Работа призера заключительного этапа

командной инженерной олимпиады школьников

Олимпиада Национальной технологической инициативы

Профиль «БОЛЬШИЕ ДАННЫЕ И МАШИННОЕ ОБУЧЕНИЕ»

Себякин Андрей Сергеевич

Класс: 11

Город: Москва

Школа: ГОБУ «Физтех-лицей»

Регион: Московская область

Уникальный номер участника: 228666

Команда на заключительном этапе: wafcatn

Результаты заключительного этапа:

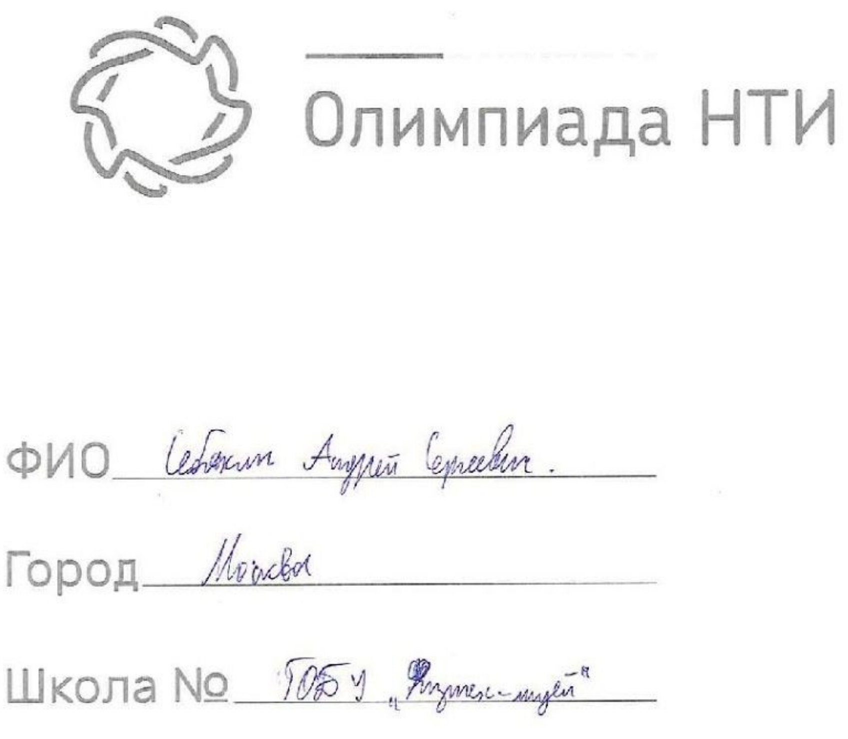
Индивидуальный тур				Командный тур			Итого:
Математ. нормиров	Коэф. 40%	Информат. нормиров..	Коэф. 40%	Задача 1 норм.	Задача 2 норм.	Коэф. команд. задач 60%	
0,7	27.4	0.2	6.7	0.9	0.9	106.9	141

Расшифровка индивидуальной части:

Индивидуальный тур													
Математика							Информатика						
2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	Итого	Итого нормиров.	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	Итого	Итого норм.
0	5	4	11	4	24	0.7	0	5	0	0	0	5	0.2

Индивидуальная часть

Персональный лист участника с номером 228666:



Олимпиада НТИ

ФИО Себякин Андрей Сергеевич.

Город Москва

Школа № ГОБУ „[illegible]-лицей"

Математика

Командная инженерная олимпиада «Олимпиада НТИ»

Направление БД

Предмет математика

Номер участника 228666

24 МА

Стратификация классов.

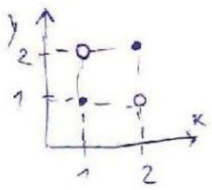
1. Да, может. Необходимо выбрать 10 людей, которые питаются нормально. НИЧЬЯ —
2. Да, необходимо ~~[illegible]~~ выбрать 10 переедающих человек. +
3. Да, необходимо взять 5 человек, питающихся нормально и 5 человек, которые переедают. +
4. 1) Да, необходимо выбрать 30 нормально питающихся людей. НИЧЬЯ —

   2) Да, необходимо выбрать 10 переедающих, 10 переедающих и 10 нормально питающихся. НИЧЬЯ —

   3) Да, необходимо выбрать 10 переедающих и 20 нормально питающихся +

Профиль компактности и метод ближайшего соседа

1) Да



+

2, 3) Нет. —

Разделяющие поверхности и ГМТ.

1) $A(5;8)$; $O(-2;1)$. $\rho(A;O)=\sqrt{49+49}=$ ~~$\sqrt{98}$~~ $\sqrt{49\cdot 2}=7\sqrt{2}$

$\rho_{иск}=\rho(A;O)-1=7\sqrt{2}-1$. +

Ответ: $7\sqrt{2}-1$.

2) ~~[illegible]~~ $A(5;6)$

$\rho(A;(x;y))=|y-1|$ $\Rightarrow$ $\rho(A;(x;y))=y-1$.

$(x-5)^2+(y-6)^2=(y-1)^2$; $y\geq 1$.

$x^2-10x+25+y^2-12y+36-(y^2-2y+1)=0$.

$x^2-10x+60-10y=0$; $y=\frac{x^2-10x+60}{10}$ ~~$=\frac{x^2-10x+25+35}{10}$~~

Ответ: ~~[illegible]~~ все точки, заданные ур-ем ~~$\frac{x^2-10x+60}{10}$~~ $y=\frac{x^2-10x+60}{10}$. +

3) ~~[illegible]~~ Радиус второй окружности любой, меньший суммы расстояния между центрами окружностей и радиуса первой окружности (10). 3.1) = Да, зависит. —

Ответ: $R_2<R_1+\rho(O_1;O_2)$

Командная инженерная олимпиада «Олимпиада НТИ»

Направление БД

Предмет математика

Номер участника 228666

Baseline.

1. $1-\frac{300}{600}\cdot\frac{299}{599}=1-\underset{\uparrow\ \frac{1}{2}}{\frac{1}{3}}\cdot\frac{299}{599}=1-\frac{299}{1797}=\frac{1498}{1797}.$ $+$

2. $1-\left(\frac{1}{6}\cdot\frac{300}{600}\cdot\frac{299}{599}+\frac{1}{6}\cdot\frac{200}{600}\cdot\frac{199}{599}+\frac{1}{6}\cdot\frac{100}{600}\cdot\frac{99}{599}+\frac{1}{6}\cdot\frac{100}{600}\cdot\frac{200}{599}\right.$ $\mp$

Информатика

Задача 2.3.1 Кусочно постоянные функции (подзадачи 1, 2, 3)

Решение не предоставлено

Задача 2.3.2 Метод ближайшего соседа (подзадачи 1, 2, 3)

Подзадача 1:

```
print(int(input())//2)
```

Результат: 5 баллов

Подзадача 2-3:
Решение не предоставлено

Задача 2.3.3 Монотонные классификаторы (подзадачи 1, 2, 3)

Решение не предоставлено

Задача 2.3.4 Различные методы (подзадачи 1, 2, 3)

Решение не предоставлено

Задача 2.3.5 Классификация и кластеризация (подзадачи 1, 2, 3)

Подзадача 1:
Решение не предоставлено

Подзадача 2:

```
def sqdist(a, b):
        return (a[0] - b[0])**2 + (a[1] - b[1])**2

n = int(input())
img = []
for i in range(n):
        img.append(input())

def check_square():
        topleft, topright, downleft, downright = (-1, -1), (-1, -1), (-1, -1), (-1, -1)
        for i in range(n):
        for j in range(n):
        if img[i][j] == '#':
                if topleft[0] == -1:
                topleft = (i, j)
                else:
                if j >= downright[1]:
                downright = (i, j)
        for i in range(n - 1, -1, -1):
        for j in range(n):
        if img[i][j] == '#':
                if downleft[0] == -1:
```

```
                downleft = (i, j)
                else:
                if j >= topright[1]:
                topright = (i, j)
        #print(topleft, topright, downleft, downright)
        if (sqdist(topleft, topright) - sqdist(downleft, downright) <= 1) and \
        (sqdist(downleft, topleft) - sqdist(downright, topright) <= 1) and \
        (sqdist(downright, downright) - sqdist(downleft, topleft) <= 1):
        return True
        return False


def check_circle():
        topleft, topright, downleft, downright = (-1, -1), (-1, -1), (-1, -1), (-1, -1)
        for i in range(n):
        if topleft[0] != -1:
        break
        for j in range(n):
        if img[i][j] == '#':
                if topleft[0] == -1:
                topleft = (i, j)
                else:
                topright = (i, j)
        for i in range(n - 1, -1, -1):
        if downleft[0] != -1:
        break
        for j in range(n):
        if img[i][j] == '#':
                if downleft[0] == -1:
                downleft = (i, j)
                else:
                downright = (i, j)
        #print(topleft, topright, downleft, downright)
        if (sqdist(topleft, topright) - sqdist(downleft, downright) == 0) and \
        (sqdist(downleft, topleft) - sqdist(downright, topright) == 0):
        return True
        return False


if check_square():
        print(2)
elif check_circle():
        print(1)
else:
        print(3)
```

Результат: 0 баллов

Подзадача 3:
Решение не предоставлено

Командная часть

Результаты были получены в рамках выступления команды: wafcatn

Личный состав команды:

- Себякин Андрей Сергеевич
- Иванов Евгений Андреевич

Фото команды:

Рис.1 – Фото участников команды

Решения командного этапа:

