<Point> tmp;  long long len\_tmp = len / n;  long long len\_tmp1 = 0;  while (len\_tmp1 + len\_tmp <= len) {  tmp = GramconvexHull(&points[len\_tmp1], len\_tmp);  len\_tmp1 += len\_tmp;  tmp\_vect.insert(tmp\_vect.end(), tmp.begin(), tmp.end());  }  for (long long i = len\_tmp1; i < len; ++i) {  tmp\_vect.push\_back(points[i]);  }  return JarvisconvexHull(tmp\_vect);  } |

Листинг 1. Алгоритм Чена

Рассмотрим Листинг 1:

Функция *convexhull* принимает множество точек: массив точек и его длину.

Если множество точек меньше 3, то мы считаем, что это и есть выпуклая оболочка.

Иначе разбиваем множество на n подмножеств. Разбиение – простое разделение по порядку в исходном наборе.

Функция *Gramconvexhull* строит оболочку по подмножества сканированием Грэхема. (Использована реализация сканирования Грэхема: [Graham Scan algorithm implementation](#_Список_литературы) )

Записываем точки полученных оболочек и остаточные точки в *tmp\_vect.*

Функция *Jarvisconvexhull* строит оболочку алгоритмом «заворачивания Джарвиса» по полученным точкам оболочек разбиения и, таким образом, строит выпуклую оболочку по всему множеству точек. (Использована реализация алгоритма Джарвиса:[Jarvis March algorithm implementation](#_Список_литературы) )

Проведем несколько тестов на случайных точках.

Таблица 12. Тесты алгоритма на случайных точках

|  |  |  |
| --- | --- | --- |
| точки | Грэхем | Чен |
| 10000 | 1.198 | 0.961 |
| 100000 | 5.61 | 5.2 |
| 2000000 | 25.267 | 23.343 |

Некоторые тесты приведены в Таблице 12. Алгоритм опередил сканирование по Грэхема во всех тестах. Очевидно, что алгоритм быстрее заворачивания по Джарвису, так как его сложность O(n2), а Чена и Грэхема O(n log n).

Код решения задачи и сведения о проверенных моделях приведены в [Приложении к задаче 5.](#_Приложение_к_задаче_1)

# **Заключение к заданию №5**

1. . Алгоритм нахождения выпуклой оболочки показал отличные результаты на случайных точках, обогнав сканирование Грэхема и заворачивание Джарвиса по отдельности. Если попробовать вводить разные значения n (значение количества разбиений множества точек), можно добиться большей эффективности / скорости алгоритма.
2. Алгоритм поддерживает довольно простое распараллеливание, поэтому есть возможность увеличить скорость его работы в разы.

# **Задача №6**

Найти имплементацию алгоритма Чена для трехмерного случая

Идея алгоритма Чена также заключается в изначальном делении всех точек на группы по m штук в каждой. Соответственно, количество групп равно r = nm. Для каждой из групп строится выпуклая оболочка алгоритмом разделяй и властвуй Preparata-Hong за O(mlogm) ,то есть для всех групп понадобится O(rmlogm) = O(nlogm)) времени. Затем, начиная с самой левой нижней точки, для получившихся в результате разбиения оболочек строится общая выпуклая оболочка по аналогу Джарвису. В итоге, на обход требуется O(hr logm) = O((hnm)/logm) времени. То есть алгоритм Чена работает за O((n+nhm)/logm), при этом, если получить m = h, то за O(nlogh).

(Описание алгоритма взято со статьи [Timothy M. Chan, State: A Minimalist’s Implementation of the 3-d Divide-and-Conquer Convex Hull Algorithm.](#_Список_литературы) ).

Алгоритм *Preparata-Hong* заключается в делении отсортированного множества на 2 подмножества, пока в подмножество не будет состоять из <8 точек. По данные подмножествам ищутся оболочки при помощи алгоритма инкременты, а потом все оболочки постепенно соединяются в одну.

(Реализация алгоритма написана по псевдокоду [Roger Hernando, Slides: Covex hull algorithms in 3D](#_Список_литературы) )

|  |
| --- |
| std::vector<Point3D\_tri> PreparataHullBody(const std::vector<Point3D>& P) {  if (P.size() < 8) {  ConvexHull hull(P);  auto faces = hull.GetFaces();  std::vector<Point3D\_tri> answer;  if (faces.size() > 0) {  for (const auto& f : faces) {  answer.push\_back(Point3D\_tri(f.vertices[0], f.vertices[1], f.vertices[2]));  }  }  else {  return GiftWrap(P);  }  return answer;  }  std::vector<Point3D> P1(P.begin(), P.cbegin() + P.size() / 2);  std::vector<Point3D> P2(P.cbegin() + P.size()/2, P.cend());  std::vector<Point3D\_tri> H1 = PreparataHullBody(P1);  std::vector<Point3D\_tri> H2 = PreparataHullBody(P2);  return merge(H1, H2);  } |

Листинг 2. Алгоритм Preparata-Hong.

Рассмотрим Листинг 2: объект класса ConvexHull, и его метод GetFaces() находят оболочку алгоритмом инкремента. Если оболочку найти не удалось (все точки находятся в одной плоскости), то оболочка ищется алгоритмом заворачивания подарка.

(Реализация алгоритма инкремента взята [Incremental algorithm implementation by Dung-Han-Lee](#_Список_литературы) )

Функция *merge* соединяет оболочки.

В качестве аналога алгоритма Джарвиса используем алгоритм «заворачивания подарка». Суть алгоритма заключается в нахождении треугольника, который принадлежит выпуклой оболочки. После относительно всех граней этого треугольника ищутся новые треугольники при помощи сравнения ориентированных объемов тетраэрдов полученных относительно точки из множества точек и треугольника.

|  |
| --- |
| std::vector<Point3D\_tri> GiftWrap(const std::vector<Point3D>& P) {  Point3D\_tri t = FindTriangleOnHull(P);  std::vector<Point3D\_tri> Q;  Q.push\_back(Point3D\_tri(t.a, t.b, t.c));  Q.push\_back(Point3D\_tri(t.c, t.b, t.a));  Q.push\_back(Point3D\_tri(t.a, t.c, t.b));  std::vector<Point3D\_tri> H;  H.push\_back(t);  std::vector<Point3D\_tri> Proc;  while (Q.size() != 0) {  Point3D\_tri e = Q.back();  Q.pop\_back();  if (NotProcessed(e, Proc)) {  Point3D q = PivotAroundEdge(e.a, e.b, P, e.c);  t = Point3D\_tri(e.a, e.b, q);  add\_to\_vector<Point3D\_tri>(H, t);  Point3D\_tri tmp = Point3D\_tri(t.a, t.b, t.c);  Q.push\_back(tmp);  tmp = Point3D\_tri(t.c, t.b, t.a);  Q.push\_back(tmp);  tmp = Point3D\_tri(t.a, t.c, t.b);  Q.push\_back(tmp);  Proc.push\_back(e);  }  }  return H;  } |

Листинг 3. алгоритм Giftwrap.

Разберем некоторые функции из листинга 3. Функция *FindTriangleOnHull* находит стартовый треугольник, с которого начнется обход. Функция *PivotAroundEdge* реализует поиск следующего треугольника в оболочки.

Сам алгоритм Чена делит множество точек на n подмножеств и строит по ним выпуклые оболочки, а потом алгоритм заворачивания подарка строит уже по ним конечную выпуклую оболочку.

|  |
| --- |
| std::vector<Point3D\_tri>  convexHull(std::vector<Point3D> Point3Ds) {  long long len = Point3Ds.size();    long long n = (long long)pow(2,pow(2,ceil(log(log((len))))));  std::vector<Point3D> tmp\_vect;  std::vector<Point3D\_tri> tmp;  qsort(&Point3Ds[0], Point3Ds.size(), sizeof(Point3D), compare);  long long len\_tmp = len / n;  long long len\_tmp1 = 0;  while (len\_tmp1 + len\_tmp <= len) {  tmp = PreparataHullBody(std::vector<quickhull::Vector3<double>>(Point3Ds.cbegin() + len\_tmp1, Point3Ds.cbegin() + len\_tmp1 + len\_tmp));    for (long long int i = 0; i < tmp.size(); ++i) {  add\_to\_vector(tmp\_vect,tmp[i].a);  add\_to\_vector(tmp\_vect, tmp[i].b);  add\_to\_vector(tmp\_vect, tmp[i].c);  }  len\_tmp1 += len\_tmp;  }  for (long long i = len\_tmp1; i < len; ++i) {  tmp\_vect.push\_back(Point3Ds[i]);  }  return GiftWrap(tmp\_vect);  } |

Листинг 4. Алгоритм Чена.

Функция *qsort,* представленная в Листнге 4*,* сортирует множество, чтобы в начале была точка с минимальными *x, y, z.*

Функция *PreparataHullBody* выполняет рекурсивную часть алгоритма. Основная не нужна, так как уже множество отсортировано.

В *tmp\_vect* записываем все точки полученные алгоритмом.

Функция *GiftWrap* находит оболочку алгоритмом «заворачивание подарка»

Проведем некоторые тесты на случайных точках и сравним работу полученного алгоритма с алгоритмом *Preparata-Hong* и *Giftwrap*. Все тесты приведены в Таблице 13.