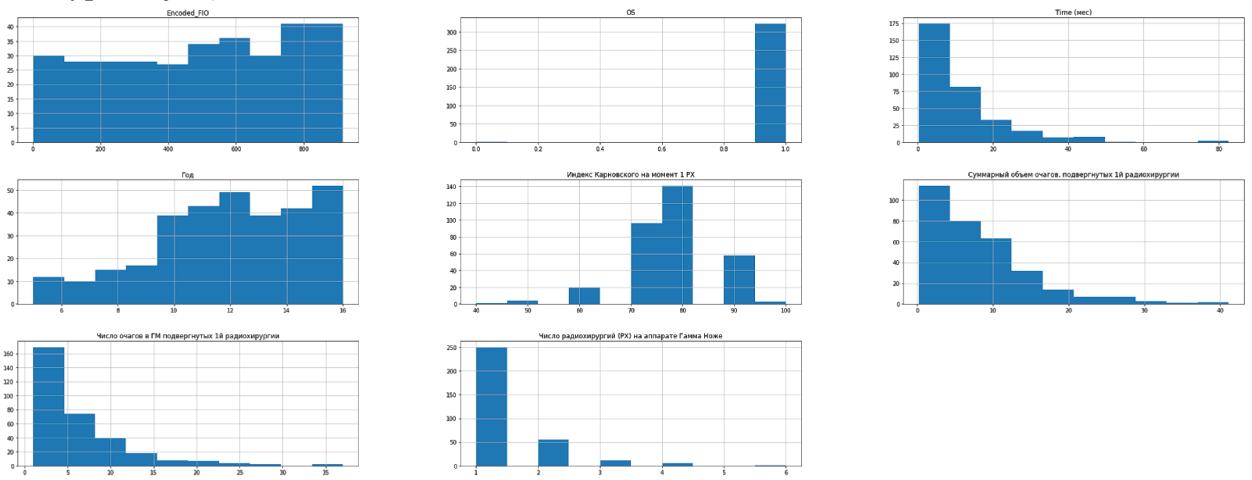
Код программы на языке Pyton, обеспечивающий решение задачи №1

```
import os
folder_path = 'C:\\Users\\erqups\\NTI_BD_2018_1_task'
if not os.getcwd() == folder_path:
    os.chdir(folder_path)
import pandas as pd
import numpy as np
pd.options.display.max_columns=100
from datetime import datetime
import sklearn
import random
from fancyimpute import KNN


def save_predicted_data(predicted_data):
    fname = "y_test_data-{}.txt".format(random.randint(1, 10000))
    with open(fname, "w") as y_test:
        for row in predicted_data:
            y_test.write(str(row))
            y_test.write("\n")
    print("saved as ", fname)
    return True


def normalize_date(date):
    date = date.split(".")
    prefix = "20" if int(date[-1]) <= 18 else "19"
    date[-1] = prefix + date[-1]
    return datetime.strptime(".".join(date), "%d.%m.%Y")


def normalize_dataset(train_dead):
    useless_columns = ["Encoded_FIO", "Time (мес)", "OS", "DEATH", "Дата последнего
наблюдения ", "HID"]
    for us_col in useless_columns:
        try:
            del train_dead[us_col]
        except:pass

    train_dead["Дата рождения"] = [normalize_date(date) for date in train_dead["Дата
рождения"]]
    train_dead["Дата рождения"] = train_dead["Дата рождения"].astype("datetime64[ns]")

    proccessed_column = "Пол"
    treatment_type = {
        "М": 0,
        "м": 0,
        "ж": 1,
        "Ж": 1,
    }
    train_dead[proccessed_column] = train_dead[proccessed_column].replace(treatment_type)

    proccessed_column = "Лекарственное лечение на момент проведения
радиохирургии/гтпофракционирования"
    train_dead[proccessed_column] = train_dead[proccessed_column].fillna(0)
    treatment_type = {
        "Без лечения": 0,
        "Химиотерапия": 1,
        "Таргетная терапия": 2,
    }
    train_dead[proccessed_column] = train_dead[proccessed_column].replace(treatment_type)

    proccessed_column = "Онкологический диагноз"
    treatment_type = {
        "НМРЛ": 0,
        "РМЖ": 1,
        "РП": 2,
        "КРР": 3,
        "Меланома": 4,
    }
    train_dead[proccessed_column] = train_dead[proccessed_column].replace(treatment_type)

    proccessed_column = "Активирующие мутации"
    treatment_type = {
        "нет": 0,
        "есть": 1
    }
    train_dead[proccessed_column] = train_dead[proccessed_column].replace(treatment_type)


    proccessed_column = "Экстракраниальные метастазы на момент проведения 
радиохирургии / гиплфракционирования"
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    processed_list = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj != "есть":
            processed_list += [0]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list

    proccessed_column = "Локальный рецидив после радиохирургии / 
гипофракционирования"
    processed_list = []
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "нет":
            processed_list += [0]
        elif obj == "nan":
            processed_list += [np.nan]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Дистантные метастазы после радиохирургии / 
гипофракционирования"
    processed_list = []
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if (obj == "нет"):
            processed_list += [0]
        elif obj == "nan":
            processed_list += [np.nan]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Интракраниальная прогрессия (локальные рецидивы +дистантные 
метастазы)"
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    process_LR = []
    process_DM = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "ЛР":
            process_LR += [1]
            process_DM += [0]
        elif obj == "ЛР+ДМ":
            process_LR += [1]
            process_DM += [1]
        elif obj == "ДМ":
            process_LR += [0]
            process_DM += [1]
        else:
            process_LR += [0]
            process_DM += [0]

    del train_dead[proccessed_column]

    train_dead["ЛР"] = process_LR
    train_dead["ДМ"] = process_DM
    train_dead["ЛР"] = train_dead["ЛР"].astype("int64")
    train_dead["ДМ"] = train_dead["ДМ"].astype("int64")

    proccessed_column = "Лечение ИК рецидива после после 1-ой радиохирургии / 
гипофракционирования"
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    processed_list = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "РХ":
            processed_list += [1]
        elif obj == "Оп":
            processed_list += [2]
        elif obj == "ОВГМ":
            processed_list += [3]
        elif obj == 'химиотерапия':
            processed_list += [4]
        elif obj == 'таргетная терапия':
            processed_list += [5]
        elif obj == 'nan':
            processed_list += [np.nan]
        else:
            processed_list += [0]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Индекс Карновского на момент 1 РХ"
    new_column = "Индекс Карновского - 2"
    processed_ind = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj >= 80:
            processed_ind += [1]
        else:
            processed_ind += [0]

    train_dead[new_column] = processed_ind

    proccessed_column = "Суммарный объем очагов, подвергнутых 1й радиохирургии"
    new_column = "Приемлемый объем МГМ"
    processed_ind = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj <= 5:
            processed_ind += [1]
        else:
            processed_ind += [0]
    train_dead[new_column] = processed_ind

    proccessed_column = "Дата постановки онкологического диагноза/ начала первичного
лечения"
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    processed_days = []

    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == 'nan':
            processed_days += [np.nan]
        else:
            processed_days += [(normalize_date(obj) - train_dead.iloc[itr]["Дата рождения"]).days]

    train_dead[proccessed_column] = processed_days
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")



    #train_dead[proccessed_column] =
train_dead[proccessed_column].fillna(train_dead[proccessed_column].median())

    proccessed_column = "Дата 1й РХ на ГН"
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    processed_days = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        processed_days += [(normalize_date(obj) - train_dead.iloc[itr]["Дата рождения"]).days - 
train_dead.iloc[itr]["Дата постановки онкологического диагноза/ начала первичного 
лечения"]]
    train_dead[proccessed_column] = processed_days
    #train_dead[proccessed_column] = 
train_dead[proccessed_column].fillna(train_dead[proccessed_column].median())


    proccessed_column = "Первичный очаг (дата удаления)"
    processed_list = []
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "не удален":
            processed_list += [0]
        elif obj == "nan":
            processed_list += [np.nan]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Дата развития МГМ "
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    processed_days = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "nan":
            processed_days += [np.nan]
        else:
            processed_days += [(normalize_date(obj) - train_dead.iloc[itr]["Дата рождения"]).days - 
train_dead.iloc[itr]["Дата постановки онкологического диагноза/ начала первичного 
лечения"]]
    train_dead[proccessed_column] = processed_days
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")
    #train_dead[proccessed_column] = 
train_dead[proccessed_column].fillna(train_dead[proccessed_column].median())

    proccessed_column = "Дата проведения ОВГМ"
    processed_list = []
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "нет":
            processed_list += [0]
        elif obj == 'nan':
            processed_list += [np.nan]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Дата операции на ГМ"
    processed_list = []
    train_dead[proccessed_column] = train_dead[proccessed_column].astype(str)
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj == "нет":
            processed_list += [0]
        elif obj == 'nan':
            processed_list += [np.nan]
        else:
            processed_list += [1]
    train_dead[proccessed_column] = processed_list
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")
    #proccessed_column = "Индекс Карновского на момент 1 РХ"
    #train_dead[proccessed_column] = train_dead[proccessed_column] // 10

    proccessed_column = "Объем макс очага, подвергнутого 1й радиохирургии"

    train_dead[proccessed_column] = train_dead[proccessed_column].apply(lambda x:
x.replace(",", "."))
    train_dead[proccessed_column] = train_dead[proccessed_column].astype("float64")

    proccessed_column = "Дата постановки онкологического диагноза/ начала первичного
лечения"
    new_column = "Возраст в годах при первичном лечении"
    processed_ind = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]

        processed_ind += [round(obj / 365.5)]

    train_dead[new_column] = processed_ind
    del train_dead[proccessed_column]

    proccessed_column = "Возраст в годах при первичном лечении"
    new_column = "Старше 70"
    processed_ind = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj >= 70:
            processed_ind += [1]
        else:
            processed_ind += [0]
    train_dead[new_column] = processed_ind

    proccessed_column = "Число очагов в ГМ подвергнутых 1й радиохирургии"
    new_column = "Приемлимое кол-во очагов"
    processed_ind = []
    for itr in range(len(train_dead[proccessed_column])):
        obj = train_dead.iloc[itr][proccessed_column]
        if obj >= 3:
            processed_ind += [1]
        else:
            processed_ind += [0]
    train_dead[new_column] = processed_ind



    return train_dead

def remove_nan_rows(X_in, y_in):
    important_columns = ["Пол",
                "Дата рождения",
                "Онкологический диагноз",
                "Дата постановки онкологического диагноза/ начала первичного лечения",
                "Первичный очаг (дата удаления)",
                "Дата развития МГМ ",
                "Активирующие мутации",
                "Дата проведения ОВГМ",
                "Дата операции на ГМ",
                "Число радиохирургий (РХ) на аппарате Гамма Ноже",
                "Дата 1й РХ на ГН",
                "Индекс Карновского на момент 1 РХ",
                "Число очагов в ГМ подвергнутых 1й радиохирургии",
                "Суммарный объем очагов, подвергнутых 1й радиохирургии",
                "Экстракраниальные метастазы на момент проведения радиохирургии /
гиплфракционирования",
                "Лекарственное лечение на момент проведения
радиохирургии/гтпофракционирования",
                "Локальный рецидив после радиохирургии / гипофракционирования",
                "Дистантные метастазы после радиохирургии / гипофракционирования",
                "Интракраниальная прогрессия (локальные рецидивы +дистантные
метастазы)",
                "Лечение ИК рецидива после после 1-ой радиохирургии /
гипофракционирования"]
    row_list = []
    X_in = pd.concat((X_in, y_in), axis = 1)
    X_importants = X_in[important_columns]
    for itr in range(X_importants.shape[0]):
        processed_row = X_importants.iloc[itr]
        nan_objects_count = len([bl for bl in processed_row.isnull() if bl])
        if nan_objects_count < 10:
            row_list += [X_in.iloc[itr]]
    X_result = pd.DataFrame(data=row_list, columns=X_in.columns)
    y_result = X_result[0]
    del X_result[0]
    return (X_result, y_result)

# Max score: 27.31
using_columns = [
    "Онкологический диагноз",
    #"Первичный очаг (дата удаления)",
    #"Дата развития МГМ ",
    #"Активирующие мутации",
    #"Дата проведения ОВГМ",
    #"Дата операции на ГМ",
    "Число радиохирургий (РХ) на аппарате Гамма Ноже",
    "Дата 1й РХ на ГН",
    "Индекс Карновского на момент 1 РХ",
    #"Число очагов в ГМ подвергнутых 1й радиохирургии",
    #"Суммарный объем очагов, подвергнутых 1й радиохирургии"
    "Объем макс очага, подвергнутого 1й радиохирургии",
    "Экстракраниальные метастазы на момент проведения радиохирургии /
гиплфракционирования",
    "Лекарственное лечение на момент проведения
радиохирургии/гтпофракционирования",
    "Локальный рецидив после радиохирургии / гипофракционирования",
    "Дистантные метастазы после радиохирургии / гипофракционирования",
    "Лечение ИК рецидива после после 1-ой радиохирургии / гипофракционирования",
    "ЛР",
    "ДМ",
    "Индекс Карновского - 2",
    "Приемлемый объем МГМ",
    #"Возраст в годах при первичном лечении",
    #"Старше 70",
    "Приемлимое кол-во очагов",
]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion
of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will
be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
y_in = pd.read_csv("y_train_new.csv", encoding="windows-1251", header=None)
train_dead = pd.read_csv("X_train_new.csv", encoding="windows-1251")
X_test = pd.read_csv("X_test_new.csv", encoding="windows-1251")
X_in_cleared, y_in = remove_nan_rows(train_dead, y_in)
%matplotlib inline
import seaborn as sns
X_in_cleared.hist(figsize=(40,15))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A091240>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A0AC518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B19E9BE80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000022B19E79358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A1A59E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A1A5A20>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A06F5C0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A10A198>,
    <matplotlib.axes._subplots.AxesSubplot object at 0x0000022B1A274828>]],
   dtype=object)
```



```
X_normalized = normalize_dataset(X_in_cleared)[using_columns]
X_test_normalized = normalize_dataset(X_test)[using_columns]
X_normalized = KNN(k=5).complete(X_normalized)
X_test_normalized = KNN(k=5).complete(X_test_normalized)
X_normalized = pd.DataFrame(data=X_normalized, columns=using_columns)
X_test_normalized = pd.DataFrame(data=X_test_normalized, columns=using_columns)
logarithm_features = [
    #"Суммарный объем очагов, подвергнутых 1й радиохирургии",
   # "Число очагов в ГМ подвергнутых 1й радиохирургии",
    "Объем макс очага, подвергнутого 1й радиохирургии"
]
for feature in logarithm_features:
    log_feature = np.log(X_normalized[feature])
    X_normalized[feature] = log_feature

    log_feature = np.log(X_test_normalized[feature])
    X_test_normalized[feature] = log_feature

#X_in_cleared = X_in_cleared[using_columns]
#X_test_normalized = X_test_normalized[using_columns]

old_used_columns = [val for val in using_columns]

for col in old_used_columns:
    #for pol in range(2, 5):
    #    col_name = col + " ^{}".format(pol)
    #    X_normalized[col_name] = (X_normalized[col] + 1) ** pol
    #    X_test_normalized[col_name] = (X_test_normalized[col] + 1) ** pol
        #X_alive[col_name] = (X_alive[col] + 1) ** pol

    col_name = " cos  {}".format(col)
    X_normalized[col_name] = np.cos(X_normalized[col] + 1)
    X_test_normalized[col_name] = np.cos(X_test_normalized[col] + 1)
    #X_alive[col_name] = np.cos(X_alive[col] + 1)

    col_name =  " cos + 1 {}".format(col)
    X_normalized[col_name] = np.cos(X_normalized[col] + 1)
    X_test_normalized[col_name] = np.cos(X_test_normalized[col] + 1)
    #X_alive[col_name] = np.cos(X_alive[col] + 1)
X_test_normalized.shape, X_normalized.shape

Imputing row 1/323 with 1 missing, elapsed time: 0.020
Imputing row 101/323 with 2 missing, elapsed time: 0.023
Imputing row 201/323 with 3 missing, elapsed time: 0.025
Imputing row 301/323 with 3 missing, elapsed time: 0.028
Imputing row 1/144 with 1 missing, elapsed time: 0.007
Imputing row 101/144 with 0 missing, elapsed time: 0.009

((144, 45), (323, 45))
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
#def add_pca(x):
#    pca = PCA(n_components=3, svd_solver='full', whiten=True, tol=0.0, iterated_power='auto',
random_state=42)
#    pca_feats = pd.DataFrame(pca.fit_transform(x)*100)
#    return pd.concat((x, pca_feats), axis=1)


def add_tsne(x):
    components = 3 # comp. = 5, init=pca,angle=5,iter=2000 - 27.7; comp. = 3,
init=pca,angle=5,iter=2000 - 28.6;
    tsne = TSNE(n_components=components, perplexity=30.0, early_exaggeration=12.0,
learning_rate=200.0,
         n_iter=10000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='cosine',
init='pca',
         verbose=5, random_state=53, method='exact')
    tsne_feats = pd.DataFrame(tsne.fit_transform(x))
    for i in range(components):
        t = []
        for el in tsne_feats[i]:
            if el > 0.5:
                t.append(np.log(el)*10)
            elif el < 0:
                t.append(-np.log(-el)*10)
            else:
                t.append(0)
        tsne_feats[i] = np.array(t)
    tsne_feats.columns = ['tsne'+str(i) for i in range(components)]
    #print(tsne_feats)
    return pd.concat((x, tsne_feats), axis=1)

[t-SNE] Computing pairwise distances...
[t-SNE] Computed conditional probabilities for sample 323 / 323
[t-SNE] Mean sigma: 0.006989
[t-SNE] Iteration 50: error = 80.9573439, gradient norm = 0.0021937 (50 iterations in 0.320s)
[t-SNE] Iteration 100: error = 87.5508493, gradient norm = 0.0014130 (50 iterations in 0.298s)
[t-SNE] Iteration 150: error = 77.3866373, gradient norm = 0.0007216 (50 iterations in 0.284s)
```

```
[t-SNE] Iteration 200: error = 84.7027979, gradient norm = 0.0010764 (50 iterations in 0.282s)
[t-SNE] Iteration 250: error = 82.2918945, gradient norm = 0.0006882 (50 iterations in 0.300s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 82.291895
[t-SNE] Iteration 300: error = 3.5545681, gradient norm = 0.0002276 (50 iterations in 0.299s)
[t-SNE] Iteration 350: error = 3.2052849, gradient norm = 0.0000999 (50 iterations in 0.295s)
[t-SNE] Iteration 400: error = 3.0123652, gradient norm = 0.0000614 (50 iterations in 0.277s)
[t-SNE] Iteration 450: error = 2.8804078, gradient norm = 0.0000436 (50 iterations in 0.277s)
[t-SNE] Iteration 500: error = 2.7809328, gradient norm = 0.0000335 (50 iterations in 0.329s)
[t-SNE] Iteration 550: error = 2.7015027, gradient norm = 0.0000270 (50 iterations in 0.328s)
[t-SNE] Iteration 600: error = 2.6357740, gradient norm = 0.0000224 (50 iterations in 0.267s)
[t-SNE] Iteration 650: error = 2.5799278, gradient norm = 0.0000191 (50 iterations in 0.284s)
[t-SNE] Iteration 700: error = 2.5314208, gradient norm = 0.0000166 (50 iterations in 0.266s)
[t-SNE] Iteration 750: error = 2.4886453, gradient norm = 0.0000146 (50 iterations in 0.267s)
[t-SNE] Iteration 800: error = 2.4506014, gradient norm = 0.0000130 (50 iterations in 0.282s)
[t-SNE] Iteration 850: error = 2.4165904, gradient norm = 0.0000117 (50 iterations in 0.267s)
[t-SNE] Iteration 900: error = 2.3860761, gradient norm = 0.0000105 (50 iterations in 0.267s)
[t-SNE] Iteration 950: error = 2.3585877, gradient norm = 0.0000096 (50 iterations in 0.284s)
[t-SNE] Iteration 1000: error = 2.3336952, gradient norm = 0.0000088 (50 iterations in 0.267s)
[t-SNE] Iteration 1050: error = 2.3110045, gradient norm = 0.0000081 (50 iterations in 0.266s)
[t-SNE] Iteration 1100: error = 2.2901717, gradient norm = 0.0000075 (50 iterations in 0.284s)
[t-SNE] Iteration 1150: error = 2.2708934, gradient norm = 0.0000070 (50 iterations in 0.267s)
[t-SNE] Iteration 1200: error = 2.2528847, gradient norm = 0.0000066 (50 iterations in 0.283s)
[t-SNE] Iteration 1250: error = 2.2359129, gradient norm = 0.0000062 (50 iterations in 0.279s)
[t-SNE] Iteration 1300: error = 2.2198081, gradient norm = 0.0000058 (50 iterations in 0.270s)
[t-SNE] Iteration 1350: error = 2.2044641, gradient norm = 0.0000055 (50 iterations in 0.266s)
[t-SNE] Iteration 1400: error = 2.1898219, gradient norm = 0.0000052 (50 iterations in 0.299s)
[t-SNE] Iteration 1450: error = 2.1758582, gradient norm = 0.0000050 (50 iterations in 0.275s)
[t-SNE] Iteration 1500: error = 2.1625657, gradient norm = 0.0000047 (50 iterations in 0.347s)
[t-SNE] Iteration 1550: error = 2.1499278, gradient norm = 0.0000045 (50 iterations in 0.303s)
[t-SNE] Iteration 1600: error = 2.1379144, gradient norm = 0.0000043 (50 iterations in 0.301s)
[t-SNE] Iteration 1650: error = 2.1264825, gradient norm = 0.0000041 (50 iterations in 0.298s)
[t-SNE] Iteration 1700: error = 2.1155867, gradient norm = 0.0000039 (50 iterations in 0.310s)
[t-SNE] Iteration 1750: error = 2.1051852, gradient norm = 0.0000037 (50 iterations in 0.294s)
[t-SNE] Iteration 1800: error = 2.0952398, gradient norm = 0.0000036 (50 iterations in 0.295s)
[t-SNE] Iteration 1850: error = 2.0857090, gradient norm = 0.0000034 (50 iterations in 0.308s)
[t-SNE] Iteration 1900: error = 2.0765483, gradient norm = 0.0000033 (50 iterations in 0.306s)
[t-SNE] Iteration 1950: error = 2.0677223, gradient norm = 0.0000032 (50 iterations in 0.314s)
[t-SNE] Iteration 2000: error = 2.0592048, gradient norm = 0.0000031 (50 iterations in 0.301s)
[t-SNE] Iteration 2050: error = 2.0509762, gradient norm = 0.0000030 (50 iterations in 0.283s)
[t-SNE] Iteration 2100: error = 2.0430222, gradient norm = 0.0000029 (50 iterations in 0.295s)
[t-SNE] Iteration 2150: error = 2.0353287, gradient norm = 0.0000028 (50 iterations in 0.294s)
[t-SNE] Iteration 2200: error = 2.0278836, gradient norm = 0.0000027 (50 iterations in 0.277s)
[t-SNE] Iteration 2250: error = 2.0206720, gradient norm = 0.0000026 (50 iterations in 0.266s)
[t-SNE] Iteration 2300: error = 2.0136777, gradient norm = 0.0000025 (50 iterations in 0.283s)
[t-SNE] Iteration 2350: error = 2.0068840, gradient norm = 0.0000024 (50 iterations in 0.289s)
[t-SNE] Iteration 2400: error = 2.0002781, gradient norm = 0.0000024 (50 iterations in 0.278s)
[t-SNE] Iteration 2450: error = 1.9938478, gradient norm = 0.0000023 (50 iterations in 0.286s)
[t-SNE] Iteration 2500: error = 1.9875807, gradient norm = 0.0000022 (50 iterations in 0.265s)
[t-SNE] Iteration 2550: error = 1.9814659, gradient norm = 0.0000022 (50 iterations in 0.283s)
[t-SNE] Iteration 2600: error = 1.9754931, gradient norm = 0.0000021 (50 iterations in 0.267s)
[t-SNE] Iteration 2650: error = 1.9696546, gradient norm = 0.0000021 (50 iterations in 0.266s)
```

```
[t-SNE] Iteration 2700: error = 1.9639437, gradient norm = 0.0000020 (50 iterations in 0.276s)
[t-SNE] Iteration 2750: error = 1.9583536, gradient norm = 0.0000020 (50 iterations in 0.284s)
[t-SNE] Iteration 2800: error = 1.9528786, gradient norm = 0.0000019 (50 iterations in 0.256s)
[t-SNE] Iteration 2850: error = 1.9475139, gradient norm = 0.0000019 (50 iterations in 0.285s)
[t-SNE] Iteration 2900: error = 1.9422558, gradient norm = 0.0000018 (50 iterations in 0.265s)
[t-SNE] Iteration 2950: error = 1.9371013, gradient norm = 0.0000018 (50 iterations in 0.281s)
[t-SNE] Iteration 3000: error = 1.9320461, gradient norm = 0.0000018 (50 iterations in 0.269s)
[t-SNE] Iteration 3050: error = 1.9270868, gradient norm = 0.0000017 (50 iterations in 0.276s)
[t-SNE] Iteration 3100: error = 1.9222188, gradient norm = 0.0000017 (50 iterations in 0.272s)
[t-SNE] Iteration 3150: error = 1.9174382, gradient norm = 0.0000017 (50 iterations in 0.257s)
[t-SNE] Iteration 3200: error = 1.9127414, gradient norm = 0.0000016 (50 iterations in 0.277s)
[t-SNE] Iteration 3250: error = 1.9081255, gradient norm = 0.0000016 (50 iterations in 0.277s)
[t-SNE] Iteration 3300: error = 1.9035907, gradient norm = 0.0000016 (50 iterations in 0.257s)
[t-SNE] Iteration 3350: error = 1.8991373, gradient norm = 0.0000015 (50 iterations in 0.267s)
[t-SNE] Iteration 3400: error = 1.8947654, gradient norm = 0.0000015 (50 iterations in 0.266s)
[t-SNE] Iteration 3450: error = 1.8904752, gradient norm = 0.0000015 (50 iterations in 0.283s)
[t-SNE] Iteration 3500: error = 1.8862673, gradient norm = 0.0000014 (50 iterations in 0.268s)
[t-SNE] Iteration 3550: error = 1.8821418, gradient norm = 0.0000014 (50 iterations in 0.280s)
[t-SNE] Iteration 3600: error = 1.8781003, gradient norm = 0.0000014 (50 iterations in 0.271s)
[t-SNE] Iteration 3650: error = 1.8741438, gradient norm = 0.0000014 (50 iterations in 0.283s)
[t-SNE] Iteration 3700: error = 1.8702713, gradient norm = 0.0000013 (50 iterations in 0.282s)
[t-SNE] Iteration 3750: error = 1.8664814, gradient norm = 0.0000013 (50 iterations in 0.281s)
[t-SNE] Iteration 3800: error = 1.8627718, gradient norm = 0.0000013 (50 iterations in 0.270s)
[t-SNE] Iteration 3850: error = 1.8591389, gradient norm = 0.0000013 (50 iterations in 0.282s)
[t-SNE] Iteration 3900: error = 1.8555788, gradient norm = 0.0000012 (50 iterations in 0.267s)
[t-SNE] Iteration 3950: error = 1.8520877, gradient norm = 0.0000012 (50 iterations in 0.283s)
[t-SNE] Iteration 4000: error = 1.8486617, gradient norm = 0.0000012 (50 iterations in 0.268s)
[t-SNE] Iteration 4050: error = 1.8452978, gradient norm = 0.0000012 (50 iterations in 0.281s)
[t-SNE] Iteration 4100: error = 1.8419918, gradient norm = 0.0000011 (50 iterations in 0.283s)
[t-SNE] Iteration 4150: error = 1.8387396, gradient norm = 0.0000011 (50 iterations in 0.270s)
[t-SNE] Iteration 4200: error = 1.8355385, gradient norm = 0.0000011 (50 iterations in 0.295s)
[t-SNE] Iteration 4250: error = 1.8323862, gradient norm = 0.0000011 (50 iterations in 0.292s)
[t-SNE] Iteration 4300: error = 1.8292797, gradient norm = 0.0000011 (50 iterations in 0.278s)
[t-SNE] Iteration 4350: error = 1.8262160, gradient norm = 0.0000011 (50 iterations in 0.283s)
[t-SNE] Iteration 4400: error = 1.8231934, gradient norm = 0.0000010 (50 iterations in 0.273s)
[t-SNE] Iteration 4450: error = 1.8202095, gradient norm = 0.0000010 (50 iterations in 0.293s)
[t-SNE] Iteration 4500: error = 1.8172616, gradient norm = 0.0000010 (50 iterations in 0.283s)
[t-SNE] Iteration 4550: error = 1.8143469, gradient norm = 0.0000010 (50 iterations in 0.268s)
[t-SNE] Iteration 4600: error = 1.8114623, gradient norm = 0.0000010 (50 iterations in 0.299s)
[t-SNE] Iteration 4650: error = 1.8086051, gradient norm = 0.0000010 (50 iterations in 0.283s)
[t-SNE] Iteration 4700: error = 1.8057727, gradient norm = 0.0000010 (50 iterations in 0.298s)
[t-SNE] Iteration 4750: error = 1.8029631, gradient norm = 0.0000009 (50 iterations in 0.320s)
[t-SNE] Iteration 4800: error = 1.8001740, gradient norm = 0.0000009 (50 iterations in 0.250s)
[t-SNE] Iteration 4850: error = 1.7974035, gradient norm = 0.0000009 (50 iterations in 0.317s)
[t-SNE] Iteration 4900: error = 1.7946502, gradient norm = 0.0000009 (50 iterations in 0.304s)
[t-SNE] Iteration 4950: error = 1.7919131, gradient norm = 0.0000009 (50 iterations in 0.262s)
[t-SNE] Iteration 5000: error = 1.7891917, gradient norm = 0.0000009 (50 iterations in 0.289s)
[t-SNE] Iteration 5050: error = 1.7864853, gradient norm = 0.0000009 (50 iterations in 0.277s)
[t-SNE] Iteration 5100: error = 1.7837945, gradient norm = 0.0000009 (50 iterations in 0.302s)
[t-SNE] Iteration 5150: error = 1.7811196, gradient norm = 0.0000009 (50 iterations in 0.282s)
[t-SNE] Iteration 5200: error = 1.7784604, gradient norm = 0.0000009 (50 iterations in 0.267s)
```

```
[t-SNE] Iteration 5250: error = 1.7758161, gradient norm = 0.0000008 (50 iterations in 0.296s)
[t-SNE] Iteration 5300: error = 1.7731868, gradient norm = 0.0000008 (50 iterations in 0.286s)
[t-SNE] Iteration 5350: error = 1.7705719, gradient norm = 0.0000008 (50 iterations in 0.284s)
[t-SNE] Iteration 5400: error = 1.7679717, gradient norm = 0.0000008 (50 iterations in 0.280s)
[t-SNE] Iteration 5450: error = 1.7653852, gradient norm = 0.0000008 (50 iterations in 0.289s)
[t-SNE] Iteration 5500: error = 1.7628123, gradient norm = 0.0000008 (50 iterations in 0.265s)
[t-SNE] Iteration 5550: error = 1.7602521, gradient norm = 0.0000008 (50 iterations in 0.283s)
[t-SNE] Iteration 5600: error = 1.7577035, gradient norm = 0.0000008 (50 iterations in 0.262s)
[t-SNE] Iteration 5650: error = 1.7551665, gradient norm = 0.0000008 (50 iterations in 0.286s)
[t-SNE] Iteration 5700: error = 1.7526409, gradient norm = 0.0000008 (50 iterations in 0.252s)
[t-SNE] Iteration 5750: error = 1.7501261, gradient norm = 0.0000008 (50 iterations in 0.286s)
[t-SNE] Iteration 5800: error = 1.7476220, gradient norm = 0.0000008 (50 iterations in 0.279s)
[t-SNE] Iteration 5850: error = 1.7451280, gradient norm = 0.0000007 (50 iterations in 0.294s)
[t-SNE] Iteration 5900: error = 1.7426443, gradient norm = 0.0000007 (50 iterations in 0.273s)
[t-SNE] Iteration 5950: error = 1.7401708, gradient norm = 0.0000007 (50 iterations in 0.293s)
[t-SNE] Iteration 6000: error = 1.7377078, gradient norm = 0.0000007 (50 iterations in 0.302s)
[t-SNE] Iteration 6050: error = 1.7352552, gradient norm = 0.0000007 (50 iterations in 0.282s)
[t-SNE] Iteration 6100: error = 1.7328133, gradient norm = 0.0000007 (50 iterations in 0.297s)
[t-SNE] Iteration 6150: error = 1.7303828, gradient norm = 0.0000007 (50 iterations in 0.349s)
[t-SNE] Iteration 6200: error = 1.7279633, gradient norm = 0.0000007 (50 iterations in 0.298s)
[t-SNE] Iteration 6250: error = 1.7255551, gradient norm = 0.0000007 (50 iterations in 0.359s)
[t-SNE] Iteration 6300: error = 1.7231582, gradient norm = 0.0000007 (50 iterations in 0.313s)
[t-SNE] Iteration 6350: error = 1.7207725, gradient norm = 0.0000007 (50 iterations in 0.306s)
[t-SNE] Iteration 6400: error = 1.7183987, gradient norm = 0.0000007 (50 iterations in 0.331s)
[t-SNE] Iteration 6450: error = 1.7160366, gradient norm = 0.0000007 (50 iterations in 0.332s)
[t-SNE] Iteration 6500: error = 1.7136864, gradient norm = 0.0000007 (50 iterations in 0.301s)
[t-SNE] Iteration 6550: error = 1.7113476, gradient norm = 0.0000007 (50 iterations in 0.348s)
[t-SNE] Iteration 6600: error = 1.7090205, gradient norm = 0.0000007 (50 iterations in 0.322s)
[t-SNE] Iteration 6650: error = 1.7067053, gradient norm = 0.0000006 (50 iterations in 0.284s)
[t-SNE] Iteration 6700: error = 1.7044015, gradient norm = 0.0000006 (50 iterations in 0.282s)
[t-SNE] Iteration 6750: error = 1.7021090, gradient norm = 0.0000006 (50 iterations in 0.283s)
[t-SNE] Iteration 6800: error = 1.6998273, gradient norm = 0.0000006 (50 iterations in 0.279s)
[t-SNE] Iteration 6850: error = 1.6975555, gradient norm = 0.0000006 (50 iterations in 0.272s)
[t-SNE] Iteration 6900: error = 1.6952932, gradient norm = 0.0000006 (50 iterations in 0.284s)
[t-SNE] Iteration 6950: error = 1.6930399, gradient norm = 0.0000006 (50 iterations in 0.251s)
[t-SNE] Iteration 7000: error = 1.6907944, gradient norm = 0.0000006 (50 iterations in 0.266s)
[t-SNE] Iteration 7050: error = 1.6885558, gradient norm = 0.0000006 (50 iterations in 0.342s)
[t-SNE] Iteration 7100: error = 1.6863230, gradient norm = 0.0000006 (50 iterations in 0.274s)
[t-SNE] Iteration 7150: error = 1.6840958, gradient norm = 0.0000006 (50 iterations in 0.336s)
[t-SNE] Iteration 7200: error = 1.6818739, gradient norm = 0.0000006 (50 iterations in 0.281s)
[t-SNE] Iteration 7250: error = 1.6796570, gradient norm = 0.0000006 (50 iterations in 0.305s)
[t-SNE] Iteration 7300: error = 1.6774449, gradient norm = 0.0000006 (50 iterations in 0.285s)
[t-SNE] Iteration 7350: error = 1.6752380, gradient norm = 0.0000006 (50 iterations in 0.270s)
[t-SNE] Iteration 7400: error = 1.6730367, gradient norm = 0.0000006 (50 iterations in 0.288s)
[t-SNE] Iteration 7450: error = 1.6708414, gradient norm = 0.0000006 (50 iterations in 0.302s)
[t-SNE] Iteration 7500: error = 1.6686527, gradient norm = 0.0000006 (50 iterations in 0.320s)
[t-SNE] Iteration 7550: error = 1.6664713, gradient norm = 0.0000006 (50 iterations in 0.293s)
[t-SNE] Iteration 7600: error = 1.6642975, gradient norm = 0.0000006 (50 iterations in 0.312s)
[t-SNE] Iteration 7650: error = 1.6621321, gradient norm = 0.0000006 (50 iterations in 0.297s)
[t-SNE] Iteration 7700: error = 1.6599759, gradient norm = 0.0000006 (50 iterations in 0.296s)