Таблица №13. Сравнение работы алгоритмов *Chan, Preparata-Hong, Giftwrap* в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество точек | *Chan* | *n* | *Preparata-Hong* | *Giftwrap* |
| 100 | 0.015 | 8 | 0.017 | 0.001 |
| 1000 | 0.138 | 8 | 0.142 | 0.011 |
| 10000 | 1.659 | 32 | 1.623 | 0.136 |
| 100000 | 15.963 | 21 | 19.092 | 13.699 |
| 100000 | 20.198 | 32 | 19.092 | 13.699 |
| 500000 | 80.5 | 32 | 78.114 | 106.098 |

Как мы видим из таблицы №13, алгоритм Чена обходит по скорости алгоритм *Praparata-Hong,* но как мы видим из теста на 100000 случайных точек скорость работы алгоритма очень сильно зависит от значения n. при разных значениях n алгоритм Чена способен показывать разные результаты. алгоритм Giftwrap был быстрее в некоторых тестах обоих оппонентов. Связано это с разбросом и неравномерностью случайных точек.

Основной отличительной чертой алгоритма Чена является возможность использования других алгоритмов построения оболочки вместо *Preparata-Hong* и *Giftwrap*. Так было получено еще 4 имплементации алгоритма Чена с использованием алгоритма инкремента и *quickhull* (Использована реализация [Quichhull implementation by Antti Kuukka](#_Список_литературы) ).

Сравним работу написанного алгоритма (лучшей версии) со встроенным алгоритмом поиска выпуклой оболочки в *Python.* Проведем ряд тестов на случайных точках.

Таблица №14. Сравнение работы встроенного алгоритма и имплементации.

|  |  |  |
| --- | --- | --- |
| Количество точек | Чен | *Python* |
| 100 | 0.002 | 0.002 |
| 1000 | 0.009 | 0.002 |
| 10000 | 0.092 | 0.013 |
| 100000 | 0.298 | 0.048 |
| 1000000 | 0.923 | 0.503 |
| 10000000 | 4.58 | 4.915 |

Из таблицы №14 видно, что разность (отставание) между алгоритмами не слишком сильно увеличивается, а в какой-то момент (10 000 000 точек) наша имплементация обгоняет встроенную в *Python.*

Также из некоторых строк таблицы №14 (100 и 1000 точек, 10 000 и 100 000 точек) можно заметить, что встроенная реализация *Python* является распараллеленной. Наш же алгоритм таковым не является, но все же ему удалось обойти “оппонента” в скорости. Это доказывает эффективность принципа разбиения на подмножества.

Приведем также некоторые визуализации работы алгоритма

(Представлены рисунками № 1–6)

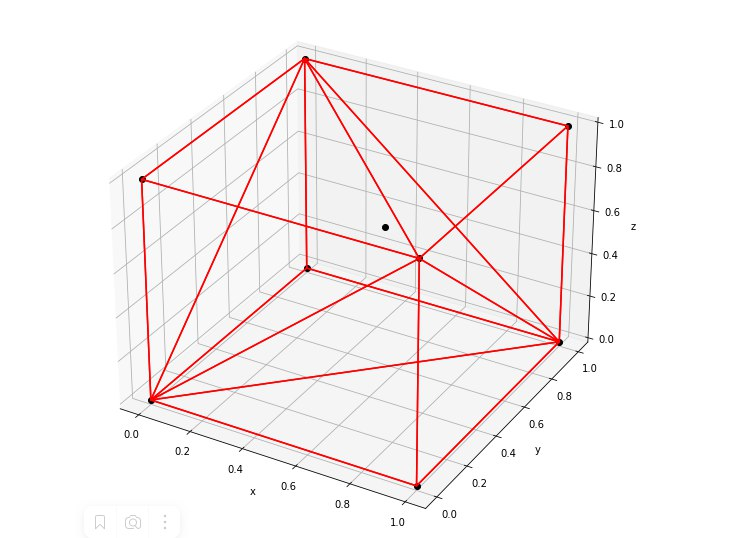


Рисунок 1. куб

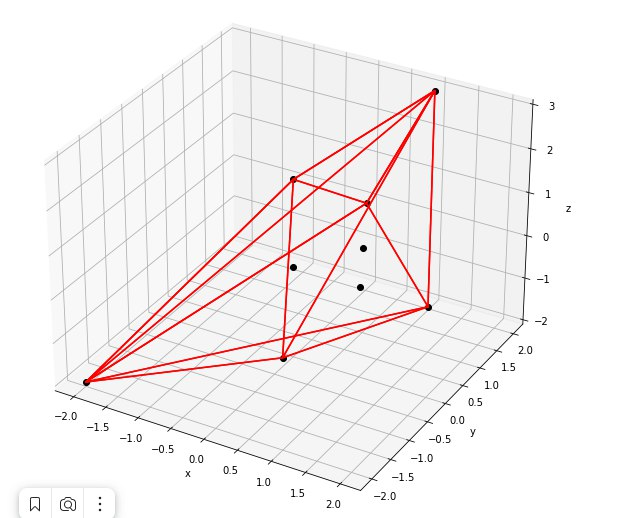


Рисунок 2. 9 точек

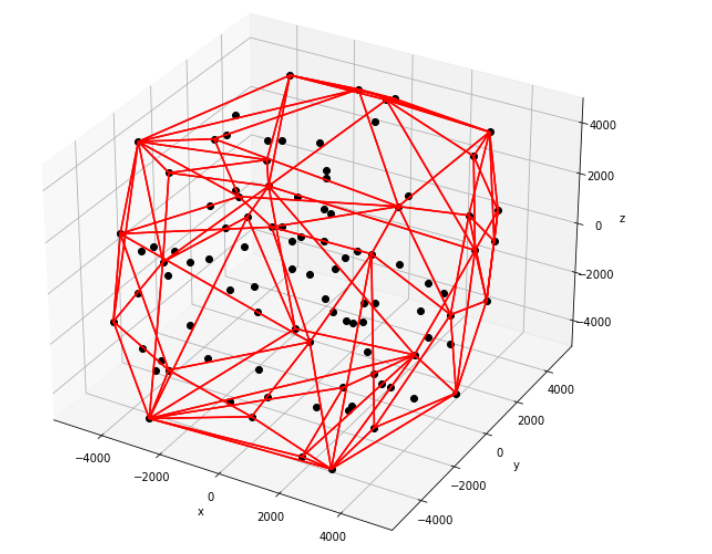


Рисунок 3 случайные точки

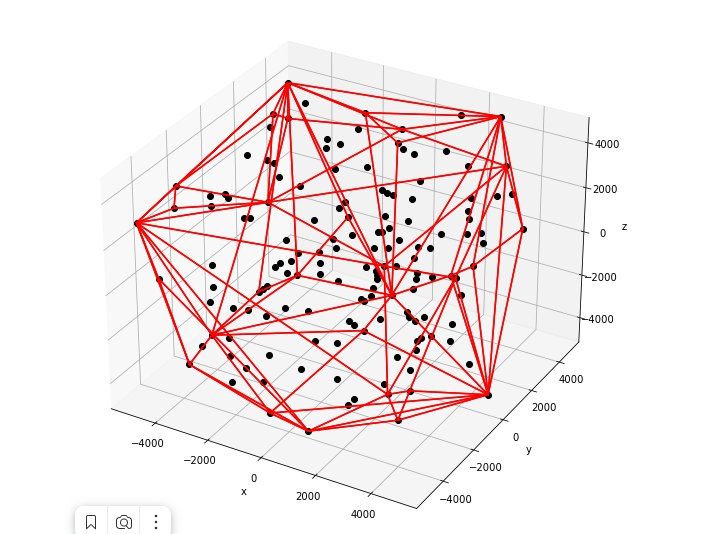


Рисунок 4 случайные точки

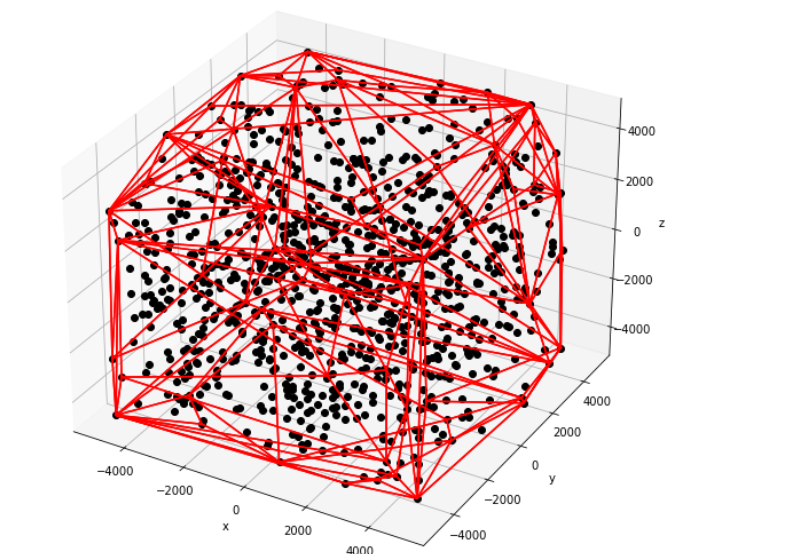


Рисунок 5 случайные точки

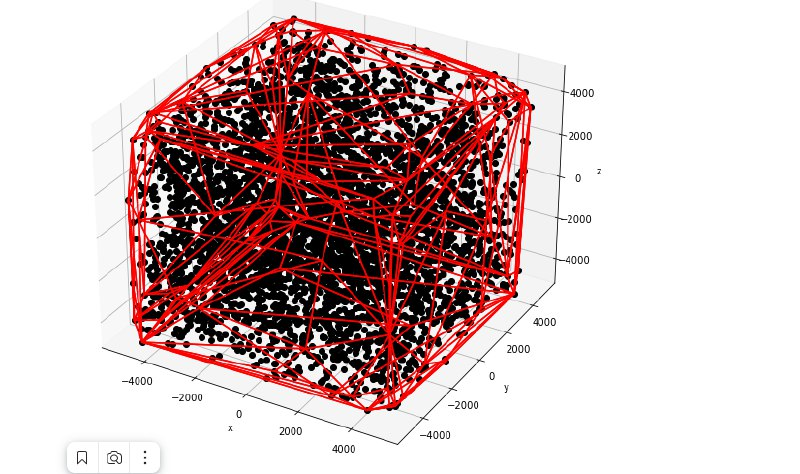


Рисунок 6 случайные точки

Из рисунков № 1—6 легко убедится в правильности работы алгоритма.

Код решения задачи и сведения о проверенных моделях приведены в

[Приложении к задаче 6.](#_Приложение_к_задаче_2)

# **Заключение к заданию №6**

1. Алгоритм реализован в 5 вариациях: обычная с алгоритмом *Praparata-Hong* и *Giftwrap*, *Quickhull* и *Giftwrap, Incremental и Quickhull* в 2-х вариациях*, Quick-Quickhull.*
2. Лучшая версия алгоритма обогнала (или не сильно отстала) по скорости встроенную реализацию в *Python*.
3. Скорость работы алгоритма не стабильна и очень сильно зависит от входного набора точек и параметра n числа разбиений.
4. Алгоритм поддерживает распараллеливание, поэтому можно добиться ускорения алгоритма в несколько раз.

# **Список** **литературы**

1. *Timothy M. Chan, State: A Minimalist’s Implementation of the 3-d Divide-and-Conquer Convex Hull Algorithm. - School of Computer Science University of Waterloo Waterloo, Ontario N2L 3G1, Canada June 4, 2003. --* [*https://tmc.web.engr.illinois.edu/ch3d/ch3d.pdf*](https://tmc.web.engr.illinois.edu/ch3d/ch3d.pdf)
2. *Roger Hernando, Slides: Covex hull algorithms in 3D. Roger Hernando – software engineer --* [*https://dccg.upc.edu/people/vera/wp-content/uploads/2014/11/GA2014-ConvexHulls3D-Roger-Hernando.pdf*](https://dccg.upc.edu/people/vera/wp-content/uploads/2014/11/GA2014-ConvexHulls3D-Roger-Hernando.pdf)
3. *Wikipedia, State: Chan's algorithm --* [*https://en.wikipedia.org/wiki/Chan%27s\_algorithm*](https://en.wikipedia.org/wiki/Chan%27s_algorithm)
4. *Jarvis March algorithm implementation --* [*https://www.sanfoundry.com/cpp-program-implement-jarvis-march-find-convex-hull/*](https://www.sanfoundry.com/cpp-program-implement-jarvis-march-find-convex-hull/)
5. *Graham Scan algorithm implementation --* [*https://www.geeksforgeeks.org/convex-hull-using-graham-scan/*](https://www.geeksforgeeks.org/convex-hull-using-graham-scan/)
6. *Quichhull implementation by Antti Kuukka:* [*https://github.com/akuukka/quickhull*](https://github.com/akuukka/quickhull)
7. *Incremental algorithm implementation by Dung-Han-Lee:* [*https://github.com/Dung-Han-Lee/Convexhull-3D-Implementation-of-incremental-convexhull-algorithm*](https://github.com/Dung-Han-Lee/Convexhull-3D-Implementation-of-incremental-convexhull-algorithm)

# **Приложение**

# Приложение к задаче №1

library("lmtest")

data = swiss

#используем набор данных swiss

#переменные:

# объясняемая: Infant.Mortality

# объясняющие: Agriculture | Examination

#задания:

#1) Оценить среднее значение, дисперсию и СКО переменных

mean(data$Infant.Mortality)

# 19.94255

mean(data$Agriculture)

# 50.65957

mean(data$Examination)

# 16.48936 процент людей занимающихся фермерским хозяйством больше, чем получивших высокие баллы на экзамене

var(data$Infant.Mortality)

# 8.483802 дисперсия мала, отклонение от среднего не велико

var(data$Agriculture)

# 515.7994 большая (первышает 100) дисперсия, отклонение от среднего может быть велико

var(data$Examination)

# 63.64662 средняя (меньше 100, больше 10) дисперсия...

sd(data$Infant.Mortality)

# 2.912697 СКО малое...

sd(data$Agriculture)

# 22.71122 СКО большое...

sd(data$Examination)

# 7.977883 СКО не большое...

#2) Построить зависимости вида y = a + bx 3) и 4) анализ моделей, оценка зависимости объясняемой переменной от объясняющих

model1 = lm(Infant.Mortality~Agriculture,data)

summary(model1)

#Infant.Mortality = 20.338 - 0.008 \* Agriculure

#зависимость отрицательная

#Pr(agriculture) = 0.684 (>0.005)

#причинно-следственной связи нет

#Pr(Coef) <2e-16

#есть зависимость от коэф-та

#p-value: 0.6845 > 0.05 (слишком большой)

#Multiple R-squared: 0.003704, Adjusted R-squared: -0.01844

#зависимости нет. модель не отображает действительности

model2 = lm(Infant.Mortality~Examination,data)

summary(model2)

#Infant.Mortality = 20.62899 -0.04163 \* Examination

# зависимость отрицательная

#Pr(Examination) = 0.445 (>0.005)

#причинно-следственной связи нет

#Pr(Coef) <2e-16

#есть зависимость от коэф-та

#p-value: 0.4454 > 0.05 (слишком большой)

#Multiple R-squared: 0.013, Adjusted R-squared: -0.008932

#зависимости нет. модель не отображает действительности

#Вывод:

#у Infant.Mortality нет реальных зависимостей от Examination и Agriculture

#детская смертность абосолютно не зависит от того, как были сданы экзамены, и от того, аграрный район это или нет

#причиной этого может быть то, что уровень медицины не отличался для людей работающих аграрных регионах и в армии

#Результаты:

#мы не получили моделей, способных предсказывать значение Infant.Mortality

#полученные модели (model1 | model2) не отображают действительности

# Приложение к задаче №2

library("lmtest")

library(car)  
   
#пакет attitude  
data = attitude  
#help(attitude)  
   
#переменные:  
# объясняемая  
# rating  
# регрессор   
# complaints | learning | raises  
   
mean(data$rating) #64.63333  
mean(data$complaints)#66.6  
mean(data$learning)#56.36667  
mean(data$raises)#64.63333  
   
var(data$rating)#148.1713  
var(data$complaints)#177.2828  
var(data$learning)#137.7575  
var(data$raises)#108.1023  
   
sd(data$rating)#12.17256  
sd(data$complaints)#13.31476  
sd(data$learning)#11.73701  
sd(data$raises)#10.39723  
   
#средние значение близки друг к другу  
#разброс значений большой  
   
#задача:  
#1) Проверьте, что в наборе данных нет линейной зависимости

#1. способ  
 model\_test\_1 = lm(complaints~learning,data)  
 summary(model\_test\_1)  
 #y = 28.442 + 0.677 \* x  
 # зависимость положительная  
 #pr = 0.00050  
 #R^2 = 0.3561  
 #p-value: 5e-04 < 0.05  
 #зависимость между регрессорами присутствует  
 #   
 model\_test\_2 = lm(complaints~raises,data)  
 summary(model\_test\_2)  
 #y = 11.21 + 0.857\*x  
 # зависимость положительная  
 #pr = 5.27e-05  
 #R^2 = 0.4478  
 #p-value: 5.268e-05 < 0.05  
 # зависимость между регрессорами присутствует  
 #  
   
 model\_test\_3 = lm(raises~learning,data)  
 summary(model\_test\_3)  
 #y = 32.6609 + 0.5672 \* x  
 # зависимость положительная  
 #pr = 0.000138  
 #R^2 = 0.41  
 #p-value: 0.0001384 < 0.05  
 #зависимость между регрессорами присутствует  
 #  
   
 #2. способ  
 model1 = lm(rating ~ complaints + learning + raises, data) #!  
 summary(model1)  
 vif(model1)  
 #rating = 10.523 + 0.653 \* complaints + 0.221 \* learning - 0.029 \* raises  
 #pr (complaints) = 5.82e-05 rating зависит от данной переменной  
 #pr (learning) = 0.152 rating не зависит от данной переменной  
 #pr (raises) = 0.876 rating не зависит от данной переменной  
 #p-value: 3.957e-07 < 0.05 | R^2 = 0.7083  
 #vif ~~ 2 для каждого регрессорра (есть зависимость)  
 #  
 #зависимость между регрессорами есть  
 ##регрессоры линейно выражаются друг через друга  
   
#2) Постройте линейную модель  
#mode1 показала, что rating лучше всего выражается через complaints  
#попробуем поменять местами регрессоры и/или убрать линейно выражающиеся  
#  
   
#уже найдена модель, отображающая зависимость rating от регрессоров (model1), проверим есть ли более точная модель  
   
model2 = lm(rating ~ learning + complaints + raises, data)  
summary(model2)  
#результат идентичный  
#rating лучше всего выражается через complaints  
#убираем линейновыражающиеся переменные  
   
model3 = lm(rating ~ complaints, data)  
summary(model3)  
#R^2 = 0.6813  
#p-value: 1.988e-08 < 0.05  
#модель стала менее точной (судя по R^2)  
#  
   
model4 = lm(rating ~ learning + complaints, data) #!  
summary(model4)  
# rating = 9.8709 + 0.2112 \* learning + 0.6435 \* complaints  
#зависимость положительная для обоих регрессоров  
#pr(learning) = 0.128 > 0.05  
#pr(complaints) = 9.57e-06 < 0.05  
#R^2 = 0.708  
#p-value: 6.058e-08  
#модель стала лучше по R^2, но p-value увеличился  
   
model5 = lm(rating ~ complaints + raises, data)  
summary(model5)  
#R^2 = 0.6839  
#p-value: 1.769e-07  
#модель хуже (судя по R^2)  
#модель значимо не отличается от модели 3, в будущем исключим аргумент raises  
   
modeljoke = lm(rating ~ learning + raises, data)  
summary(modeljoke)  
#R^2 = 0.4507  
#pr(learning) = 0.0333 < 0.05  
#pr(raises) = 0.0930 > 0.05  
#нет зависимости от данных переменных (что странно, так как у них есть связь с complaints)  
   