[t-SNE] Iteration 7750: error = 1.6578294, gradient norm = 0.0000006 (50 iterations in 0.328s)
```

```
[t-SNE] Iteration 7800: error = 1.6556929, gradient norm = 0.0000006 (50 iterations in 0.306s)
[t-SNE] Iteration 7850: error = 1.6535672, gradient norm = 0.0000005 (50 iterations in 0.292s)
[t-SNE] Iteration 7900: error = 1.6514526, gradient norm = 0.0000005 (50 iterations in 0.297s)
[t-SNE] Iteration 7950: error = 1.6493494, gradient norm = 0.0000005 (50 iterations in 0.298s)
[t-SNE] Iteration 8000: error = 1.6472576, gradient norm = 0.0000005 (50 iterations in 0.305s)
[t-SNE] Iteration 8050: error = 1.6451775, gradient norm = 0.0000005 (50 iterations in 0.292s)
[t-SNE] Iteration 8100: error = 1.6431091, gradient norm = 0.0000005 (50 iterations in 0.311s)
[t-SNE] Iteration 8150: error = 1.6410520, gradient norm = 0.0000005 (50 iterations in 0.270s)
[t-SNE] Iteration 8200: error = 1.6390054, gradient norm = 0.0000005 (50 iterations in 0.274s)
[t-SNE] Iteration 8250: error = 1.6369691, gradient norm = 0.0000005 (50 iterations in 0.277s)
[t-SNE] Iteration 8300: error = 1.6349428, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8350: error = 1.6329262, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8400: error = 1.6309190, gradient norm = 0.0000005 (50 iterations in 0.298s)
[t-SNE] Iteration 8450: error = 1.6289212, gradient norm = 0.0000005 (50 iterations in 0.274s)
[t-SNE] Iteration 8500: error = 1.6269327, gradient norm = 0.0000005 (50 iterations in 0.261s)
[t-SNE] Iteration 8550: error = 1.6249530, gradient norm = 0.0000005 (50 iterations in 0.269s)
[t-SNE] Iteration 8600: error = 1.6229825, gradient norm = 0.0000005 (50 iterations in 0.264s)
[t-SNE] Iteration 8650: error = 1.6210206, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 8700: error = 1.6190677, gradient norm = 0.0000005 (50 iterations in 0.282s)
[t-SNE] Iteration 8750: error = 1.6171234, gradient norm = 0.0000005 (50 iterations in 0.251s)
[t-SNE] Iteration 8800: error = 1.6151878, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 8850: error = 1.6132611, gradient norm = 0.0000005 (50 iterations in 0.287s)
[t-SNE] Iteration 8900: error = 1.6113432, gradient norm = 0.0000005 (50 iterations in 0.278s)
[t-SNE] Iteration 8950: error = 1.6094338, gradient norm = 0.0000005 (50 iterations in 0.283s)
[t-SNE] Iteration 9000: error = 1.6075328, gradient norm = 0.0000005 (50 iterations in 0.273s)
[t-SNE] Iteration 9050: error = 1.6056398, gradient norm = 0.0000005 (50 iterations in 0.262s)
[t-SNE] Iteration 9100: error = 1.6037546, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 9150: error = 1.6018766, gradient norm = 0.0000005 (50 iterations in 0.283s)
[t-SNE] Iteration 9200: error = 1.6000056, gradient norm = 0.0000005 (50 iterations in 0.283s)
[t-SNE] Iteration 9250: error = 1.5981407, gradient norm = 0.0000005 (50 iterations in 0.259s)
[t-SNE] Iteration 9300: error = 1.5962816, gradient norm = 0.0000004 (50 iterations in 0.275s)
[t-SNE] Iteration 9350: error = 1.5944274, gradient norm = 0.0000004 (50 iterations in 0.280s)
[t-SNE] Iteration 9400: error = 1.5925779, gradient norm = 0.0000004 (50 iterations in 0.275s)
[t-SNE] Iteration 9450: error = 1.5907326, gradient norm = 0.0000004 (50 iterations in 0.262s)
[t-SNE] Iteration 9500: error = 1.5888914, gradient norm = 0.0000004 (50 iterations in 0.268s)
[t-SNE] Iteration 9550: error = 1.5870543, gradient norm = 0.0000004 (50 iterations in 0.276s)
[t-SNE] Iteration 9600: error = 1.5852207, gradient norm = 0.0000004 (50 iterations in 0.297s)
[t-SNE] Iteration 9650: error = 1.5833907, gradient norm = 0.0000004 (50 iterations in 0.291s)
[t-SNE] Iteration 9700: error = 1.5815636, gradient norm = 0.0000004 (50 iterations in 0.291s)
[t-SNE] Iteration 9750: error = 1.5797397, gradient norm = 0.0000004 (50 iterations in 0.287s)
[t-SNE] Iteration 9800: error = 1.5779186, gradient norm = 0.0000004 (50 iterations in 0.303s)
[t-SNE] Iteration 9850: error = 1.5761003, gradient norm = 0.0000004 (50 iterations in 0.293s)
[t-SNE] Iteration 9900: error = 1.5742849, gradient norm = 0.0000004 (50 iterations in 0.289s)
[t-SNE] Iteration 9950: error = 1.5724728, gradient norm = 0.0000004 (50 iterations in 0.307s)
[t-SNE] Iteration 10000: error = 1.5706641, gradient norm = 0.0000004 (50 iterations in 0.278s)
[t-SNE] Error after 10000 iterations: 1.570664
[t-SNE] Computing pairwise distances...
[t-SNE] Computed conditional probabilities for sample 144 / 144
[t-SNE] Mean sigma: 0.015076
[t-SNE] Iteration 50: error = 73.9180367, gradient norm = 0.0008442 (50 iterations in 0.101s)
[t-SNE] Iteration 100: error = 98.1209027, gradient norm = 0.0043776 (50 iterations in 0.085s)
```

```
[t-SNE] Iteration 150: error = 83.5131493, gradient norm = 0.0007952 (50 iterations in 0.084s)
[t-SNE] Iteration 200: error = 79.3412466, gradient norm = 0.0002510 (50 iterations in 0.095s)
[t-SNE] Iteration 250: error = 80.7634780, gradient norm = 0.0002377 (50 iterations in 0.084s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.763478
[t-SNE] Iteration 300: error = 3.7102009, gradient norm = 0.0001848 (50 iterations in 0.086s)
[t-SNE] Iteration 350: error = 3.3257975, gradient norm = 0.0000845 (50 iterations in 0.086s)
[t-SNE] Iteration 400: error = 3.1010671, gradient norm = 0.0000523 (50 iterations in 0.090s)
[t-SNE] Iteration 450: error = 2.9431313, gradient norm = 0.0000371 (50 iterations in 0.089s)
[t-SNE] Iteration 500: error = 2.8216488, gradient norm = 0.0000284 (50 iterations in 0.082s)
[t-SNE] Iteration 550: error = 2.7235138, gradient norm = 0.0000229 (50 iterations in 0.083s)
[t-SNE] Iteration 600: error = 2.6415964, gradient norm = 0.0000191 (50 iterations in 0.086s)
[t-SNE] Iteration 650: error = 2.5719505, gradient norm = 0.0000162 (50 iterations in 0.083s)
[t-SNE] Iteration 700: error = 2.5115168, gradient norm = 0.0000141 (50 iterations in 0.082s)
[t-SNE] Iteration 750: error = 2.4578677, gradient norm = 0.0000125 (50 iterations in 0.086s)
[t-SNE] Iteration 800: error = 2.4095150, gradient norm = 0.0000111 (50 iterations in 0.083s)
[t-SNE] Iteration 850: error = 2.3656817, gradient norm = 0.0000100 (50 iterations in 0.081s)
[t-SNE] Iteration 900: error = 2.3259192, gradient norm = 0.0000090 (50 iterations in 0.090s)
[t-SNE] Iteration 950: error = 2.2898365, gradient norm = 0.0000082 (50 iterations in 0.085s)
[t-SNE] Iteration 1000: error = 2.2570103, gradient norm = 0.0000074 (50 iterations in 0.086s)
[t-SNE] Iteration 1050: error = 2.2270209, gradient norm = 0.0000068 (50 iterations in 0.084s)
[t-SNE] Iteration 1100: error = 2.1994864, gradient norm = 0.0000063 (50 iterations in 0.082s)
[t-SNE] Iteration 1150: error = 2.1741029, gradient norm = 0.0000058 (50 iterations in 0.080s)
[t-SNE] Iteration 1200: error = 2.1506336, gradient norm = 0.0000054 (50 iterations in 0.083s)
[t-SNE] Iteration 1250: error = 2.1288825, gradient norm = 0.0000050 (50 iterations in 0.082s)
[t-SNE] Iteration 1300: error = 2.1086757, gradient norm = 0.0000047 (50 iterations in 0.084s)
[t-SNE] Iteration 1350: error = 2.0898688, gradient norm = 0.0000044 (50 iterations in 0.083s)
[t-SNE] Iteration 1400: error = 2.0723320, gradient norm = 0.0000042 (50 iterations in 0.077s)
[t-SNE] Iteration 1450: error = 2.0559469, gradient norm = 0.0000039 (50 iterations in 0.076s)
[t-SNE] Iteration 1500: error = 2.0405988, gradient norm = 0.0000037 (50 iterations in 0.080s)
[t-SNE] Iteration 1550: error = 2.0261759, gradient norm = 0.0000035 (50 iterations in 0.085s)
[t-SNE] Iteration 1600: error = 2.0125899, gradient norm = 0.0000034 (50 iterations in 0.083s)
[t-SNE] Iteration 1650: error = 1.9997650, gradient norm = 0.0000032 (50 iterations in 0.086s)
[t-SNE] Iteration 1700: error = 1.9876305, gradient norm = 0.0000030 (50 iterations in 0.077s)
[t-SNE] Iteration 1750: error = 1.9761257, gradient norm = 0.0000029 (50 iterations in 0.077s)
[t-SNE] Iteration 1800: error = 1.9651985, gradient norm = 0.0000028 (50 iterations in 0.085s)
[t-SNE] Iteration 1850: error = 1.9548000, gradient norm = 0.0000027 (50 iterations in 0.092s)
[t-SNE] Iteration 1900: error = 1.9448852, gradient norm = 0.0000026 (50 iterations in 0.097s)
[t-SNE] Iteration 1950: error = 1.9354106, gradient norm = 0.0000025 (50 iterations in 0.134s)
[t-SNE] Iteration 2000: error = 1.9263353, gradient norm = 0.0000024 (50 iterations in 0.096s)
[t-SNE] Iteration 2050: error = 1.9176250, gradient norm = 0.0000023 (50 iterations in 0.081s)
[t-SNE] Iteration 2100: error = 1.9092485, gradient norm = 0.0000022 (50 iterations in 0.087s)
[t-SNE] Iteration 2150: error = 1.9011738, gradient norm = 0.0000022 (50 iterations in 0.081s)
[t-SNE] Iteration 2200: error = 1.8933718, gradient norm = 0.0000021 (50 iterations in 0.084s)
[t-SNE] Iteration 2250: error = 1.8858170, gradient norm = 0.0000020 (50 iterations in 0.087s)
[t-SNE] Iteration 2300: error = 1.8784851, gradient norm = 0.0000020 (50 iterations in 0.080s)
[t-SNE] Iteration 2350: error = 1.8713536, gradient norm = 0.0000019 (50 iterations in 0.079s)
[t-SNE] Iteration 2400: error = 1.8644018, gradient norm = 0.0000019 (50 iterations in 0.152s)
[t-SNE] Iteration 2450: error = 1.8576119, gradient norm = 0.0000018 (50 iterations in 0.128s)
[t-SNE] Iteration 2500: error = 1.8509694, gradient norm = 0.0000018 (50 iterations in 0.149s)
[t-SNE] Iteration 2550: error = 1.8444623, gradient norm = 0.0000018 (50 iterations in 0.111s)
[t-SNE] Iteration 2600: error = 1.8380792, gradient norm = 0.0000017 (50 iterations in 0.101s)
```

```
[t-SNE] Iteration 2650: error = 1.8318114, gradient norm = 0.0000017 (50 iterations in 0.173s)
[t-SNE] Iteration 2700: error = 1.8256515, gradient norm = 0.0000016 (50 iterations in 0.103s)
[t-SNE] Iteration 2750: error = 1.8195933, gradient norm = 0.0000016 (50 iterations in 0.087s)
[t-SNE] Iteration 2800: error = 1.8136321, gradient norm = 0.0000016 (50 iterations in 0.076s)
[t-SNE] Iteration 2850: error = 1.8077651, gradient norm = 0.0000015 (50 iterations in 0.081s)
[t-SNE] Iteration 2900: error = 1.8019905, gradient norm = 0.0000015 (50 iterations in 0.080s)
[t-SNE] Iteration 2950: error = 1.7963076, gradient norm = 0.0000015 (50 iterations in 0.088s)
[t-SNE] Iteration 3000: error = 1.7907164, gradient norm = 0.0000014 (50 iterations in 0.122s)
[t-SNE] Iteration 3050: error = 1.7852176, gradient norm = 0.0000014 (50 iterations in 0.075s)
[t-SNE] Iteration 3100: error = 1.7798135, gradient norm = 0.0000014 (50 iterations in 0.094s)
[t-SNE] Iteration 3150: error = 1.7745055, gradient norm = 0.0000014 (50 iterations in 0.095s)
[t-SNE] Iteration 3200: error = 1.7692953, gradient norm = 0.0000013 (50 iterations in 0.065s)
[t-SNE] Iteration 3250: error = 1.7641851, gradient norm = 0.0000013 (50 iterations in 0.078s)
[t-SNE] Iteration 3300: error = 1.7591759, gradient norm = 0.0000013 (50 iterations in 0.092s)
[t-SNE] Iteration 3350: error = 1.7542685, gradient norm = 0.0000012 (50 iterations in 0.078s)
[t-SNE] Iteration 3400: error = 1.7494636, gradient norm = 0.0000012 (50 iterations in 0.078s)
[t-SNE] Iteration 3450: error = 1.7447612, gradient norm = 0.0000012 (50 iterations in 0.086s)
[t-SNE] Iteration 3500: error = 1.7401600, gradient norm = 0.0000012 (50 iterations in 0.064s)
[t-SNE] Iteration 3550: error = 1.7356583, gradient norm = 0.0000011 (50 iterations in 0.092s)
[t-SNE] Iteration 3600: error = 1.7312536, gradient norm = 0.0000011 (50 iterations in 0.083s)
[t-SNE] Iteration 3650: error = 1.7269428, gradient norm = 0.0000011 (50 iterations in 0.080s)
[t-SNE] Iteration 3700: error = 1.7227217, gradient norm = 0.0000011 (50 iterations in 0.078s)
[t-SNE] Iteration 3750: error = 1.7185868, gradient norm = 0.0000011 (50 iterations in 0.077s)
[t-SNE] Iteration 3800: error = 1.7145348, gradient norm = 0.0000010 (50 iterations in 0.089s)
[t-SNE] Iteration 3850: error = 1.7105615, gradient norm = 0.0000010 (50 iterations in 0.067s)
[t-SNE] Iteration 3900: error = 1.7066627, gradient norm = 0.0000010 (50 iterations in 0.078s)
[t-SNE] Iteration 3950: error = 1.7028343, gradient norm = 0.0000010 (50 iterations in 0.106s)
[t-SNE] Iteration 4000: error = 1.6990725, gradient norm = 0.0000010 (50 iterations in 0.096s)
[t-SNE] Iteration 4050: error = 1.6953734, gradient norm = 0.0000010 (50 iterations in 0.101s)
[t-SNE] Iteration 4100: error = 1.6917336, gradient norm = 0.0000009 (50 iterations in 0.089s)
[t-SNE] Iteration 4150: error = 1.6881500, gradient norm = 0.0000009 (50 iterations in 0.086s)
[t-SNE] Iteration 4200: error = 1.6846197, gradient norm = 0.0000009 (50 iterations in 0.077s)
[t-SNE] Iteration 4250: error = 1.6811398, gradient norm = 0.0000009 (50 iterations in 0.089s)
[t-SNE] Iteration 4300: error = 1.6777073, gradient norm = 0.0000009 (50 iterations in 0.100s)
[t-SNE] Iteration 4350: error = 1.6743200, gradient norm = 0.0000009 (50 iterations in 0.084s)
[t-SNE] Iteration 4400: error = 1.6709757, gradient norm = 0.0000009 (50 iterations in 0.083s)
[t-SNE] Iteration 4450: error = 1.6676722, gradient norm = 0.0000008 (50 iterations in 0.113s)
[t-SNE] Iteration 4500: error = 1.6644080, gradient norm = 0.0000008 (50 iterations in 0.091s)
[t-SNE] Iteration 4550: error = 1.6611816, gradient norm = 0.0000008 (50 iterations in 0.084s)
[t-SNE] Iteration 4600: error = 1.6579911, gradient norm = 0.0000008 (50 iterations in 0.088s)
[t-SNE] Iteration 4650: error = 1.6548355, gradient norm = 0.0000008 (50 iterations in 0.080s)
[t-SNE] Iteration 4700: error = 1.6517133, gradient norm = 0.0000008 (50 iterations in 0.080s)
[t-SNE] Iteration 4750: error = 1.6486233, gradient norm = 0.0000008 (50 iterations in 0.087s)
[t-SNE] Iteration 4800: error = 1.6455647, gradient norm = 0.0000008 (50 iterations in 0.088s)
[t-SNE] Iteration 4850: error = 1.6425363, gradient norm = 0.0000008 (50 iterations in 0.084s)
[t-SNE] Iteration 4900: error = 1.6395372, gradient norm = 0.0000007 (50 iterations in 0.086s)
[t-SNE] Iteration 4950: error = 1.6365667, gradient norm = 0.0000007 (50 iterations in 0.105s)
[t-SNE] Iteration 5000: error = 1.6336235, gradient norm = 0.0000007 (50 iterations in 0.091s)
[t-SNE] Iteration 5050: error = 1.6307066, gradient norm = 0.0000007 (50 iterations in 0.095s)
[t-SNE] Iteration 5100: error = 1.6278145, gradient norm = 0.0000007 (50 iterations in 0.094s)
[t-SNE] Iteration 5150: error = 1.6249461, gradient norm = 0.0000007 (50 iterations in 0.094s)
```

```
[t-SNE] Iteration 5200: error = 1.6221004, gradient norm = 0.0000007 (50 iterations in 0.098s)
[t-SNE] Iteration 5250: error = 1.6192762, gradient norm = 0.0000007 (50 iterations in 0.085s)
[t-SNE] Iteration 5300: error = 1.6164723, gradient norm = 0.0000007 (50 iterations in 0.092s)
[t-SNE] Iteration 5350: error = 1.6136874, gradient norm = 0.0000007 (50 iterations in 0.089s)
[t-SNE] Iteration 5400: error = 1.6109208, gradient norm = 0.0000007 (50 iterations in 0.079s)
[t-SNE] Iteration 5450: error = 1.6081718, gradient norm = 0.0000007 (50 iterations in 0.066s)
[t-SNE] Iteration 5500: error = 1.6054397, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5550: error = 1.6027234, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5600: error = 1.6000223, gradient norm = 0.0000006 (50 iterations in 0.085s)
[t-SNE] Iteration 5650: error = 1.5973361, gradient norm = 0.0000006 (50 iterations in 0.091s)
[t-SNE] Iteration 5700: error = 1.5946641, gradient norm = 0.0000006 (50 iterations in 0.088s)
[t-SNE] Iteration 5750: error = 1.5920063, gradient norm = 0.0000006 (50 iterations in 0.086s)
[t-SNE] Iteration 5800: error = 1.5893615, gradient norm = 0.0000006 (50 iterations in 0.086s)
[t-SNE] Iteration 5850: error = 1.5867290, gradient norm = 0.0000006 (50 iterations in 0.085s)
[t-SNE] Iteration 5900: error = 1.5841087, gradient norm = 0.0000006 (50 iterations in 0.095s)
[t-SNE] Iteration 5950: error = 1.5815003, gradient norm = 0.0000006 (50 iterations in 0.096s)
[t-SNE] Iteration 6000: error = 1.5789040, gradient norm = 0.0000006 (50 iterations in 0.089s)