#3) Введем в модель логарифмы регрессоров  
   
#исключим raise, так как с этим аргументом модели практически не меняются  
model\_1 = lm(rating ~ I(log(complaints)) + learning, data)  
summary(model\_1)  
#R^2 = 0.7047 > 0.7  
#p-value: 7.06e-08 < 0.05  
#pr(I) = 1.12e-05 < 0.05  
#зависимость описывает rating  
#модель приемлима  
   
model\_2 = lm(rating ~ I(log(learning)) + complaints, data)  
summary(model\_2)  
#R^2 = 0.7018 > 0.7  
#p-value:8.034e-08 < 0.05  
#pr(I) = 0.184 > 0.05  
#зависимость описывает rating  
#модель приемлима, но log не улучшил результат  
   
model\_3 = lm(rating ~ I(log(complaints + learning)), data)  
summary(model\_3)  
#R^2 = 0.6496 < 0.7  
#p-value:7.204 7.68e-08 < 0.05  
#pr(I) = 7.676e-08 < 0.05  
#зависимость хуже описывает rating  
#модель приемлима, гораздо хуже предыдущих  
   
model\_4 = lm(rating ~ I(log(complaints \* learning)), data)  
summary(model\_4)  
#R^2 = 0.6439 < 0.7  
#p-value:9.636e-08 < 0.05  
#pr(I) = 9.64e-08 < 0.05  
#зависимость хуже описывает rating  
#модель приемлима, но такая же, как и \_3  
   
model\_5 = lm(rating ~ I(log(complaints)) + complaints + learning, data)  
summary(model\_5)  
#R^2 = 0.7088> 0.7  
#p-value: 3.866e-07 < 0.05  
#pr(I) = 0.791 < 0.05  
#зависимость приемлимо описывает rating  
#в модели нет высого уровня зависимости от переменных...  
#модель "плохая"  
   
#вывод: log от переменных не позволиил прийти к лучшей модели  
#  
   
#4) Введите в модель всевозможные произведения пар регрессоров.Найдите одну или несколько наилучших моделей по доле объяснённого разброса в данных R^2  
   
   
а  
   
model\_\_1 = lm(rating ~ I(complaints \* learning), data)  
summary(model\_\_1)  
#R^2 = 0.6626 < 0.7  
#p-value: 4.476e-08< 0.05  
#pr(I) = 4.48e-08 < 0.05  
#зависимость хуже описывает rating  
   
model\_\_2 = lm(rating ~ I(learning \* raises) + complaints , data)  
summary(model\_\_2)  
#R^2 = 0.7022 > 0.7  
#p-value: 7.896e-08 < 0.05  
#pr(I) = 0.1798 > 0.05  
#введение I не влияет на результат  
   
   
model\_\_3 = lm(rating ~ I(complaints\*raises\*learning), data)  
summary(model\_\_3)  
#R^2 = 0.6057 < 0.7  
#p-value:4.135e-07 < 0.05  
#pr(I) = 4.13e-07 < 0.05  
#зависимость плохо описывает rating  
#худшая модель с complaints  
   
#изучая зависимость rating от complaints, raises и learning, можно прийти к выводу  
#rating больше всего зависит от complaints и не зависит от raises  
#логарифмическая йункция и произведение регрессовров не улучшили показатели переменных  
#  
#лучшие модели - model1 и model4  
#R^2 > 0.7  
   
   
#теперь найдем доверительный интервал для model4 при p = 95%  
#p = 95%; p1 = p + (100 - p)/2 | p1 = 97.5%  
#p2 = 0.975  
#число замеров 30 | число регрессоров 2 => 30 - 2 = 28  
#найдем t-критерия Стьюдента  
t = qt(0.975,28)  
#t = 2.048 не лучший, но не худший результат  
   
#стандартная ошибка learning\_q = 0.134  
#стандартная ошибка complaints\_q = 0.119  
   
   
#доверительный интервал для learning = [0.2112 - 0.274;0.2112 + 0.274] = [-0.0628;0.4852]  
#0 принадлежит интервалу, так что мы не можем отвергнуть статистическую гипотезу о том, что коэффициент равен 0  
   
#доверительный интервал для complaints = [0.6435 - 0.119 \* 2.048; 0.6435 + 0.119 \* 2.048] = [0.399788;0.887212]  
#0 не принадлежит интервалу, так что мы можем отвергнуть статистическую гипотезу о том, что коэффициент равен 0  
   
#теперь посчитаем доверительный интервал для прогноза  
#возьмем complaints = 60, learning = 55  
new.data = data.frame(complaints = 60, learning = 55)  
predict(model4,new.data,interval = "confidence")  
#прогноз модели 60.09749  
#нижняя граница 57.17933  
#верхняя граница 63.01565

# Приложение к задаче №3

#install.packages("devtools")  
#devtools::install\_github("bdemeshev/rlms")  
   
library("lmtest")  
library("rlms")  
library("dplyr")  
library("GGally")  
library("car")  
library("sandwich")  
   
data <- read.csv("D:**\\**data\_R\_1**\\**r27i\_os\_31.csv")  
data3 = data[c(1:400),]  
data3  
glimpse(data3) #,wj11.1,wj1.1.1,,wj24  
data2 = select(data3, wj13.2, w\_age, wh5, w\_educ, status, wj6.2, w\_marst,region, wj1.1.2,w\_occup08,wj23)  
   
   
data2$wj13.2[which(data2$wj13.2 >= 99999990)] = NaN  
data2$w\_age[which(data2$w\_age >= 99999990)] = NaN  
data2$wh5[which(data2$wh5 >= 99999990)] = NaN  
data2$w\_educ[which(data2$w\_educ >= 99999990)] = NaN  
data2$status[which(data2$status >= 99999990)] = NaN  
data2$wj6.2[which(data2$wj6.2 >= 99999990)] = NaN  
   
data2$w\_marst[which(data2$w\_marst >= 99999990)] = NaN  
data2$region[which(data2$region >= 99999990)] = NaN  
data2$wj1.1.2[which(data2$wj1.1.2 >= 99999990)] = NaN  
data2$w\_occup08[which(data2$w\_occup08 >= 99999990)] = NaN  
data2$wj23[which(data2$wj23 >= 99999990)] = NaN  
   
data2 = na.omit(data2)  
glimpse(data2)  
#нормализация данных  
   
sal1 = as.character(data2$wj13.2)  
sal2 = lapply(sal1, as.integer)  
sal = as.numeric(unlist(sal2))  
mean(sal)  
sal[which(sal >= 99999990)] = NaN  
   
data2["salary"] = (sal - mean(sal)) / sqrt(var(sal))  
data2["salary"]  
   
age1 = as.character(data2$w\_age)  
age2 = lapply(age1, as.integer)  
age3 = as.numeric(unlist(age2))  
data2["age"]= (age3 - mean(age3)) / sqrt(var(age3))  
data2["age"]  
   
data2["sex"]=data2$wh5  
data2["sex"] = lapply(data2["sex"], as.character)  
data2$sex[which(data2$sex!='1')] <- 0  
data2$sex[which(data2$sex=='1')] <- 1  
data2$sex = as.numeric(data2$sex)  
   
data2["h\_educ"] = data2$w\_educ  
data2["h\_educ"] = lapply(data2["h\_educ"], as.character)  
data2["higher\_educ"] = data2$u\_educ  
data2["higher\_educ"] = 0  
data2$higher\_educ[which(data2$h\_educ=='21')] <- 1  
data2$higher\_educ[which(data2$h\_educ=='22')] <- 1  
data2$higher\_educ[which(data2$h\_educ=='23')] <- 1  
   
data2["status"]=data2$status  
data2["status"] = lapply(data2["status"], as.character)  
data2["city\_status"] = 0  
data2$city\_status[which(data2$status=='1')] <- 1  
data2$city\_status[which(data2$status=='2')] <- 1  
data2$city\_status = as.numeric(data2$city\_status)  
   
dur1 = as.character(data2$wj6.2)  
dur2 = lapply(dur1, as.integer)  
dur3 = as.numeric(unlist(dur2))  
data2["dur"] = (dur3 - mean(dur3)) / sqrt(var(dur3))  
   
data2["wed"]= data2$w\_marst  
data2["wed"] = lapply(data2["wed"], as.character)  
data2$wed1 = 0  
data2$wed1[which(data2$wed=='2')] <- 1  
data2$wed1 = as.numeric(data2$wed1)  
   
data2["wed2"] = lapply(data2["wed"], as.character)  
data2$wed2 = 0  
data2$wed2[which(data2$wed=='4')] <- 1  
data2$wed2[which(data2$wed=='5')] <- 1  
data2$wed2 = as.numeric(data2$wed2)  
   
data2["wed3"]=data2$w\_marst  
data2$wed3 = 0  
data2$wed3[which(data2$wed=='1')] <- 1  
data2$wed3 = as.numeric(data2$wed3)  
   
   
data2["rich\_reg"] = data2$region  
data2["rich\_reg"] = lapply(data2["rich\_reg"], as.character)  
data2["rich\_region"] = data2$rich\_reg  
data2["rich\_region"] = 0  
data2$rich\_region[which(data2$rich\_reg == '1')] <- 1  
data2$rich\_region[which(data2$rich\_reg == '138')] <- 1  
data2$rich\_region[which(data2$rich\_reg == '141')] <- 1  
data2$rich\_region[which(data2$rich\_reg == '142')] <- 1  
data2$rich\_region = as.numeric(data2$rich\_region)  
data2$rich\_region  
   
data2["en"]=data2$wj1.1.2  
data2["en"] = lapply(data2["en"], as.character)  
data2["enjoy"] = data2$en  
data2$enjoy = 0  
data2$enjoy[which(data2$en == '1')] <- 1  
data2$enjoy[which(data2$en == '2')] <- 1  
data2$enjoy = as.numeric(data2$enjoy)  
   
data2["prof"] = data2$w\_occup08  
data2["prof"] = lapply(data2["prof"],as.character)  
data2["prof1"] = data2$prof  
data2["prof2"] = data2$prof  
data2["prof3"] = data2$prof  
data2["prof4"] = data2$prof  
data2["prof5"] = data2$prof  
data2$prof1 = 0  
data2$prof2 = 0  
data2$prof3 = 0  
data2$prof4 = 0  
data2$prof5 = 0  
   
data2$prof1[which(data2$prof == '0')] <- 1  
   
data2$prof2[which(data2$prof == '1')] <- 1  
data2$prof2[which(data2$prof == '2')] <- 1  
   
data2$prof3[which(data2$prof == '3')] <- 1  
data2$prof3[which(data2$prof == '4')] <- 1  
data2$prof3[which(data2$prof == '5')] <- 1  
   
data2$prof4[which(data2$prof >= '6' & data2$prof <= '8')] <- 1  
   
data2$prof5[which(data2$prof == '9')] <- 1  
   
data2$prof1 = as.numeric(data2$prof1)  
data2$prof2 = as.numeric(data2$prof2)  
data2$prof3 = as.numeric(data2$prof3)  
data2$prof4 = as.numeric(data2$prof4)  
data2$prof5 = as.numeric(data2$prof5)  
   
   
data2["gos"] = data2$wj23  
data2["gos"] = lapply(data2["gos"],as.character)  
data2$gos[which(data2$gos == '1')] <- 1  
data2$gos[which(data2$gos != '1')] <- 0  
data2$gos = as.numeric(data2$gos)  
   
   
data2["salary"] = data2["salary"]\*1.0252\*1.034 #наверное надо  
glimpse(data2)  
   
#закончили нормализацию  
   
data\_t = select(data2,salary,age,sex,higher\_educ,city\_status, dur,wed1,wed2,wed3,rich\_region,enjoy,prof1,prof2,prof3,prof4,prof5,gos)  
   
model1 = lm(data = data2, salary~age + sex+higher\_educ+city\_status+dur+wed1+wed2+wed3+rich\_region+enjoy+prof1+prof2+prof3+prof4+prof5+gos)  
summary(model1)  
#Nan prof1, prof5  
   
model2 = lm(data = data2, salary~age + sex+higher\_educ+city\_status+dur+wed1+wed2+wed3+rich\_region+enjoy+prof2+prof3+prof4+gos)  
summary(model2)  
vif(model2)  
#R^2 = 0.2682  
   
model\_test =lm(data = data2, prof2~higher\_educ+wed1+wed2+wed3+prof3+prof4)  
summary(model\_test)  
vif(model\_test)  
#prof2 зависит от height-educ, prof3, prof4  
# R^2 = 0.7046  
   
model3 = lm(data = data2, salary~age + sex+higher\_educ+city\_status+dur+wed1+wed2+wed3+rich\_region+enjoy+prof3+prof4+gos)  
summary(model3)  
vif(model3)  
#R^2 = 0.2664  
   
model\_test1 = lm(data = data2, wed1~higher\_educ+wed2+wed3+prof3+prof4)  
summary(model\_test1)  
#wed1 зависит от wed2 wed3  
#R^2 =0.6519  
   
model4 = lm(data = data2, salary~age + sex+higher\_educ+city\_status+dur+wed2+wed3+rich\_region+enjoy+prof3+prof4+gos)  
summary(model4)  
vif(model4)  
#R^2 = 0.2667  
   
model\_test2 = lm(data = data2, prof4~higher\_educ+wed2+wed3+prof3)  
summary(model\_test2)  
#prof4 зависит от prof3 height\_educ  
#R^2 = 0.3466   
#можем оставить  
   
model5 = lm(data = data2, salary~age + sex+higher\_educ+city\_status+dur+wed2+wed3+rich\_region+enjoy+prof3+gos)  
summary(model5)  
vif(model5)  
#R^2 = 0.2644  
   
model6 = lm(data = data2, salary~I(sign(age)\*log(abs(age))) + sex+I(higher\_educ^0.2)+city\_status+dur+wed2+wed3+rich\_region+enjoy+prof3+prof4+gos)  
summary(model6)  
vif(model6)  
#R^2 = 0.28  
   
#очень большой и долгий перебор  
   
model448 = lm(data = data2, salary~ I( sign( age ) \*(abs( age ) ^ 2)) + I( sign( dur ) \*(abs( dur ) ^ 2)) + higher\_educ + city\_status + wed2 + wed3 + rich\_region + enjoy + prof3 + prof4 + gos)  
summary(model448)  
   
model449 = lm(data = data2, salary~ I( sign( age ) \*(abs( age ) ^ 2)) + I( sign( dur ) \*(abs( dur ) ^ 2)) + higher\_educ + city\_status + wed2 + rich\_region + enjoy + prof3 + prof4 + gos)  
summary(model449)  
   
model450 = lm(data = data2, salary~ I( sign( age ) \*(abs( age ) ^ 2)) + I( sign( dur ) \*(abs( dur ) ^ 2)) + higher\_educ + city\_status + wed2 + rich\_region + enjoy + prof3 + prof4)  
summary(model450)  
#R^2 = 0.2587  
   
model451 = lm(data = data2, salary~ I( sign( age ) \*(abs( age ) ^ 2)) + I( sign( dur ) \*(abs( dur ) ^ 2)) + higher\_educ + wed2 + rich\_region + enjoy + prof3 + prof4)  
summary(model451)  
#R^2 = 0.2631  
   
model452 = lm(data = data2, salary~ I( sign( dur ) \*(abs( dur ) ^ 2)) + higher\_educ + wed2 + rich\_region + enjoy + prof3 + prof4)  
summary(model452)  
#R^2 = 0.2684  
   
model453 = lm(data = data2, salary~ I( sign( dur ) \*(abs( dur ) ^ 2)) + wed2 + rich\_region + enjoy + prof3 + prof4)  
summary(model453)  
vif(model453)  
#R^2 =0.1903   
   
#худшая модель 453  
#лучшая модель 228  
#R^2 = 0.2802  
   
model228 = lm(data = data2, salary~ I( sign( age ) \* log( abs( age ))) + I( sign( dur ) \*(abs( dur ) ^ 1.1)) +sex + higher\_educ + city\_status + wed2 + wed3 + rich\_region + enjoy + prof3 + prof4 + gos)  
summary(model228)  
   
qt(0.975,136-12) #1.97928  
# Residuals:  
# Min 1Q Median 3Q Max   
# -2.2832 -0.4655 -0.1500 0.4632 3.0713   
#   
# Coefficients:  
# Estimate Std. Error t value Pr(>|t|)   
# (Intercept) -0.73577 0.27731 -2.653 0.009008 \*\*   
# I(sign(age) \* log(abs(age))) 0.12610 0.08469 1.489 0.139027   
# I(sign(dur) \* (abs(dur)^1.1)) 0.23852 0.07862 3.034 0.002937 \*\*   
# sex 0.29120 0.19796 1.471 0.143809   
# higher\_educ 0.67349 0.18893 3.565 0.000517 \*\*\*  
# city\_status 0.12209 0.19347 0.631 0.529155   
# wed2 -0.26967 0.19612 -1.375 0.171588   
# wed3 -0.10005 0.22653 -0.442 0.659502   
# rich\_region 0.76207 0.30957 2.462 0.015191 \*   
# enjoy 0.47663 0.18881 2.524 0.012843 \*   
# prof3 -0.38554 0.19856 -1.942 0.054431 .   
# prof4 0.34780 0.24279 1.433 0.154490   
# gos -0.02974 0.16809 -0.177 0.859864  
   
# все зависимости от регрессоров положительные, кроме wed2, wed3, prof3, gos  
# есть слабая зависимость от того, в "богатом" ли регионе живет человек и наслаждается ли он работой; зависимость от степени 1.1 времени работы и сильная зависимость от наличия высшего образования  
# вывод по полученной модели:зарплата выше у тех, кто живет в москве, спб или мос облати, имеет высшее образование и имеет много рабочих часов(ключевые факторы), также играет роль удовлетворение работой  
   
   
data\_sub\_1 = subset(data2,city\_status == 1)  
data\_sub\_2 = subset(data\_sub\_1,sex == 0)  
data\_sub\_3 = subset(data\_sub\_2,wed3 == 1)  
   
model\_sub\_1\_1 = lm(data = data\_sub\_3, salary~ I( sign( age ) \* log( abs( age ))) + I( sign( dur ) \*(abs( dur ) ^ 1.1)) +sex + higher\_educ + city\_status + wed2 + wed3 + rich\_region + enjoy + prof3 + prof4 + gos)  
summary(model\_sub\_1\_1) #R^2 = 0.1432  
   
model\_sub\_1\_2 = lm(data = data\_sub\_3, salary~ I( sign( age ) \* log( abs( age ))) + I( sign( dur ) \*(abs( dur ) ^ 1.1)) + higher\_educ + enjoy + prof3 + gos)  
summary(model\_sub\_1\_2) #R^2 = 0.1432  
   
   
data\_sub\_4 = subset(data2, sex == 0)  
data\_sub\_5 = subset(data\_sub\_4,higher\_educ == 1 )  
data\_sub\_6 = subset(data\_sub\_5,wed2 == 1 )  
   
   
model\_sub\_2\_1 = lm(data = data\_sub\_6,salary~ I( sign( age ) \* log( abs( age ))) + I( sign( dur ) \*(abs( dur ) ^ 1.1)) +sex + higher\_educ + city\_status + wed2 + wed3 + rich\_region + enjoy + prof3 + prof4 + gos)  
summary(model\_sub\_2\_1) #R^2 = 0.4407  
   
model\_sub\_2\_2 = lm(data = data\_sub\_6,salary~ I( sign( age ) \* log( abs( age ))) + I( sign( dur ) \*(abs( dur ) ^ 1.1)) + city\_status + rich\_region + enjoy + gos)  
summary(model\_sub\_2\_2) #R^2 = 0.4407  
   