[t-SNE] Iteration 6050: error = 1.5763191, gradient norm = 0.0000006 (50 iterations in 0.083s)
[t-SNE] Iteration 6100: error = 1.5737461, gradient norm = 0.0000006 (50 iterations in 0.083s)
[t-SNE] Iteration 6150: error = 1.5711845, gradient norm = 0.0000006 (50 iterations in 0.095s)
[t-SNE] Iteration 6200: error = 1.5686343, gradient norm = 0.0000006 (50 iterations in 0.096s)
[t-SNE] Iteration 6250: error = 1.5660957, gradient norm = 0.0000006 (50 iterations in 0.120s)
[t-SNE] Iteration 6300: error = 1.5635681, gradient norm = 0.0000005 (50 iterations in 0.104s)
[t-SNE] Iteration 6350: error = 1.5610517, gradient norm = 0.0000005 (50 iterations in 0.089s)
[t-SNE] Iteration 6400: error = 1.5585465, gradient norm = 0.0000005 (50 iterations in 0.083s)
[t-SNE] Iteration 6450: error = 1.5560521, gradient norm = 0.0000005 (50 iterations in 0.082s)
[t-SNE] Iteration 6500: error = 1.5535692, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 6550: error = 1.5510979, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 6600: error = 1.5486380, gradient norm = 0.0000005 (50 iterations in 0.113s)
[t-SNE] Iteration 6650: error = 1.5461899, gradient norm = 0.0000005 (50 iterations in 0.099s)
[t-SNE] Iteration 6700: error = 1.5437536, gradient norm = 0.0000005 (50 iterations in 0.092s)
[t-SNE] Iteration 6750: error = 1.5413293, gradient norm = 0.0000005 (50 iterations in 0.088s)
[t-SNE] Iteration 6800: error = 1.5389169, gradient norm = 0.0000005 (50 iterations in 0.070s)
[t-SNE] Iteration 6850: error = 1.5365167, gradient norm = 0.0000005 (50 iterations in 0.083s)
[t-SNE] Iteration 6900: error = 1.5341290, gradient norm = 0.0000005 (50 iterations in 0.074s)
[t-SNE] Iteration 6950: error = 1.5317542, gradient norm = 0.0000005 (50 iterations in 0.079s)
[t-SNE] Iteration 7000: error = 1.5293925, gradient norm = 0.0000005 (50 iterations in 0.063s)
[t-SNE] Iteration 7050: error = 1.5270445, gradient norm = 0.0000005 (50 iterations in 0.099s)
[t-SNE] Iteration 7100: error = 1.5247107, gradient norm = 0.0000005 (50 iterations in 0.064s)
[t-SNE] Iteration 7150: error = 1.5223914, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 7200: error = 1.5200869, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 7250: error = 1.5177977, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 7300: error = 1.5155242, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 7350: error = 1.5132669, gradient norm = 0.0000005 (50 iterations in 0.091s)
[t-SNE] Iteration 7400: error = 1.5110263, gradient norm = 0.0000004 (50 iterations in 0.081s)
[t-SNE] Iteration 7450: error = 1.5088022, gradient norm = 0.0000004 (50 iterations in 0.091s)
[t-SNE] Iteration 7500: error = 1.5065948, gradient norm = 0.0000004 (50 iterations in 0.094s)
[t-SNE] Iteration 7550: error = 1.5044044, gradient norm = 0.0000004 (50 iterations in 0.090s)
[t-SNE] Iteration 7600: error = 1.5022305, gradient norm = 0.0000004 (50 iterations in 0.092s)
[t-SNE] Iteration 7650: error = 1.5000733, gradient norm = 0.0000004 (50 iterations in 0.092s)
[t-SNE] Iteration 7700: error = 1.4979324, gradient norm = 0.0000004 (50 iterations in 0.087s)
```

```
[t-SNE] Iteration 7750: error = 1.4958085, gradient norm = 0.0000004 (50 iterations in 0.083s)
[t-SNE] Iteration 7800: error = 1.4937007, gradient norm = 0.0000004 (50 iterations in 0.083s)
[t-SNE] Iteration 7850: error = 1.4916093, gradient norm = 0.0000004 (50 iterations in 0.085s)
[t-SNE] Iteration 7900: error = 1.4895339, gradient norm = 0.0000004 (50 iterations in 0.081s)
[t-SNE] Iteration 7950: error = 1.4874741, gradient norm = 0.0000004 (50 iterations in 0.069s)
[t-SNE] Iteration 8000: error = 1.4854298, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8050: error = 1.4834007, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8100: error = 1.4813871, gradient norm = 0.0000004 (50 iterations in 0.108s)
[t-SNE] Iteration 8150: error = 1.4793890, gradient norm = 0.0000004 (50 iterations in 0.092s)
[t-SNE] Iteration 8200: error = 1.4774060, gradient norm = 0.0000004 (50 iterations in 0.091s)
[t-SNE] Iteration 8250: error = 1.4754383, gradient norm = 0.0000004 (50 iterations in 0.088s)
[t-SNE] Iteration 8300: error = 1.4734857, gradient norm = 0.0000004 (50 iterations in 0.085s)
[t-SNE] Iteration 8350: error = 1.4715484, gradient norm = 0.0000004 (50 iterations in 0.083s)
[t-SNE] Iteration 8400: error = 1.4696263, gradient norm = 0.0000004 (50 iterations in 0.084s)
[t-SNE] Iteration 8450: error = 1.4677192, gradient norm = 0.0000004 (50 iterations in 0.089s)
[t-SNE] Iteration 8500: error = 1.4658267, gradient norm = 0.0000004 (50 iterations in 0.082s)
[t-SNE] Iteration 8550: error = 1.4639493, gradient norm = 0.0000004 (50 iterations in 0.080s)
[t-SNE] Iteration 8600: error = 1.4620869, gradient norm = 0.0000004 (50 iterations in 0.076s)
[t-SNE] Iteration 8650: error = 1.4602396, gradient norm = 0.0000004 (50 iterations in 0.068s)
[t-SNE] Iteration 8700: error = 1.4584071, gradient norm = 0.0000004 (50 iterations in 0.082s)
[t-SNE] Iteration 8750: error = 1.4565896, gradient norm = 0.0000004 (50 iterations in 0.084s)
[t-SNE] Iteration 8800: error = 1.4547868, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8850: error = 1.4529982, gradient norm = 0.0000003 (50 iterations in 0.084s)
[t-SNE] Iteration 8900: error = 1.4512241, gradient norm = 0.0000003 (50 iterations in 0.087s)
[t-SNE] Iteration 8950: error = 1.4494643, gradient norm = 0.0000003 (50 iterations in 0.084s)
[t-SNE] Iteration 9000: error = 1.4477186, gradient norm = 0.0000003 (50 iterations in 0.088s)
[t-SNE] Iteration 9050: error = 1.4459869, gradient norm = 0.0000003 (50 iterations in 0.091s)
[t-SNE] Iteration 9100: error = 1.4442693, gradient norm = 0.0000003 (50 iterations in 0.092s)
[t-SNE] Iteration 9150: error = 1.4425653, gradient norm = 0.0000003 (50 iterations in 0.094s)
[t-SNE] Iteration 9200: error = 1.4408744, gradient norm = 0.0000003 (50 iterations in 0.088s)
[t-SNE] Iteration 9250: error = 1.4391965, gradient norm = 0.0000003 (50 iterations in 0.079s)
[t-SNE] Iteration 9300: error = 1.4375316, gradient norm = 0.0000003 (50 iterations in 0.070s)
[t-SNE] Iteration 9350: error = 1.4358794, gradient norm = 0.0000003 (50 iterations in 0.087s)
[t-SNE] Iteration 9400: error = 1.4342394, gradient norm = 0.0000003 (50 iterations in 0.077s)
[t-SNE] Iteration 9450: error = 1.4326111, gradient norm = 0.0000003 (50 iterations in 0.063s)
[t-SNE] Iteration 9500: error = 1.4309945, gradient norm = 0.0000003 (50 iterations in 0.092s)
[t-SNE] Iteration 9550: error = 1.4293893, gradient norm = 0.0000003 (50 iterations in 0.075s)
[t-SNE] Iteration 9600: error = 1.4277947, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9650: error = 1.4262108, gradient norm = 0.0000003 (50 iterations in 0.090s)
[t-SNE] Iteration 9700: error = 1.4246371, gradient norm = 0.0000003 (50 iterations in 0.083s)
[t-SNE] Iteration 9750: error = 1.4230734, gradient norm = 0.0000003 (50 iterations in 0.076s)
[t-SNE] Iteration 9800: error = 1.4215191, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9850: error = 1.4199742, gradient norm = 0.0000003 (50 iterations in 0.061s)
[t-SNE] Iteration 9900: error = 1.4184386, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9950: error = 1.4169117, gradient norm = 0.0000003 (50 iterations in 0.091s)
[t-SNE] Iteration 10000: error = 1.4153936, gradient norm = 0.0000003 (50 iterations in 0.083s)
[t-SNE] Error after 10000 iterations: 1.415394
```

```
from sklearn.linear_model import ElasticNet, Lars, LassoLars, TheilSenRegressor,
RANSACRegressor, HuberRegressor, LogisticRegression
from sklearn.kernel_ridge import KernelRidge################30.45
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
rf_1 =  sklearn.ensemble.ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth =
3, random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
kernel_ridge_1 = KernelRidge(alpha=1)
kernel_ridge_2 = KernelRidge(alpha=0.5)
kernel_ridge_3 = KernelRidge(alpha=0.3)
kernel_ridge_4 = KernelRidge(alpha=0.1)
lasso_lars1 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-4)
lasso_lars2 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-3)
lasso_lars3 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-2)
lasso_lars4 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-1)
rf_1 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_10 =  ExtraTreesRegressor(n_estimators=3000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_2 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_7 =  ExtraTreesRegressor(n_estimators=6000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_8 =  ExtraTreesRegressor(n_estimators=7000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_9 =  ExtraTreesRegressor(n_estimators=8000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_3 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_4 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 4,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_5 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_11 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_12 =  ExtraTreesRegressor(n_estimators=8000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_6 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 6,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_13 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 8,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
from sklearn import linear_model
bay_ridge = linear_model.BayesianRidge()

X_normalized = add_tsne(X_normalized)
X_test_normalized = add_tsne(X_test_normalized)
```

```
[t-SNE] Computing pairwise distances...
[t-SNE] Computed conditional probabilities for sample 323 / 323
[t-SNE] Mean sigma: 0.029105
[t-SNE] Iteration 50: error = 80.2198603, gradient norm = 0.0026766 (50 iterations in 0.311s)
[t-SNE] Iteration 100: error = 84.2202060, gradient norm = 0.0009344 (50 iterations in 0.279s)
[t-SNE] Iteration 150: error = 75.3743545, gradient norm = 0.0002442 (50 iterations in 0.291s)
[t-SNE] Iteration 200: error = 75.9501466, gradient norm = 0.0002201 (50 iterations in 0.275s)
[t-SNE] Iteration 250: error = 83.0664873, gradient norm = 0.0002701 (50 iterations in 0.276s)
```

```
[t-SNE] KL divergence after 250 iterations with early exaggeration: 83.066487
[t-SNE] Iteration 300: error = 3.4821722, gradient norm = 0.0002003 (50 iterations in 0.297s)
[t-SNE] Iteration 350: error = 3.1803957, gradient norm = 0.0000771 (50 iterations in 0.285s)
[t-SNE] Iteration 400: error = 3.0432021, gradient norm = 0.0000445 (50 iterations in 0.258s)
[t-SNE] Iteration 450: error = 2.9543605, gradient norm = 0.0000309 (50 iterations in 0.276s)
[t-SNE] Iteration 500: error = 2.8857173, gradient norm = 0.0000238 (50 iterations in 0.280s)
[t-SNE] Iteration 550: error = 2.8272161, gradient norm = 0.0000197 (50 iterations in 0.272s)
[t-SNE] Iteration 600: error = 2.7745773, gradient norm = 0.0000169 (50 iterations in 0.282s)
[t-SNE] Iteration 650: error = 2.7259919, gradient norm = 0.0000148 (50 iterations in 0.284s)
[t-SNE] Iteration 700: error = 2.6807315, gradient norm = 0.0000132 (50 iterations in 0.267s)
[t-SNE] Iteration 750: error = 2.6384556, gradient norm = 0.0000118 (50 iterations in 0.266s)
[t-SNE] Iteration 800: error = 2.5989526, gradient norm = 0.0000107 (50 iterations in 0.268s)
[t-SNE] Iteration 850: error = 2.5620168, gradient norm = 0.0000097 (50 iterations in 0.267s)
[t-SNE] Iteration 900: error = 2.5274312, gradient norm = 0.0000089 (50 iterations in 0.270s)
[t-SNE] Iteration 950: error = 2.4949919, gradient norm = 0.0000082 (50 iterations in 0.263s)
[t-SNE] Iteration 1000: error = 2.4645063, gradient norm = 0.0000076 (50 iterations in 0.283s)
[t-SNE] Iteration 1050: error = 2.4357637, gradient norm = 0.0000071 (50 iterations in 0.251s)
[t-SNE] Iteration 1100: error = 2.4085458, gradient norm = 0.0000066 (50 iterations in 0.268s)
[t-SNE] Iteration 1150: error = 2.3826450, gradient norm = 0.0000062 (50 iterations in 0.265s)
[t-SNE] Iteration 1200: error = 2.3578917, gradient norm = 0.0000059 (50 iterations in 0.282s)
[t-SNE] Iteration 1250: error = 2.3341476, gradient norm = 0.0000056 (50 iterations in 0.267s)
[t-SNE] Iteration 1300: error = 2.3113137, gradient norm = 0.0000053 (50 iterations in 0.268s)
[t-SNE] Iteration 1350: error = 2.2893211, gradient norm = 0.0000051 (50 iterations in 0.314s)
[t-SNE] Iteration 1400: error = 2.2681130, gradient norm = 0.0000048 (50 iterations in 0.336s)
[t-SNE] Iteration 1450: error = 2.2476488, gradient norm = 0.0000046 (50 iterations in 0.330s)
[t-SNE] Iteration 1500: error = 2.2279004, gradient norm = 0.0000044 (50 iterations in 0.304s)
[t-SNE] Iteration 1550: error = 2.2088502, gradient norm = 0.0000042 (50 iterations in 0.278s)
[t-SNE] Iteration 1600: error = 2.1904928, gradient norm = 0.0000040 (50 iterations in 0.288s)
[t-SNE] Iteration 1650: error = 2.1728154, gradient norm = 0.0000039 (50 iterations in 0.267s)
[t-SNE] Iteration 1700: error = 2.1557910, gradient norm = 0.0000037 (50 iterations in 0.284s)
[t-SNE] Iteration 1750: error = 2.1393784, gradient norm = 0.0000036 (50 iterations in 0.313s)
[t-SNE] Iteration 1800: error = 2.1235326, gradient norm = 0.0000034 (50 iterations in 0.270s)
[t-SNE] Iteration 1850: error = 2.1082092, gradient norm = 0.0000033 (50 iterations in 0.266s)
[t-SNE] Iteration 1900: error = 2.0933698, gradient norm = 0.0000032 (50 iterations in 0.267s)
[t-SNE] Iteration 1950: error = 2.0789875, gradient norm = 0.0000031 (50 iterations in 0.283s)
[t-SNE] Iteration 2000: error = 2.0650384, gradient norm = 0.0000030 (50 iterations in 0.270s)
[t-SNE] Iteration 2050: error = 2.0515041, gradient norm = 0.0000029 (50 iterations in 0.264s)
[t-SNE] Iteration 2100: error = 2.0383646, gradient norm = 0.0000028 (50 iterations in 0.267s)
[t-SNE] Iteration 2150: error = 2.0255937, gradient norm = 0.0000027 (50 iterations in 0.281s)
[t-SNE] Iteration 2200: error = 2.0131582, gradient norm = 0.0000026 (50 iterations in 0.269s)
[t-SNE] Iteration 2250: error = 2.0010197, gradient norm = 0.0000025 (50 iterations in 0.314s)
[t-SNE] Iteration 2300: error = 1.9891396, gradient norm = 0.0000025 (50 iterations in 0.300s)
[t-SNE] Iteration 2350: error = 1.9774860, gradient norm = 0.0000024 (50 iterations in 0.268s)
[t-SNE] Iteration 2400: error = 1.9660301, gradient norm = 0.0000024 (50 iterations in 0.269s)
[t-SNE] Iteration 2450: error = 1.9547478, gradient norm = 0.0000023 (50 iterations in 0.265s)
[t-SNE] Iteration 2500: error = 1.9436192, gradient norm = 0.0000023 (50 iterations in 0.267s)
[t-SNE] Iteration 2550: error = 1.9326285, gradient norm = 0.0000022 (50 iterations in 0.279s)
[t-SNE] Iteration 2600: error = 1.9217664, gradient norm = 0.0000022 (50 iterations in 0.264s)
[t-SNE] Iteration 2650: error = 1.9110266, gradient norm = 0.0000021 (50 iterations in 0.272s)
[t-SNE] Iteration 2700: error = 1.9004059, gradient norm = 0.0000021 (50 iterations in 0.268s)
[t-SNE] Iteration 2750: error = 1.8899054, gradient norm = 0.0000021 (50 iterations in 0.266s)
```

```
[t-SNE] Iteration 2800: error = 1.8795266, gradient norm = 0.0000020 (50 iterations in 0.268s)
[t-SNE] Iteration 2850: error = 1.8692713, gradient norm = 0.0000020 (50 iterations in 0.278s)
[t-SNE] Iteration 2900: error = 1.8591438, gradient norm = 0.0000019 (50 iterations in 0.344s)
[t-SNE] Iteration 2950: error = 1.8491526, gradient norm = 0.0000019 (50 iterations in 0.261s)
[t-SNE] Iteration 3000: error = 1.8393064, gradient norm = 0.0000019 (50 iterations in 0.282s)
[t-SNE] Iteration 3050: error = 1.8296145, gradient norm = 0.0000018 (50 iterations in 0.267s)
[t-SNE] Iteration 3100: error = 1.8200862, gradient norm = 0.0000018 (50 iterations in 0.290s)
[t-SNE] Iteration 3150: error = 1.8107272, gradient norm = 0.0000017 (50 iterations in 0.307s)
[t-SNE] Iteration 3200: error = 1.8015394, gradient norm = 0.0000017 (50 iterations in 0.270s)
[t-SNE] Iteration 3250: error = 1.7925195, gradient norm = 0.0000017 (50 iterations in 0.268s)
[t-SNE] Iteration 3300: error = 1.7836625, gradient norm = 0.0000016 (50 iterations in 0.321s)
[t-SNE] Iteration 3350: error = 1.7749618, gradient norm = 0.0000016 (50 iterations in 0.262s)
[t-SNE] Iteration 3400: error = 1.7664094, gradient norm = 0.0000016 (50 iterations in 0.316s)
[t-SNE] Iteration 3450: error = 1.7579984, gradient norm = 0.0000015 (50 iterations in 0.357s)
[t-SNE] Iteration 3500: error = 1.7497231, gradient norm = 0.0000015 (50 iterations in 0.276s)
[t-SNE] Iteration 3550: error = 1.7415785, gradient norm = 0.0000015 (50 iterations in 0.268s)
[t-SNE] Iteration 3600: error = 1.7335599, gradient norm = 0.0000015 (50 iterations in 0.286s)
[t-SNE] Iteration 3650: error = 1.7256629, gradient norm = 0.0000014 (50 iterations in 0.264s)
[t-SNE] Iteration 3700: error = 1.7178842, gradient norm = 0.0000014 (50 iterations in 0.268s)
[t-SNE] Iteration 3750: error = 1.7102207, gradient norm = 0.0000014 (50 iterations in 0.265s)
[t-SNE] Iteration 3800: error = 1.7026674, gradient norm = 0.0000014 (50 iterations in 0.277s)
[t-SNE] Iteration 3850: error = 1.6952217, gradient norm = 0.0000013 (50 iterations in 0.257s)
[t-SNE] Iteration 3900: error = 1.6878846, gradient norm = 0.0000013 (50 iterations in 0.284s)
[t-SNE] Iteration 3950: error = 1.6806645, gradient norm = 0.0000013 (50 iterations in 0.273s)
[t-SNE] Iteration 4000: error = 1.6735827, gradient norm = 0.0000013 (50 iterations in 0.292s)
[t-SNE] Iteration 4050: error = 1.6666724, gradient norm = 0.0000012 (50 iterations in 0.268s)
[t-SNE] Iteration 4100: error = 1.6599561, gradient norm = 0.0000012 (50 iterations in 0.299s)
[t-SNE] Iteration 4150: error = 1.6534155, gradient norm = 0.0000012 (50 iterations in 0.273s)
[t-SNE] Iteration 4200: error = 1.6470141, gradient norm = 0.0000012 (50 iterations in 0.359s)
[t-SNE] Iteration 4250: error = 1.6407283, gradient norm = 0.0000012 (50 iterations in 0.338s)
[t-SNE] Iteration 4300: error = 1.6345481, gradient norm = 0.0000011 (50 iterations in 0.317s)
[t-SNE] Iteration 4350: error = 1.6284678, gradient norm = 0.0000011 (50 iterations in 0.308s)
[t-SNE] Iteration 4400: error = 1.6224810, gradient norm = 0.0000011 (50 iterations in 0.350s)
[t-SNE] Iteration 4450: error = 1.6165821, gradient norm = 0.0000011 (50 iterations in 0.311s)
[t-SNE] Iteration 4500: error = 1.6107663, gradient norm = 0.0000011 (50 iterations in 0.334s)
[t-SNE] Iteration 4550: error = 1.6050292, gradient norm = 0.0000011 (50 iterations in 0.331s)
[t-SNE] Iteration 4600: error = 1.5993659, gradient norm = 0.0000010 (50 iterations in 0.289s)
[t-SNE] Iteration 4650: error = 1.5937721, gradient norm = 0.0000010 (50 iterations in 0.267s)
[t-SNE] Iteration 4700: error = 1.5882437, gradient norm = 0.0000010 (50 iterations in 0.347s)
[t-SNE] Iteration 4750: error = 1.5827765, gradient norm = 0.0000010 (50 iterations in 0.400s)
[t-SNE] Iteration 4800: error = 1.5773678, gradient norm = 0.0000010 (50 iterations in 0.281s)
[t-SNE] Iteration 4850: error = 1.5720147, gradient norm = 0.0000010 (50 iterations in 0.337s)
[t-SNE] Iteration 4900: error = 1.5667141, gradient norm = 0.0000010 (50 iterations in 0.276s)
[t-SNE] Iteration 4950: error = 1.5614637, gradient norm = 0.0000010 (50 iterations in 0.282s)
[t-SNE] Iteration 5000: error = 1.5562606, gradient norm = 0.0000009 (50 iterations in 0.285s)
[t-SNE] Iteration 5050: error = 1.5511026, gradient norm = 0.0000009 (50 iterations in 0.266s)
[t-SNE] Iteration 5100: error = 1.5459885, gradient norm = 0.0000009 (50 iterations in 0.285s)
[t-SNE] Iteration 5150: error = 1.5409172, gradient norm = 0.0000009 (50 iterations in 0.353s)
[t-SNE] Iteration 5200: error = 1.5358883, gradient norm = 0.0000009 (50 iterations in 0.355s)
[t-SNE] Iteration 5250: error = 1.5309027, gradient norm = 0.0000009 (50 iterations in 0.327s)
[t-SNE] Iteration 5300: error = 1.5259600, gradient norm = 0.0000009 (50 iterations in 0.305s)
```

```
[t-SNE] Iteration 5350: error = 1.5210603, gradient norm = 0.0000009 (50 iterations in 0.337s)
[t-SNE] Iteration 5400: error = 1.5162043, gradient norm = 0.0000009 (50 iterations in 0.305s)
[t-SNE] Iteration 5450: error = 1.5113925, gradient norm = 0.0000009 (50 iterations in 0.323s)
[t-SNE] Iteration 5500: error = 1.5066256, gradient norm = 0.0000008 (50 iterations in 0.348s)
[t-SNE] Iteration 5550: error = 1.5019050, gradient norm = 0.0000008 (50 iterations in 0.300s)
[t-SNE] Iteration 5600: error = 1.4972326, gradient norm = 0.0000008 (50 iterations in 0.292s)
[t-SNE] Iteration 5650: error = 1.4926104, gradient norm = 0.0000008 (50 iterations in 0.329s)
[t-SNE] Iteration 5700: error = 1.4880404, gradient norm = 0.0000008 (50 iterations in 0.293s)
[t-SNE] Iteration 5750: error = 1.4835240, gradient norm = 0.0000008 (50 iterations in 0.262s)
[t-SNE] Iteration 5800: error = 1.4790618, gradient norm = 0.0000008 (50 iterations in 0.293s)
[t-SNE] Iteration 5850: error = 1.4746544, gradient norm = 0.0000008 (50 iterations in 0.271s)
[t-SNE] Iteration 5900: error = 1.4703012, gradient norm = 0.0000008 (50 iterations in 0.286s)
[t-SNE] Iteration 5950: error = 1.4660011, gradient norm = 0.0000008 (50 iterations in 0.329s)
[t-SNE] Iteration 6000: error = 1.4617525, gradient norm = 0.0000007 (50 iterations in 0.286s)
[t-SNE] Iteration 6050: error = 1.4575540, gradient norm = 0.0000007 (50 iterations in 0.292s)
[t-SNE] Iteration 6100: error = 1.4534033, gradient norm = 0.0000007 (50 iterations in 0.311s)
[t-SNE] Iteration 6150: error = 1.4492992, gradient norm = 0.0000007 (50 iterations in 0.291s)
[t-SNE] Iteration 6200: error = 1.4452406, gradient norm = 0.0000007 (50 iterations in 0.310s)
[t-SNE] Iteration 6250: error = 1.4412271, gradient norm = 0.0000007 (50 iterations in 0.301s)
[t-SNE] Iteration 6300: error = 1.4372583, gradient norm = 0.0000007 (50 iterations in 0.292s)
[t-SNE] Iteration 6350: error = 1.4333342, gradient norm = 0.0000007 (50 iterations in 0.277s)
[t-SNE] Iteration 6400: error = 1.4294555, gradient norm = 0.0000007 (50 iterations in 0.262s)
[t-SNE] Iteration 6450: error = 1.4256228, gradient norm = 0.0000007 (50 iterations in 0.257s)
[t-SNE] Iteration 6500: error = 1.4218357, gradient norm = 0.0000007 (50 iterations in 0.265s)
[t-SNE] Iteration 6550: error = 1.4180945, gradient norm = 0.0000007 (50 iterations in 0.266s)
[t-SNE] Iteration 6600: error = 1.4143993, gradient norm = 0.0000006 (50 iterations in 0.250s)
[t-SNE] Iteration 6650: error = 1.4107497, gradient norm = 0.0000006 (50 iterations in 0.250s)
[t-SNE] Iteration 6700: error = 1.4071440, gradient norm = 0.0000006 (50 iterations in 0.320s)
[t-SNE] Iteration 6750: error = 1.4035805, gradient norm = 0.0000006 (50 iterations in 0.265s)
[t-SNE] Iteration 6800: error = 1.4000576, gradient norm = 0.0000006 (50 iterations in 0.267s)
[t-SNE] Iteration 6850: error = 1.3965737, gradient norm = 0.0000006 (50 iterations in 0.266s)
[t-SNE] Iteration 6900: error = 1.3931277, gradient norm = 0.0000006 (50 iterations in 0.267s)
[t-SNE] Iteration 6950: error = 1.3897181, gradient norm = 0.0000006 (50 iterations in 0.266s)
[t-SNE] Iteration 7000: error = 1.3863444, gradient norm = 0.0000006 (50 iterations in 0.266s)
[t-SNE] Iteration 7050: error = 1.3830061, gradient norm = 0.0000006 (50 iterations in 0.267s)
[t-SNE] Iteration 7100: error = 1.3797026, gradient norm = 0.0000006 (50 iterations in 0.253s)
[t-SNE] Iteration 7150: error = 1.3764338, gradient norm = 0.0000006 (50 iterations in 0.280s)
[t-SNE] Iteration 7200: error = 1.3731998, gradient norm = 0.0000006 (50 iterations in 0.254s)
[t-SNE] Iteration 7250: error = 1.3700004, gradient norm = 0.0000006 (50 iterations in 0.264s)
[t-SNE] Iteration 7300: error = 1.3668358, gradient norm = 0.0000006 (50 iterations in 0.281s)
[t-SNE] Iteration 7350: error = 1.3637057, gradient norm = 0.0000006 (50 iterations in 0.252s)
[t-SNE] Iteration 7400: error = 1.3606109, gradient norm = 0.0000005 (50 iterations in 0.264s)
[t-SNE] Iteration 7450: error = 1.3575515, gradient norm = 0.0000005 (50 iterations in 0.276s)
[t-SNE] Iteration 7500: error = 1.3545272, gradient norm = 0.0000005 (50 iterations in 0.270s)
[t-SNE] Iteration 7550: error = 1.3515379, gradient norm = 0.0000005 (50 iterations in 0.257s)
[t-SNE] Iteration 7600: error = 1.3485832, gradient norm = 0.0000005 (50 iterations in 0.279s)
[t-SNE] Iteration 7650: error = 1.3456618, gradient norm = 0.0000005 (50 iterations in 0.254s)
[t-SNE] Iteration 7700: error = 1.3427727, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 7750: error = 1.3399151, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 7800: error = 1.3370874, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 7850: error = 1.3342883, gradient norm = 0.0000005 (50 iterations in 0.266s)
```

```
[t-SNE] Iteration 7900: error = 1.3315169, gradient norm = 0.0000005 (50 iterations in 0.268s)
[t-SNE] Iteration 7950: error = 1.3287721, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8000: error = 1.3260526, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 8050: error = 1.3233569, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8100: error = 1.3206843, gradient norm = 0.0000005 (50 iterations in 0.268s)
[t-SNE] Iteration 8150: error = 1.3180340, gradient norm = 0.0000005 (50 iterations in 0.283s)
[t-SNE] Iteration 8200: error = 1.3154045, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 8250: error = 1.3127953, gradient norm = 0.0000005 (50 iterations in 0.251s)
[t-SNE] Iteration 8300: error = 1.3102053, gradient norm = 0.0000005 (50 iterations in 0.264s)
[t-SNE] Iteration 8350: error = 1.3076339, gradient norm = 0.0000005 (50 iterations in 0.269s)
[t-SNE] Iteration 8400: error = 1.3050804, gradient norm = 0.0000005 (50 iterations in 0.269s)
[t-SNE] Iteration 8450: error = 1.3025441, gradient norm = 0.0000005 (50 iterations in 0.264s)
[t-SNE] Iteration 8500: error = 1.3000242, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8550: error = 1.2975201, gradient norm = 0.0000005 (50 iterations in 0.267s)
[t-SNE] Iteration 8600: error = 1.2950316, gradient norm = 0.0000005 (50 iterations in 0.266s)
[t-SNE] Iteration 8650: error = 1.2925580, gradient norm = 0.0000004 (50 iterations in 0.251s)
[t-SNE] Iteration 8700: error = 1.2900991, gradient norm = 0.0000004 (50 iterations in 0.264s)
[t-SNE] Iteration 8750: error = 1.2876540, gradient norm = 0.0000004 (50 iterations in 0.268s)
[t-SNE] Iteration 8800: error = 1.2852225, gradient norm = 0.0000004 (50 iterations in 0.268s)
[t-SNE] Iteration 8850: error = 1.2828040, gradient norm = 0.0000004 (50 iterations in 0.265s)
[t-SNE] Iteration 8900: error = 1.2803980, gradient norm = 0.0000004 (50 iterations in 0.269s)
[t-SNE] Iteration 8950: error = 1.2780038, gradient norm = 0.0000004 (50 iterations in 0.265s)
[t-SNE] Iteration 9000: error = 1.2756208, gradient norm = 0.0000004 (50 iterations in 0.268s)
[t-SNE] Iteration 9050: error = 1.2732486, gradient norm = 0.0000004 (50 iterations in 0.266s)
[t-SNE] Iteration 9100: error = 1.2708863, gradient norm = 0.0000004 (50 iterations in 0.268s)
[t-SNE] Iteration 9150: error = 1.2685337, gradient norm = 0.0000004 (50 iterations in 0.266s)
[t-SNE] Iteration 9200: error = 1.2661899, gradient norm = 0.0000004 (50 iterations in 0.267s)
[t-SNE] Iteration 9250: error = 1.2638547, gradient norm = 0.0000004 (50 iterations in 0.250s)
[t-SNE] Iteration 9300: error = 1.2615273, gradient norm = 0.0000004 (50 iterations in 0.281s)
[t-SNE] Iteration 9350: error = 1.2592077, gradient norm = 0.0000004 (50 iterations in 0.249s)
[t-SNE] Iteration 9400: error = 1.2568955, gradient norm = 0.0000004 (50 iterations in 0.282s)
[t-SNE] Iteration 9450: error = 1.2545905, gradient norm = 0.0000004 (50 iterations in 0.255s)
[t-SNE] Iteration 9500: error = 1.2522924, gradient norm = 0.0000004 (50 iterations in 0.267s)
[t-SNE] Iteration 9550: error = 1.2500015, gradient norm = 0.0000004 (50 iterations in 0.264s)
[t-SNE] Iteration 9600: error = 1.2477180, gradient norm = 0.0000004 (50 iterations in 0.269s)
[t-SNE] Iteration 9650: error = 1.2454420, gradient norm = 0.0000004 (50 iterations in 0.265s)
[t-SNE] Iteration 9700: error = 1.2431743, gradient norm = 0.0000004 (50 iterations in 0.279s)
[t-SNE] Iteration 9750: error = 1.2409152, gradient norm = 0.0000004 (50 iterations in 0.293s)
[t-SNE] Iteration 9800: error = 1.2386651, gradient norm = 0.0000004 (50 iterations in 0.263s)
[t-SNE] Iteration 9850: error = 1.2364249, gradient norm = 0.0000004 (50 iterations in 0.267s)
[t-SNE] Iteration 9900: error = 1.2341947, gradient norm = 0.0000004 (50 iterations in 0.266s)
[t-SNE] Iteration 9950: error = 1.2319749, gradient norm = 0.0000004 (50 iterations in 0.265s)
[t-SNE] Iteration 10000: error = 1.2297653, gradient norm = 0.0000004 (50 iterations in 0.252s)
[t-SNE] Error after 10000 iterations: 1.229765
[t-SNE] Computing pairwise distances...
[t-SNE] Computed conditional probabilities for sample 144 / 144
[t-SNE] Mean sigma: 0.052891
[t-SNE] Iteration 50: error = 90.1861500, gradient norm = 0.0012556 (50 iterations in 0.078s)
[t-SNE] Iteration 100: error = 80.7968271, gradient norm = 0.0002332 (50 iterations in 0.091s)
[t-SNE] Iteration 150: error = 82.0691164, gradient norm = 0.0002620 (50 iterations in 0.089s)
[t-SNE] Iteration 200: error = 100.7204219, gradient norm = 0.0005112 (50 iterations in 0.083s)
```

```
[t-SNE] Iteration 250: error = 87.3432212, gradient norm = 0.0001535 (50 iterations in 0.082s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 87.343221
[t-SNE] Iteration 300: error = 4.3174053, gradient norm = 0.0001362 (50 iterations in 0.087s)
[t-SNE] Iteration 350: error = 4.0699838, gradient norm = 0.0000616 (50 iterations in 0.089s)
[t-SNE] Iteration 400: error = 3.9163227, gradient norm = 0.0000399 (50 iterations in 0.083s)
[t-SNE] Iteration 450: error = 3.7997640, gradient norm = 0.0000289 (50 iterations in 0.087s)
[t-SNE] Iteration 500: error = 3.7035874, gradient norm = 0.0000229 (50 iterations in 0.082s)
[t-SNE] Iteration 550: error = 3.6185973, gradient norm = 0.0000192 (50 iterations in 0.092s)
[t-SNE] Iteration 600: error = 3.5410717, gradient norm = 0.0000166 (50 iterations in 0.092s)
[t-SNE] Iteration 650: error = 3.4696827, gradient norm = 0.0000145 (50 iterations in 0.081s)
[t-SNE] Iteration 700: error = 3.4037718, gradient norm = 0.0000128 (50 iterations in 0.080s)
[t-SNE] Iteration 750: error = 3.3428479, gradient norm = 0.0000114 (50 iterations in 0.071s)
[t-SNE] Iteration 800: error = 3.2864944, gradient norm = 0.0000102 (50 iterations in 0.078s)
[t-SNE] Iteration 850: error = 3.2343132, gradient norm = 0.0000093 (50 iterations in 0.078s)
[t-SNE] Iteration 900: error = 3.1858566, gradient norm = 0.0000084 (50 iterations in 0.062s)
[t-SNE] Iteration 950: error = 3.1406492, gradient norm = 0.0000077 (50 iterations in 0.078s)
[t-SNE] Iteration 1000: error = 3.0982498, gradient norm = 0.0000072 (50 iterations in 0.078s)
[t-SNE] Iteration 1050: error = 3.0582863, gradient norm = 0.0000067 (50 iterations in 0.078s)
[t-SNE] Iteration 1100: error = 3.0204696, gradient norm = 0.0000062 (50 iterations in 0.078s)
[t-SNE] Iteration 1150: error = 2.9845860, gradient norm = 0.0000058 (50 iterations in 0.063s)
[t-SNE] Iteration 1200: error = 2.9504682, gradient norm = 0.0000055 (50 iterations in 0.078s)
[t-SNE] Iteration 1250: error = 2.9179727, gradient norm = 0.0000051 (50 iterations in 0.078s)
[t-SNE] Iteration 1300: error = 2.8869717, gradient norm = 0.0000048 (50 iterations in 0.078s)
[t-SNE] Iteration 1350: error = 2.8573544, gradient norm = 0.0000046 (50 iterations in 0.071s)
[t-SNE] Iteration 1400: error = 2.8290205, gradient norm = 0.0000043 (50 iterations in 0.078s)
[t-SNE] Iteration 1450: error = 2.8018775, gradient norm = 0.0000041 (50 iterations in 0.066s)
[t-SNE] Iteration 1500: error = 2.7758413, gradient norm = 0.0000039 (50 iterations in 0.078s)
[t-SNE] Iteration 1550: error = 2.7508338, gradient norm = 0.0000037 (50 iterations in 0.063s)