#у вcех 4-х нет сильных зависимостей от регрессоров

# Приложение к задаче №4

**import** pandas  
import numpy **as** np  
import io  
from sklearn.linear\_model **import** LogisticRegression  
from sklearn **import** datasets  
from sklearn.model\_selection **import** train\_test\_split  
   
data = pandas.read\_csv('kc\_house\_data.csv', index\_col = 'id')  
data  
   
data.info()  
   
data.isnull().sum()  
   
data\_sel = data.loc[:,data.columns.isin(['price', 'bedrooms', 'bathrooms','sqft\_living','sqft\_lot','floors','waterfront','view','condition','grade','sqft\_above','sqft\_basement','yr\_renovated','yr\_built','lat','long','sqft\_living15','sqft\_lot15'])]  
data\_sel = data\_sel.dropna()  
data\_sel['bedrooms'] = np.where(data['bedrooms'] <= 3, 0, 1)  
data\_sel['view'] = np.where(data['view'] == 0, 0, 1)  
data\_sel['condition'] = np.where(data['condition'] >= 4, 1, 0)  
data\_sel['yr\_renovated'] = np.where(data['yr\_renovated'] > 100, 1, 0)  
data\_sel['sqft\_lot15'] = np.where(data['sqft\_lot15'] > 10000, 1, 0)  
   
Bedrooms = data\_sel.loc[:,data\_sel.columns.isin(['bedrooms'])]  
X = data\_sel.loc[:,data\_sel.columns.isin(['price', 'bathrooms','sqft\_living','sqft\_lot','floors','waterfront','view','condition','grade','sqft\_above','sqft\_basement','yr\_renovated','yr\_built','lat','long','sqft\_living15','sqft\_lot15'])]  
   
x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Bedrooms,test\_size = 0.3)  
   
classifier = LogisticRegression(random\_state = 0, solver = 'lbfgs', multi\_class = 'multinomial', penalty = 'l2')*#, penalty = 'elasticnet', l1\_ration = '0.5' #'saga'*   
classifier.fit(x\_train, y\_train)  
   
y\_pred = classifier.predict(x\_test)  
classifier.score(x\_test, y\_test)  
   
print(len(x\_test))  
print(len(y\_test))  
print(sum(y\_test['bedrooms']))  
print(sum(y\_pred))  
print(np.where(y\_test == 1)[0])  
   
TP = 0  
FP = 0  
FN = 0  
TN = 0  
for i **in** np.where(y\_test == 1)[0]:  
 **if** y\_pred[i] == 1:  
 TP+=1  
 **else**:  
 FN+=1  
for i **in** np.where(y\_test == 0)[0]:  
 **if** y\_pred[i] == 1:  
 FP+=1  
 **else**:  
 TN+=1  
print(TP,FP,FN, TN)  
precision = TP / (TP + FP)  
recall = TP/(TP + FN)  
F1 = 2\* (precision\*recall)/(precision+recall)  
print(precision, recall,F1)  
   
from sklearn.model\_selection **import** cross\_val\_score  
print("accuracy:"+str(np.average(cross\_val\_score(classifier, x\_test, y\_test, scoring='accuracy'))))  
print("f1:"+str(np.average(cross\_val\_score(classifier, x\_test, y\_test, scoring='f1'))))  
print("precision:"+str(np.average(cross\_val\_score(classifier, x\_test, y\_test, scoring='precision'))))  
print("recall:"+str(np.average(cross\_val\_score(classifier, x\_test, y\_test, scoring='recall'))))  
   
from sklearn.ensemble **import** RandomForestClassifier  
   
forest = RandomForestClassifier(n\_estimators = 210)  
   
print("accuracy:"+str(np.average(cross\_val\_score(forest, x\_test, y\_test, scoring='accuracy'))))  
print("f1:"+str(np.average(cross\_val\_score(forest, x\_test, y\_test, scoring='f1'))))  
print("precision:"+str(np.average(cross\_val\_score(forest, x\_test, y\_test, scoring='precision'))))  
print("recall:"+str(np.average(cross\_val\_score(forest, x\_test, y\_test, scoring='recall'))))  
   
from sklearn.model\_selection **import** GridSearchCV  
   
param\_grid = {'n\_estimators':[100,150,200,250,300],'max\_features':['auto'],'max\_depth': list(range(1,20))}  
forest = GridSearchCV(estimator = RandomForestClassifier(),param\_grid = param\_grid,cv = 5, refit = True)  
forest.fit(x\_train, y\_train)  
   
forest.best\_estimator\_  
y\_pred = forest.best\_estimator\_.predict(x\_test)  
print(y\_pred)  
   
print("accuracy:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='accuracy'))))  
print("f1:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='f1'))))  
print("precision:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='precision'))))  
print("recall:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='recall'))))  
   
forest.best\_estimator\_  
   
param\_grid = {'n\_estimators':list(range(100, 351, 10)),'max\_features':['auto'],'max\_depth': list(range(1,20)),'criterion': ['gini']}  
forest = GridSearchCV(estimator = RandomForestClassifier(),param\_grid = param\_grid,cv = 5, refit = True)  
forest.fit(x\_train, y\_train)  
   
forest.best\_estimator\_  
   
print("accuracy:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='accuracy'))))  
print("f1:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='f1'))))  
print("precision:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='precision'))))  
print("recall:"+str(np.average(cross\_val\_score(forest.best\_estimator\_, x\_test, y\_test, scoring='recall'))))

# Приложение к задаче №5

#include <iostream>  
#include <stack>  
#include <stdlib.h>  
   
   
#include <algorithm>  
#include <random>  
   
#include<time.h>  
#include<vector>  
   
struct Point  
{  
 double x;  
 double y;  
 Point(const double& inp\_x, const double& inp\_y) {  
 x = inp\_x; y = inp\_y;  
 }  
 Point() {  
 x = 0; y = 0;  
 }  
};  
   
// A global point needed for sorting points with reference  
// to the first point Used in compare function of qsort()  
Point p0;  
   
// A utility function to find next to top in a stack  
Point nextToTop(std::stack<Point>& S)  
{  
 Point p = S.top();  
 S.pop();  
 Point res = S.top();  
 S.push(p);  
 return res;  
}  
   
// A utility function to swap two points  
void swap(Point& p1, Point& p2)  
{  
 Point temp = p1;  
 p1 = p2;  
 p2 = temp;  
}  
   
// A utility function to return square of distance  
// between p1 and p2  
int distSq(const Point& p1, const Point& p2)  
{  
 return (p1.x - p2.x) \* (p1.x - p2.x) +  
 (p1.y - p2.y) \* (p1.y - p2.y);  
}  
   
// To find orientation of ordered triplet (p, q, r).  
// The function returns following values  
// 0 --> p, q and r are collinear  
// 1 --> Clockwise  
// 2 --> Counterclockwise  
int orientation(const Point& p, const Point& q, const Point& r)  
{  
 double val = (q.y - p.y) \* (r.x - q.x) -  
 (q.x - p.x) \* (r.y - q.y);  
   
 // collinear return 0  
 return (abs(val) < 0.001) ? 0 : (val > 0.001) ? 1 : 2; // clock or counterclock wise  
}  
   
// A function used by library function qsort() to sort an array of  
// points with respect to the first point  
int compare(const void\* vp1, const void\* vp2)  
{  
 Point\* p1 = (Point\*)vp1;  
 Point\* p2 = (Point\*)vp2;  
   
 // Find orientation  
 int o = orientation(p0, \*p1, \*p2);  
   
 return (o == 0) ? ( (distSq(p0, \*p2) >= distSq(p0, \*p1)) ? -1 : 1) : ((o == 2) ? -1 : 1);  
}  
   
// Prints convex hull of a set of n points.  
std::vector<Point> GramconvexHull(Point\* points, const long long& n)  
{  
 if (points == NULL || n < 1) throw "wrong input";  
 //long long n = points.size();  
 // Find the bottommost point  
 double ymin = points[0].y; long long min = 0;  
 for (long long i = 1; i < n; ++i)  
 {  
 double y = points[i].y;  
   
 // Pick the bottom-most or choose the left  
 // most point in case of tie  
 if ((y < ymin) || (ymin == y &&  
 points[i].x < points[min].x))  
 ymin = points[i].y, min = i;  
 }  
   
 // Place the bottom-most point at first position  
 swap(points[0], points[min]);  
   
 // Sort n-1 points with respect to the first point.  
 // A point p1 comes before p2 in sorted output if p2  
 // has larger polar angle (in counterclockwise  
 // direction) than p1  
 p0 = points[0];  
 qsort(&points[1], n - 1, sizeof(Point), compare);  
   