[t-SNE] Iteration 1600: error = 2.7267835, gradient norm = 0.0000036 (50 iterations in 0.078s)
[t-SNE] Iteration 1650: error = 2.7036293, gradient norm = 0.0000034 (50 iterations in 0.078s)
[t-SNE] Iteration 1700: error = 2.6813130, gradient norm = 0.0000033 (50 iterations in 0.078s)
[t-SNE] Iteration 1750: error = 2.6597830, gradient norm = 0.0000031 (50 iterations in 0.063s)
[t-SNE] Iteration 1800: error = 2.6389937, gradient norm = 0.0000030 (50 iterations in 0.078s)
[t-SNE] Iteration 1850: error = 2.6189046, gradient norm = 0.0000029 (50 iterations in 0.078s)
[t-SNE] Iteration 1900: error = 2.5994766, gradient norm = 0.0000028 (50 iterations in 0.078s)
[t-SNE] Iteration 1950: error = 2.5806730, gradient norm = 0.0000027 (50 iterations in 0.080s)
[t-SNE] Iteration 2000: error = 2.5624575, gradient norm = 0.0000026 (50 iterations in 0.074s)
[t-SNE] Iteration 2050: error = 2.5447957, gradient norm = 0.0000025 (50 iterations in 0.062s)
[t-SNE] Iteration 2100: error = 2.5276558, gradient norm = 0.0000024 (50 iterations in 0.086s)
[t-SNE] Iteration 2150: error = 2.5110075, gradient norm = 0.0000023 (50 iterations in 0.063s)
[t-SNE] Iteration 2200: error = 2.4948221, gradient norm = 0.0000023 (50 iterations in 0.077s)
[t-SNE] Iteration 2250: error = 2.4790717, gradient norm = 0.0000022 (50 iterations in 0.078s)
[t-SNE] Iteration 2300: error = 2.4637311, gradient norm = 0.0000021 (50 iterations in 0.078s)
[t-SNE] Iteration 2350: error = 2.4487784, gradient norm = 0.0000021 (50 iterations in 0.063s)
[t-SNE] Iteration 2400: error = 2.4341932, gradient norm = 0.0000020 (50 iterations in 0.078s)
[t-SNE] Iteration 2450: error = 2.4199560, gradient norm = 0.0000020 (50 iterations in 0.078s)
[t-SNE] Iteration 2500: error = 2.4060483, gradient norm = 0.0000019 (50 iterations in 0.078s)
[t-SNE] Iteration 2550: error = 2.3924529, gradient norm = 0.0000019 (50 iterations in 0.078s)
[t-SNE] Iteration 2600: error = 2.3791556, gradient norm = 0.0000018 (50 iterations in 0.063s)
[t-SNE] Iteration 2650: error = 2.3661417, gradient norm = 0.0000018 (50 iterations in 0.078s)
[t-SNE] Iteration 2700: error = 2.3533972, gradient norm = 0.0000017 (50 iterations in 0.076s)
```

```
[t-SNE] Iteration 2750: error = 2.3409096, gradient norm = 0.0000017 (50 iterations in 0.078s)
[t-SNE] Iteration 2800: error = 2.3286668, gradient norm = 0.0000017 (50 iterations in 0.078s)
[t-SNE] Iteration 2850: error = 2.3166580, gradient norm = 0.0000016 (50 iterations in 0.063s)
[t-SNE] Iteration 2900: error = 2.3048725, gradient norm = 0.0000016 (50 iterations in 0.078s)
[t-SNE] Iteration 2950: error = 2.2933018, gradient norm = 0.0000015 (50 iterations in 0.078s)
[t-SNE] Iteration 3000: error = 2.2819373, gradient norm = 0.0000015 (50 iterations in 0.078s)
[t-SNE] Iteration 3050: error = 2.2707715, gradient norm = 0.0000015 (50 iterations in 0.062s)
[t-SNE] Iteration 3100: error = 2.2597979, gradient norm = 0.0000014 (50 iterations in 0.078s)
[t-SNE] Iteration 3150: error = 2.2490107, gradient norm = 0.0000014 (50 iterations in 0.078s)
[t-SNE] Iteration 3200: error = 2.2384045, gradient norm = 0.0000014 (50 iterations in 0.081s)
[t-SNE] Iteration 3250: error = 2.2279750, gradient norm = 0.0000014 (50 iterations in 0.082s)
[t-SNE] Iteration 3300: error = 2.2177178, gradient norm = 0.0000013 (50 iterations in 0.086s)
[t-SNE] Iteration 3350: error = 2.2076292, gradient norm = 0.0000013 (50 iterations in 0.090s)
[t-SNE] Iteration 3400: error = 2.1977050, gradient norm = 0.0000013 (50 iterations in 0.086s)
[t-SNE] Iteration 3450: error = 2.1879417, gradient norm = 0.0000013 (50 iterations in 0.090s)
[t-SNE] Iteration 3500: error = 2.1783358, gradient norm = 0.0000012 (50 iterations in 0.094s)
[t-SNE] Iteration 3550: error = 2.1688828, gradient norm = 0.0000012 (50 iterations in 0.083s)
[t-SNE] Iteration 3600: error = 2.1595783, gradient norm = 0.0000012 (50 iterations in 0.085s)
[t-SNE] Iteration 3650: error = 2.1504187, gradient norm = 0.0000012 (50 iterations in 0.082s)
[t-SNE] Iteration 3700: error = 2.1413992, gradient norm = 0.0000011 (50 iterations in 0.084s)
[t-SNE] Iteration 3750: error = 2.1325155, gradient norm = 0.0000011 (50 iterations in 0.082s)
[t-SNE] Iteration 3800: error = 2.1237635, gradient norm = 0.0000011 (50 iterations in 0.092s)
[t-SNE] Iteration 3850: error = 2.1151391, gradient norm = 0.0000011 (50 iterations in 0.084s)
[t-SNE] Iteration 3900: error = 2.1066386, gradient norm = 0.0000011 (50 iterations in 0.072s)
[t-SNE] Iteration 3950: error = 2.0982583, gradient norm = 0.0000010 (50 iterations in 0.080s)
[t-SNE] Iteration 4000: error = 2.0899953, gradient norm = 0.0000010 (50 iterations in 0.066s)
[t-SNE] Iteration 4050: error = 2.0818462, gradient norm = 0.0000010 (50 iterations in 0.082s)
[t-SNE] Iteration 4100: error = 2.0738082, gradient norm = 0.0000010 (50 iterations in 0.063s)
[t-SNE] Iteration 4150: error = 2.0658789, gradient norm = 0.0000010 (50 iterations in 0.078s)
[t-SNE] Iteration 4200: error = 2.0580562, gradient norm = 0.0000010 (50 iterations in 0.078s)
[t-SNE] Iteration 4250: error = 2.0503375, gradient norm = 0.0000009 (50 iterations in 0.078s)
[t-SNE] Iteration 4300: error = 2.0427218, gradient norm = 0.0000009 (50 iterations in 0.063s)
[t-SNE] Iteration 4350: error = 2.0352075, gradient norm = 0.0000009 (50 iterations in 0.078s)
[t-SNE] Iteration 4400: error = 2.0277933, gradient norm = 0.0000009 (50 iterations in 0.078s)
[t-SNE] Iteration 4450: error = 2.0204786, gradient norm = 0.0000009 (50 iterations in 0.078s)
[t-SNE] Iteration 4500: error = 2.0132622, gradient norm = 0.0000009 (50 iterations in 0.078s)
[t-SNE] Iteration 4550: error = 2.0061433, gradient norm = 0.0000009 (50 iterations in 0.063s)
[t-SNE] Iteration 4600: error = 1.9991208, gradient norm = 0.0000009 (50 iterations in 0.087s)
[t-SNE] Iteration 4650: error = 1.9921936, gradient norm = 0.0000008 (50 iterations in 0.088s)
[t-SNE] Iteration 4700: error = 1.9853612, gradient norm = 0.0000008 (50 iterations in 0.082s)
[t-SNE] Iteration 4750: error = 1.9786226, gradient norm = 0.0000008 (50 iterations in 0.080s)
[t-SNE] Iteration 4800: error = 1.9719766, gradient norm = 0.0000008 (50 iterations in 0.084s)
[t-SNE] Iteration 4850: error = 1.9654227, gradient norm = 0.0000008 (50 iterations in 0.083s)
[t-SNE] Iteration 4900: error = 1.9589598, gradient norm = 0.0000008 (50 iterations in 0.082s)
[t-SNE] Iteration 4950: error = 1.9525861, gradient norm = 0.0000008 (50 iterations in 0.084s)
[t-SNE] Iteration 5000: error = 1.9463001, gradient norm = 0.0000008 (50 iterations in 0.081s)
[t-SNE] Iteration 5050: error = 1.9401005, gradient norm = 0.0000007 (50 iterations in 0.084s)
[t-SNE] Iteration 5100: error = 1.9339852, gradient norm = 0.0000007 (50 iterations in 0.092s)
[t-SNE] Iteration 5150: error = 1.9279520, gradient norm = 0.0000007 (50 iterations in 0.078s)
[t-SNE] Iteration 5200: error = 1.9219994, gradient norm = 0.0000007 (50 iterations in 0.075s)
[t-SNE] Iteration 5250: error = 1.9161243, gradient norm = 0.0000007 (50 iterations in 0.073s)
```

```
[t-SNE] Iteration 5300: error = 1.9103235, gradient norm = 0.0000007 (50 iterations in 0.078s)
[t-SNE] Iteration 5350: error = 1.9045941, gradient norm = 0.0000007 (50 iterations in 0.063s)
[t-SNE] Iteration 5400: error = 1.8989335, gradient norm = 0.0000007 (50 iterations in 0.078s)
[t-SNE] Iteration 5450: error = 1.8933401, gradient norm = 0.0000007 (50 iterations in 0.078s)
[t-SNE] Iteration 5500: error = 1.8878109, gradient norm = 0.0000007 (50 iterations in 0.063s)
[t-SNE] Iteration 5550: error = 1.8823436, gradient norm = 0.0000007 (50 iterations in 0.073s)
[t-SNE] Iteration 5600: error = 1.8769363, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5650: error = 1.8715868, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5700: error = 1.8662935, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5750: error = 1.8610548, gradient norm = 0.0000006 (50 iterations in 0.063s)
[t-SNE] Iteration 5800: error = 1.8558697, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 5850: error = 1.8507371, gradient norm = 0.0000006 (50 iterations in 0.068s)
[t-SNE] Iteration 5900: error = 1.8456553, gradient norm = 0.0000006 (50 iterations in 0.087s)
[t-SNE] Iteration 5950: error = 1.8406233, gradient norm = 0.0000006 (50 iterations in 0.066s)
[t-SNE] Iteration 6000: error = 1.8356400, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 6050: error = 1.8307043, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 6100: error = 1.8258155, gradient norm = 0.0000006 (50 iterations in 0.069s)
[t-SNE] Iteration 6150: error = 1.8209726, gradient norm = 0.0000006 (50 iterations in 0.078s)
[t-SNE] Iteration 6200: error = 1.8161747, gradient norm = 0.0000006 (50 iterations in 0.084s)
[t-SNE] Iteration 6250: error = 1.8114216, gradient norm = 0.0000006 (50 iterations in 0.088s)
[t-SNE] Iteration 6300: error = 1.8067126, gradient norm = 0.0000006 (50 iterations in 0.088s)
[t-SNE] Iteration 6350: error = 1.8020476, gradient norm = 0.0000006 (50 iterations in 0.093s)
[t-SNE] Iteration 6400: error = 1.7974266, gradient norm = 0.0000005 (50 iterations in 0.088s)
[t-SNE] Iteration 6450: error = 1.7928489, gradient norm = 0.0000005 (50 iterations in 0.073s)
[t-SNE] Iteration 6500: error = 1.7883149, gradient norm = 0.0000005 (50 iterations in 0.089s)
[t-SNE] Iteration 6550: error = 1.7838244, gradient norm = 0.0000005 (50 iterations in 0.081s)
[t-SNE] Iteration 6600: error = 1.7793774, gradient norm = 0.0000005 (50 iterations in 0.082s)
[t-SNE] Iteration 6650: error = 1.7749741, gradient norm = 0.0000005 (50 iterations in 0.065s)
[t-SNE] Iteration 6700: error = 1.7706144, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 6750: error = 1.7662981, gradient norm = 0.0000005 (50 iterations in 0.090s)
[t-SNE] Iteration 6800: error = 1.7620246, gradient norm = 0.0000005 (50 iterations in 0.081s)
[t-SNE] Iteration 6850: error = 1.7577947, gradient norm = 0.0000005 (50 iterations in 0.080s)
[t-SNE] Iteration 6900: error = 1.7536078, gradient norm = 0.0000005 (50 iterations in 0.067s)
[t-SNE] Iteration 6950: error = 1.7494638, gradient norm = 0.0000005 (50 iterations in 0.105s)
[t-SNE] Iteration 7000: error = 1.7453623, gradient norm = 0.0000005 (50 iterations in 0.081s)
[t-SNE] Iteration 7050: error = 1.7413032, gradient norm = 0.0000005 (50 iterations in 0.065s)
[t-SNE] Iteration 7100: error = 1.7372860, gradient norm = 0.0000005 (50 iterations in 0.086s)
[t-SNE] Iteration 7150: error = 1.7333101, gradient norm = 0.0000005 (50 iterations in 0.066s)
[t-SNE] Iteration 7200: error = 1.7293758, gradient norm = 0.0000005 (50 iterations in 0.088s)
[t-SNE] Iteration 7250: error = 1.7254820, gradient norm = 0.0000005 (50 iterations in 0.078s)
[t-SNE] Iteration 7300: error = 1.7216288, gradient norm = 0.0000005 (50 iterations in 0.074s)
[t-SNE] Iteration 7350: error = 1.7178155, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7400: error = 1.7140415, gradient norm = 0.0000004 (50 iterations in 0.063s)
[t-SNE] Iteration 7450: error = 1.7103063, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7500: error = 1.7066081, gradient norm = 0.0000004 (50 iterations in 0.076s)
[t-SNE] Iteration 7550: error = 1.7029466, gradient norm = 0.0000004 (50 iterations in 0.084s)
[t-SNE] Iteration 7600: error = 1.6993215, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7650: error = 1.6957316, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7700: error = 1.6921762, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7750: error = 1.6886540, gradient norm = 0.0000004 (50 iterations in 0.063s)
[t-SNE] Iteration 7800: error = 1.6851647, gradient norm = 0.0000004 (50 iterations in 0.073s)
```

```
[t-SNE] Iteration 7850: error = 1.6817067, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7900: error = 1.6782800, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 7950: error = 1.6748833, gradient norm = 0.0000004 (50 iterations in 0.084s)
[t-SNE] Iteration 8000: error = 1.6715160, gradient norm = 0.0000004 (50 iterations in 0.068s)
[t-SNE] Iteration 8050: error = 1.6681777, gradient norm = 0.0000004 (50 iterations in 0.071s)
[t-SNE] Iteration 8100: error = 1.6648674, gradient norm = 0.0000004 (50 iterations in 0.082s)
[t-SNE] Iteration 8150: error = 1.6615841, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8200: error = 1.6583272, gradient norm = 0.0000004 (50 iterations in 0.062s)
[t-SNE] Iteration 8250: error = 1.6550960, gradient norm = 0.0000004 (50 iterations in 0.085s)
[t-SNE] Iteration 8300: error = 1.6518901, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8350: error = 1.6487091, gradient norm = 0.0000004 (50 iterations in 0.063s)
[t-SNE] Iteration 8400: error = 1.6455523, gradient norm = 0.0000004 (50 iterations in 0.089s)
[t-SNE] Iteration 8450: error = 1.6424190, gradient norm = 0.0000004 (50 iterations in 0.079s)
[t-SNE] Iteration 8500: error = 1.6393086, gradient norm = 0.0000004 (50 iterations in 0.063s)
[t-SNE] Iteration 8550: error = 1.6362209, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8600: error = 1.6331550, gradient norm = 0.0000004 (50 iterations in 0.086s)
[t-SNE] Iteration 8650: error = 1.6301103, gradient norm = 0.0000004 (50 iterations in 0.070s)
[t-SNE] Iteration 8700: error = 1.6270865, gradient norm = 0.0000004 (50 iterations in 0.078s)
[t-SNE] Iteration 8750: error = 1.6240833, gradient norm = 0.0000003 (50 iterations in 0.063s)
[t-SNE] Iteration 8800: error = 1.6211003, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 8850: error = 1.6181368, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 8900: error = 1.6151925, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 8950: error = 1.6122667, gradient norm = 0.0000003 (50 iterations in 0.063s)
[t-SNE] Iteration 9000: error = 1.6093594, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9050: error = 1.6064696, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9100: error = 1.6035980, gradient norm = 0.0000003 (50 iterations in 0.081s)
[t-SNE] Iteration 9150: error = 1.6007433, gradient norm = 0.0000003 (50 iterations in 0.067s)
[t-SNE] Iteration 9200: error = 1.5979057, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9250: error = 1.5950846, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9300: error = 1.5922794, gradient norm = 0.0000003 (50 iterations in 0.063s)
[t-SNE] Iteration 9350: error = 1.5894902, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9400: error = 1.5867166, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9450: error = 1.5839579, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9500: error = 1.5812143, gradient norm = 0.0000003 (50 iterations in 0.062s)
[t-SNE] Iteration 9550: error = 1.5784856, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9600: error = 1.5757711, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9650: error = 1.5730705, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9700: error = 1.5703834, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9750: error = 1.5677102, gradient norm = 0.0000003 (50 iterations in 0.067s)
[t-SNE] Iteration 9800: error = 1.5650500, gradient norm = 0.0000003 (50 iterations in 0.083s)
[t-SNE] Iteration 9850: error = 1.5624029, gradient norm = 0.0000003 (50 iterations in 0.066s)
[t-SNE] Iteration 9900: error = 1.5597685, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Iteration 9950: error = 1.5571467, gradient norm = 0.0000003 (50 iterations in 0.063s)
[t-SNE] Iteration 10000: error = 1.5545366, gradient norm = 0.0000003 (50 iterations in 0.078s)
[t-SNE] Error after 10000 iterations: 1.554537
```

```
regressors=[kernel_ridge_1, rf_2, rf_3, lasso_lars1, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9, rf_10,
lasso_lars2, kernel_ridge_2, rf_11, kernel_ridge_3, lasso_lars3, kernel_ridge_4]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                    meta_regressor=rf_1)
```