 // If two or more points make same angle with p0,  
 // Remove all but the one that is farthest from p0  
 // Remember that, in above sorting, our criteria was  
 // to keep the farthest point at the end when more than  
 // one points have same angle.  
 long long m = 1; // Initialize size of modified array  
 for (long long i = 1; i < n; ++i)  
 {  
 // Keep removing i while angle of i and i+1 is same  
 // with respect to p0  
 while (i < n - 1 && orientation(p0, points[i],  
 points[i + 1]) == 0)  
 i++;  
   
 points[m] = points[i];  
 m++; // Update size of modified array  
 }  
   
 // If modified array of points has less than 3 points,  
 // convex hull is not possible  
 if (m < 3) return std::vector<Point>(points, points + m);  
   
 // Create an empty stack and push first three points  
 // to it.  
 std::stack<Point> S;  
 S.push(points[0]);  
 S.push(points[1]);  
 S.push(points[2]);  
   
 // Process remaining n-3 points  
 for (long long i = 3; i < m; ++i)  
 {  
 // Keep removing top while the angle formed by  
 // points next-to-top, top, and points[i] makes  
 // a non-left turn  
 while (S.size() > 1 && orientation(nextToTop(S), S.top(), points[i]) != 2)  
 S.pop();  
 S.push(points[i]);  
 }  
   
 std::vector<Point> answer;  
 // Now stack has the output points  
 while (!S.empty())  
 {  
 answer.push\_back(S.top());  
 S.pop();  
 }  
 return answer;  
}  
   
   
   
   
//===========================================================================  
   
   
   
// Define Infinite (Using INT\_MAX caused overflow problems)  
   
   
std::vector<Point> JarvisconvexHull(std::vector<Point>& points)  
{  
 long long n = points.size();  
 // There must be at least 3 points  
 if (n < 3)   
 return points;  
   
   
 //========================================  
 double ymin = points[0].y; long long min = 0;  
 for (long long i = 1; i < n; ++i)  
 {  
 double y = points[i].y;  
   
 // Pick the bottom-most or choose the left  
 // most point in case of tie  
 if ((y < ymin) || (ymin == y &&  
 points[i].x < points[min].x))  
 ymin = points[i].y, min = i;  
 }  
   
 // Place the bottom-most point at first position  
 swap(points[0], points[min]);  
   
 // Sort n-1 points with respect to the first point.  
 // A point p1 comes before p2 in sorted output if p2  
 // has larger polar angle (in counterclockwise  
 // direction) than p1  
 p0 = points[0];  
 qsort(&points[1], n - 1, sizeof(Point), compare);  
   
 //=======================================  
   
 // Initialize Result  
 long long\* next;  
 if (!(next = new long long[n])) throw "memory limit";  
   
 //nessesary for printing results  
 for (long long i = 0; i < n; ++i)  
 next[i] = -1;  
   
 // Find the leftmost point  
 long long l = 0;  
 for (long long i = 1; i < n; ++i)  
 l = (points[i].x < points[l].x) ? i : l;  
   
   
   
 // Start from leftmost point, keep moving counterclockwise  
 // until reach the start point again  
 long long p = l, q;  
 do  
 {  
 // Search for a point 'q' such that orientation(p, i, q) is  
 // counterclockwise for all points 'i'  
 q = (p + 1) % n;  
 for (long long i = 0; i < n; ++i)  
 q = (orientation(points[p], points[i], points[q]) == 2)? i : q;  
 //when orientation is counterclockwise swap points(next point has more angle)  
   
 p = next[p] = q; // Add q to result as a next point of p  
 // q; // Set p as q for next iteration  
 } while (p != l);  
   
 std::vector<Point> answer;  
 // Result  
 for (long long i = 0; i < n; ++i)  
 {  
 if (next[i] != -1)  
 answer.push\_back(points[i]);  
 }  
   
 delete[]next;  
 return answer;  
}  
   
std::vector<Point> convexHull(Point\* const points, const long long& len) {  
 if (points == NULL || len < 1) throw "wrong input";  
   
 if (len < 3)  
 return std::vector<Point>(points,points + len);  
 //put n = sqrt(len)   
 //[len/n] groups  
 long long n = (long long)ceil(sqrt(len) - 1);  
   
 */\**  
 *std::cout << "\nenter n\n";*  
 *std::cin >> n;*  
 *\*/*  
  
//uncomment to use manual input n  
   
 std::cout << std::endl << "n = " << n << std::endl;  
   
 std::vector<Point> tmp\_vect;  
 std::vector<Point> tmp;  
 long long len\_tmp = len / n;  
 long long len\_tmp1 = 0;  
 while (len\_tmp1 + len\_tmp <= len) {  
 tmp = GramconvexHull(&points[len\_tmp1], len\_tmp);//subset of points  
 len\_tmp1 += len\_tmp;  
 tmp\_vect.insert(tmp\_vect.end(), tmp.begin(), tmp.end());  
 }  
 //if len/n has residue we should push points, which are not consisted in any subsets  
 for (long long i = len\_tmp1; i < len; ++i) {  
 tmp\_vect.push\_back(points[i]);  
 }  
 return JarvisconvexHull(tmp\_vect);  
}  
   
// Driver program to test above functions  
int main()  
{  
 srand(time(NULL));  
 const int RANDVALUE = 10000;  
   
 unsigned long long int len = 0;  
 std::cout << "enter amount of 2d points" << std::endl;  
 std::cin >> len;  
 Point\* points = new Point[len];  
 for (unsigned long long int i = 0; i < len; ++i) {  
 double x, y;  
 //y = (rand() % RANDVALUE - RANDVALUE / 2) \* sin(rand() % RANDVALUE), x = (rand() % RANDVALUE - RANDVALUE / 2) \* cos(rand() % RANDVALUE); //uncoment random input  
   
 std::cin >> x >> y;   
 points[i]=Point(x, y);  
 }  
 float timer1 = std::clock();  
 std::cout << "The points in the convex hull are: " << std::endl;  
   
 std::vector<Point> answer = convexHull(points, len);  
 std::cout << std::endl << "time: " << (float)(clock() - timer1) / CLOCKS\_PER\_SEC << std::endl;  
 for (int i = 0; i < answer.size(); ++i) {  
 std::cout << "( " << answer[i].x << " , " << answer[i].y << " )" << std::endl;  
 }  
 delete[]points;  
 return 0;  
}

# Приложение к задаче №6

#include <iostream>  
#include <stack>  
#include <stdlib.h>  
   
   
#include <algorithm>  
#include <random>  
   
#include<time.h>  
#include<vector>  
   
#include"incremental algorithm/incremental.h"  
   
   
   
struct Point3D\_pair { //edge  
 Point3D a, b;  
 Point3D\_pair(const Point3D& a, const Point3D& b) {  
 this->a = a; this->b = b;  
 }  
 bool operator==(const Point3D\_pair& p) {  
 return (a == p.a && b == p.b) || (a == p.b && b == p.a);  
 }  
};  
   
struct Point3D\_tri : public Point3D\_pair { //face  
 Point3D c;  
 Point3D\_tri(const Point3D& a, const Point3D& b, const Point3D& c) : Point3D\_pair(a, b) {  
 this->c = c;  
 }  
 Point3D\_tri(const Point3D\_pair& pair) : Point3D\_pair(pair.a, pair.b) {  
 this->c = Point3D(INFINITY, INFINITY, INFINITY);  
 }  
 bool operator==(const Point3D\_tri& p) {  
 return (a == p.a) && ((b == p.b && c == p.c)  
 || (b == p.c && c == p.b))  
 || (a == p.b) && ((b == p.a && c == p.c)  
 || (b == p.c && c == p.a))  
 || (a == p.c) && ((b == p.b && c == p.a)  
 || (b == p.a && c == p.b));  
 }  
};  
   
   
struct vector3d { //vector created by 2 points  
 double x, y, z;  
 vector3d(const Point3D& a, const Point3D& b) {  
 x = b.x - a.x;  
 y = b.y - a.y;  
 z = b.z - a.z;  
 }  
 vector3d(const double& x, const double& y, const double& z) {  
 this->x = x;  
 this->y = y;  
 this->z = z;  
   
 }  
 double operator/(const vector3d& p) { //scalar product vectors  
 return x \* p.x + y \* p.y + z \* p.z;  
 }  
 double operator/(const vector3d& p) const { //scalar product vectors  
 return x \* p.x + y \* p.y + z \* p.z;  
 }  
 vector3d operator\*(const vector3d& v) { //vector product vectors  
 return vector3d(y \* v.z - z \* v.y, z \* v.x - x \* v.z, x \* v.y - y \* v.x);  
 }  
 vector3d operator\*(const vector3d& v) const { ///vector product vectors  
 return vector3d(y \* v.z - z \* v.y, z \* v.x - x \* v.z, x \* v.y - y \* v.x);  
 }  
 double lenght() {  
 return sqrt(x \* x + y \* y + z \* z);  
 }  
 double lenght() const {  
 return sqrt(x \* x + y \* y + z \* z);  
 }  
};  
   
double Area(const Point3D& a, const Point3D& b, const Point3D& c) {  
 vector3d tmp = vector3d(a, b) \* vector3d(a, c);  
 return tmp.lenght() \* 0.5;  
}  
   
double Volume(const Point3D& a, const Point3D& b, const Point3D& c, const Point3D& d) {  
 vector3d tmp = vector3d(a, b) \* vector3d(a, c);  
 return (vector3d(a, d) / tmp) \* 0.166666667; //volume of tetrahedron  
}  
   
   
double angle(const vector3d& a, const vector3d& b) {  
 return acos((a / b) / a.lenght() / b.lenght());  
}  
   
Point3D PivotAroundEdge(const Point3D& q0, const Point3D& q1, const std::vector<Point3D>& P, const Point3D& start\_pnt) {  
 Point3D p = start\_pnt;  
 double area = Area(q0, q1, p), volume, area\_tmp;  
 for (int i = 0; i < P.size(); ++i) {  
 volume = Volume(q0, q1, p, P[i]);  
 if (volume < 0) {  
 p = P[i];  
 area = Area(q0, q1, p);  
 }  
 else if (volume == 0) {  
 area\_tmp = Area(q0, q1, P[i]);  
 if (area\_tmp > area && !(p == start\_pnt)) {  
 if (angle(vector3d(q0, q1), vector3d(q0, p)) + angle(vector3d(q0, q1), vector3d(q0, P[i])) - angle(vector3d(q0, p), vector3d(q0, P[i])) > 0.1) {  
 p = P[i];  
 area = area\_tmp;  
 }  
 }  
 else if (p == start\_pnt && !(P[i] == q1 || P[i] == q0 || P[i] == start\_pnt) &&  
 angle(vector3d(q0, q1), vector3d(q0, p)) + angle(vector3d(q0, q1), vector3d(q0, P[i])) - angle(vector3d(q0, p), vector3d(q0, P[i])) < 0.1) {  
 p = P[i];  
 area = area\_tmp;  
 }  
 }  
 }  
 return p;  
}  
   
   
Point3D\_tri FindEdgeOnHull(const std::vector<Point3D>& P) {  
 //find most bottom most far most back point  
 Point3D p = P[0];  
 Point3D p\_tmp = P[0];  
 for (int i = 1; i < P.size(); ++i) {  
 if ((p.z > P[i].z) || ((p.z == P[i].z) && ((p.y > P[i].y) || (p.y == P[i].y && p.x > P[i].x)))) {  
 p = P[i];  
 }  
 }  
 //find most top most close most front point  
 //temporery point for searching starting face  
 for (int i = 1; i < P.size(); ++i) {  
 if ((p.z < P[i].z) || ((p.z == P[i].z) && ((p.y < P[i].y) || (p.y == P[i].y && p.x < P[i].x)))) {  
 p\_tmp = P[i];  
 }  
 }  
   
 Point3D q = p;  
 for (int i = 0; i < P.size(); ++i) {  
 q =(q.z == P[i].z && q.y == P[i].y && q.x < P[i].x) ? P[i] : q;  
 }  
 if (q == p)  
 q = p + Point3D(1, 0, 0);  
 q = PivotAroundEdge(p, q, P, p\_tmp);  
   
 return Point3D\_tri(p, q, p\_tmp);  
}  
   
   
Point3D\_tri FindTriangleOnHull(const std::vector<Point3D>& P) {  
 Point3D\_tri tmp = FindEdgeOnHull(P);  
 tmp.c = PivotAroundEdge(tmp.a, tmp.b, P, tmp.c);  
 return tmp;  
}  
   
bool NotProcessed(const Point3D\_tri& e, const std::vector<Point3D\_tri>& Proc) {  
 for (int i = 0; i < Proc.size(); ++i) {  
 if ((e.a == Proc[i].a && e.b == Proc[i].b) || (e.b == Proc[i].a && e.a == Proc[i].b))  
 return false;  
 }  
 return true;  
}  
   
template<class T>  
void add\_to\_vector(std::vector<T>& H, const T& t) {  
 for (int i = 0; i < H.size(); ++i) {  
 if (H[i] == t) return;  
 }  
 H.push\_back(t);  
}  
   
std::vector<Point3D\_tri> GiftWrap(const std::vector<Point3D>& P) {  
 Point3D\_tri t = FindTriangleOnHull(P);  
   
 std::vector<Point3D\_tri> Q;  
 Q.push\_back(Point3D\_tri(t.a, t.b, t.c));  
 Q.push\_back(Point3D\_tri(t.c, t.b, t.a));  
 Q.push\_back(Point3D\_tri(t.a, t.c, t.b));  
   
 std::vector<Point3D\_tri> H;  
 H.push\_back(t);  
   
 std::vector<Point3D\_tri> Proc;  
   
 while (Q.size() != 0) {  
 Point3D\_tri e = Q.back();  
 Q.pop\_back();  
 if (NotProcessed(e, Proc)) {  
   
 Point3D q = PivotAroundEdge(e.a, e.b, P, e.c);  
   
 t = Point3D\_tri(e.a, e.b, q);  
 add\_to\_vector<Point3D\_tri>(H, t);  
   
 Point3D\_tri tmp = Point3D\_tri(t.a, t.b, t.c);  
 Q.push\_back(tmp);  
 tmp = Point3D\_tri(t.c, t.b, t.a);  
 Q.push\_back(tmp);  
 tmp = Point3D\_tri(t.a, t.c, t.b);  
 Q.push\_back(tmp);  
   
 Proc.push\_back(e);  
 }  
 }  
 return H;  
}  
   
   
   
   
std::vector<Point3D\_tri> merge(const std::vector<Point3D\_tri>& H1, const std::vector<Point3D\_tri>& H2) {  
 std::vector<Point3D> P1;  
   
 for (int i = 0; i < H1.size(); ++i) {  
 add\_to\_vector<Point3D>(P1, H1[i].a);  
 add\_to\_vector<Point3D>(P1, H1[i].b);  
 add\_to\_vector<Point3D>(P1, H1[i].c);  
 }  
   
 for (int i = 0; i < H2.size(); ++i) {  
 add\_to\_vector<Point3D>(P1, H2[i].a);  
 add\_to\_vector<Point3D>(P1, H2[i].b);  
 add\_to\_vector<Point3D>(P1, H2[i].c);  
 }  
   
 return GiftWrap(P1);  
}  
   
std::vector<Point3D\_tri> PreparataHullBody(Point3D\* const P, long long len) {  
   
 if (len < 8) {  
   
 //incremental algorithm;  
   
 ConvexHull hull(P,len);  
 auto faces = hull.GetFaces();  
 std::vector<Point3D\_tri> answer;  
 if (faces.size() > 0) {  
 for (const auto& f : faces) {  
 answer.push\_back(Point3D\_tri(f.vertices[0], f.vertices[1], f.vertices[2]));  
 }  
 }  
 else {  
 //if all points in same plane  
 return GiftWrap(std::vector<Point3D>(P,P+len));  
 }  
 return answer;   
   
 }  
 //divide  
 std::vector<Point3D\_tri> H1 = PreparataHullBody(P,len/2);  
 std::vector<Point3D\_tri> H2 = PreparataHullBody(&P[len/2],len - len/2);  
 return merge(H1, H2);  
}  
   
   
   
int compare(const void\* vp1, const void\* vp2)  
{  
 Point3D\* p1 = (Point3D\*)vp1;  
 Point3D\* p2 = (Point3D\*)vp2;  
 return (p1 < p2) ? 1 : -1;  
}  
//full preparata requare sort inside  
//we've already sorted array  
/\*  
std::vector<Point3D\_tri> PreparataHull(Point3D\* P,long long len) {  
 qsort(P, len, sizeof(Point3D), compare);  
 return PreparataHullBody(P,len);  
}  
\*/  
   
//======================================================  
std::vector<Point3D\_tri>   
convexHull(Point3D\* Point3Ds, long long len) {  
   
 long long n = (long long)pow(2,pow(2,ceil(log(log((len))))));  
 std::cout << std::endl << "n = " << n << std::endl;  
 //std::cout << "enter n: ";  
 //std::cin >> n;  
 //for manual input  
   
 std::vector<Point3D> tmp\_vect;  
 std::vector<Point3D\_tri> tmp;  
 qsort(&Point3Ds[0], len, sizeof(Point3D), compare);  
 long long len\_tmp = len / n;  
 long long len\_tmp1 = 0;  
   
 while (len\_tmp1 + len\_tmp <= len) {  
 tmp = PreparataHullBody(&Point3Ds[len\_tmp1],len\_tmp);  
   
 for (long long int i = 0; i < tmp.size(); ++i) {  
 add\_to\_vector(tmp\_vect,tmp[i].a);  
 add\_to\_vector(tmp\_vect, tmp[i].b);  
 add\_to\_vector(tmp\_vect, tmp[i].c);  
 }  
 len\_tmp1 += len\_tmp;  
 }  
 for (long long i = len\_tmp1; i < len; ++i) {  
 tmp\_vect.push\_back(Point3Ds[i]);  
 }  
   
 return GiftWrap(tmp\_vect);  
}  
   
   
//======================================================  
   
   
int main()  
{  
 //srand(time(NULL));  
 const int RANDVALUE = 10000;  
   
   
   
   
 unsigned long long int len = 0;  
 std::cout << "enter amount of 3d Point3Ds" << std::endl;  
 std::cin >> len;  
 Point3D\* Point3Ds = new Point3D[len];  
 for (unsigned long long int i = 0; i < len; ++i) {  
 double x, y, z;  
 //y = (rand() % RANDVALUE - RANDVALUE / 2) , x = (rand() % RANDVALUE - RANDVALUE / 2) , z = (rand() % RANDVALUE - RANDVALUE / 2);  
 //for random input  
   
 std::cin >> x >> y >> z;  
 Point3Ds[i] = Point3D(x, y,z);  
 }  
   
 std::cout << "The Point3Ds in the convex hull are: " << std::endl;  
 float timer1 = std::clock();  
   
 auto hull = convexHull(Point3Ds, len);  
   
 std::cout << std::endl << "time: " << (float)(clock() - timer1) / CLOCKS\_PER\_SEC << std::endl;  
 for (int i = 0; i < hull.size(); ++i) {  
 std::cout << "( " << hull[i].a.x << " , " << hull[i].a.y << " , " << hull[i].a.z << " ) ";  
 std::cout << "( " << hull[i].b.x << " , " << hull[i].b.y << " , " << hull[i].b.z << " ) ";  
 std::cout << "( " << hull[i].c.x << " , " << hull[i].c.y << " , " << hull[i].c.z << " )" << std::endl;  
   
 }  
   
 delete[]Point3Ds;  
 return 0;  
}