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res_new = []
for i in predicted_data:
    const_pred = 910
    if i > const_pred:
        res_new += [const_pred]
    else:
        res_new += [i]
#save_predicted_data(res)
#32....
```

```
(1004, 51)
```

```
regressors=[rf_2, rf_3, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9, rf_10, lasso_lars2, kernel_ridge_2,
bay_ridge, kernel_ridge_3, lasso_lars3, kernel_ridge_4]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                           meta_regressor=rf_1)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res_new1 = []
for i in predicted_data:
    const_pred = 910
    if i > const_pred:
        res_new1 += [const_pred]
    else:
        res_new1 += [i]
#save_predicted_data(res)
#32....
```

```
(1004, 51)

regressors=[rf_2, rf_3, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9, rf_10, rf_11, kernel_ridge_3, lasso_lars3,
kernel_ridge_4]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                    meta_regressor=rf_1)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res_new2 = []
for i in predicted_data:
   const_pred = 910
   if i > const_pred:
      res_new2 += [const_pred]
   else:
      res_new2 += [i]
#save_predicted_data(res)
#32....

(1004, 48)

save_predicted_data([round((r1+r2+r3)/3) for r1,r2,r3 in zip(res1, res2, res4)])

regressors=[rf_2, rf_3, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9, rf_10]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                    meta_regressor=rf_11)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res4 = []
for i in predicted_data:
    const_pred = 910
    if i > const_pred:
        res4 += [const_pred]
    else:
        res4 += [i]
#save_predicted_data(res)

(1004, 51)

regressors=[rf_2, rf_3, rf_4, rf_5, rf_6, rf_7, rf_8]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                          meta_regressor=rf_11)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res44 = []
for i in predicted_data:
    const_pred = 910
    if i > const_pred:
        res44 += [const_pred]
    else:
        res44 += [i]
#save_predicted_data(res)

(1004, 51)

regressors=[rf_2, rf_3, rf_4, rf_5, rf_6]
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                          meta_regressor=rf_11)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)



predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res444 = []
for i in predicted_data:
    const_pred = 910
    if i > const_pred:
        res444 += [const_pred]
    else:
        res444 += [i]
#save_predicted_data(res)

(1004, 51)

result_2_mean = [round((r1+r2)/2) for r1,r2 in zip(res4, res_new)]######
save_predicted_data([x for x in result_2_mean])

saved as  y_test_data-3503.txt

True
result_2_mean = [round((r1+r2)/2) for r1,r2 in zip(res_new2, res_new)]######33.2
save_predicted_data([x for x in result_2_mean])

saved as  y_test_data-3010.txt

True
result_2_mean = [round((r1+r2)/2) for r1,r2 in zip(res_new, res_old)]#
save_predicted_data([x for x in result_2_mean])

saved as  y_test_data-6740.txt

True
result_2_mean = [round((r1+r2)/2) for r1,r2 in zip(res_old, res4)]##33.3
save_predicted_data([x for x in result_2_mean])

saved as  y_test_data-9081.txt

True
rf_1 =  sklearn.ensemble.ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth =
3, random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
kernel_ridge_1 = KernelRidge(alpha=0.1)###########################30.12
kernel_ridge_2 = KernelRidge(alpha=0.1)
kernel_ridge_3 = KernelRidge(alpha=0.1)
kernel_ridge_4 = KernelRidge(alpha=0.1)
lasso_lars1 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-4)
lasso_lars2 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-3)
lasso_lars3 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-2)
lasso_lars4 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-1)
rf_1 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_10 =  ExtraTreesRegressor(n_estimators=3000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_2 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_7 =  ExtraTreesRegressor(n_estimators=6000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_8 =  ExtraTreesRegressor(n_estimators=7000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_9 =  ExtraTreesRegressor(n_estimators=8000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_3 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_4 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 4,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_5 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_11 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_12 =  ExtraTreesRegressor(n_estimators=8000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_6 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 6,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_13 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 8,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
regressors=[kernel_ridge_1, rf_2, rf_3, lasso_lars1, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9, rf_10,
lasso_lars2, kernel_ridge_2, rf_11, kernel_ridge_3, lasso_lars3, kernel_ridge_4]

from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=regressors,
                          meta_regressor=rf_1)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)
predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res3 = []
for i in predicted_data:
    if i > 910:
        res3 += [910]
    else:
        res3 += [i]
```

```
(1004, 48)
```

```
from sklearn.linear_model import ElasticNet, Lars, LassoLars, TheilSenRegressor,
RANSACRegressor, HuberRegressor, LogisticRegression
from sklearn.kernel_ridge import KernelRidge
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn import svm
rf_1 =  sklearn.ensemble.ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth =
3, random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
reg1 = KernelRidge(alpha=0.1)
reg1_1 = KernelRidge(alpha=0.1)
reg1_2 = KernelRidge(alpha=0.1)
reg1_3 = KernelRidge(alpha=0.1)
reg2 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-2)
reg2_3 = LassoLars(alpha=1.0, positive=True, eps=2.220446049250313e-2)
reg2_2 = LassoLars(alpha=1.0, positive=True)
rf_1 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_2 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_3 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_4 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 4,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_5 =  ExtraTreesRegressor(n_estimators=2000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_6 =  ExtraTreesRegressor(n_estimators=5000, criterion='mse', max_depth = 6,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_7 =  ExtraTreesRegressor(n_estimators=6000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_8 =  ExtraTreesRegressor(n_estimators=8000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_9 =  ExtraTreesRegressor(n_estimators=7000, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_10 =  ExtraTreesRegressor(n_estimators=5500, criterion='mse', max_depth = 3,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_11 =  ExtraTreesRegressor(n_estimators=10000, criterion='mse', max_depth = 5,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_12 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 7,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
rf_13 =  ExtraTreesRegressor(n_estimators=1000, criterion='mse', max_depth = 8,
random_state=42, n_jobs=-1, oob_score = True, bootstrap = True)
from mlxtend.regressor import StackingRegressor, StackingCVRegressor
stregr = StackingRegressor(regressors=[reg1, rf_2, rf_3, reg2, rf_4, rf_5, rf_6, rf_7, rf_8, rf_9,
rf_10, reg2_2, reg1_2, rf_11, reg1_1, reg2_3, reg1_3],
                        meta_regressor=rf_1)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_normalized, y_in)
print(X_resampled.shape)
X_resampled = pd.DataFrame(data=X_resampled, columns=X_normalized.columns)
stregr.fit(X_resampled, y_resampled)
predicted_data = stregr.predict(X_test_normalized)
predicted_data = [int(round(x)) for x in predicted_data]
res_old = []
for i in predicted_data:
    if i > 910:
        res_old += [910]
    else:
        res_old += [i]
#save_predicted_data(res)
#reg3 = HuberRegressor(max_iter=1000, alpha=0.1, epsilon=2)

(1004, 48)
```

Код программы на языке Pyton, обеспечивающий решение задачи №2

```
import pandas as pd
import numpy as np
import sklearn
import os
pd.options.display.max_columns = 100
folder_path = 'C:\\Users\\erqups\\Task2'
if not os.getcwd() == folder_path:
    os.chdir(folder_path)
X_train = pd.read_csv("X_train.csv", encoding="windows-1251")
y_train = pd.read_csv("y_train.csv", encoding="windows-1251", header=None)

import random
from fancyimpute import KNN


def normalize_date(date):
    date = datetime.strptime(date, "%d.%m.%Y")
    return time.mktime(date.timetuple())


def save_predicted_data(predicted_data):
    fname = "y_test_data-{}.txt".format(random.randint(1, 10000))
    with open(fname, "w") as y_test:
        for row in predicted_data:
            y_test.write(str(row))
            y_test.write("\n")
    print("saved as ", fname)
    return True


def process_dataset(x):
    del x["Encoded_FIO"], x['RS1'], x['SEX']

    # sex_repl = {
    #    "М":1,
    #    "м":1,
    #    "Ж":0,
    # "ж":0
    #}
    #x["SEX"].replace(sex_repl, inplace=True)
    localization_repl = {'Fr': 1, 'Oc': 2, 'Te': 3, 'Pa': 4, 'SC': 5, 'Pi': 6, 'Ce': 7, 'BS': 8, 'GB': 9, 'FrPa': 
10, 'VL': 11,
                         'PC': 12, 'н/д': 0, 'V3': 13, 'fr': 14, 'OC': 15, 'Vl': 16, 'te': 17, 'V1': 18, 'Eye': 19, 
'V4': 20, 'PaTe': 21,
                         'Op': 0, 'FrOc': 0, 'Bs': 0}
    x['Localization'].replace(localization_repl, inplace=True)

    histology_repl = {'Меланома': 1, 'РМЖ': 2, 'КРР': 3,
                      'НМРЛ': 4, 'Другой': 0, 'РП': 5, 'РЯ': 6, 'МРЛ': 7, 'РШМ': 8}
    x['CANCER_HISTOLOGY'].replace(histology_repl, inplace=True)

    bad_float_columns = [
        "V1",
        "PD1",
        "PI1",
        "MV12GY_1",
        "MV10GY_1",
        "NV12GY_1",
        "NV10GY_1",
    ]

    for column in bad_float_columns:
        x[column] = x[column].astype("str")
        x[column] = [size.replace(",", ".") for size in x[column]]
        bad_vals = ["nan", "н/д", "н\д", "н.д"]
        res_list = []
        for itr in range(len(x[column])):
            row = x.iloc[itr][column]
            if row in bad_vals:
                res_list += [np.nan]
            else:
                res_list += [float(row)]
            itr += 1
        x[column] = [0] * len(x[column])
        x[column] = x[column].astype("float64")
        x[column] = res_list
        x[column] = x[column].fillna(x[column].median())
    x['V1'] *= 100
    x['MV12GY_1'] *= 100
    x['NV12GY_1'] *= 100
    x['NV10GY_1'] *= 100

    proccessed_column = "NV12GY_1"
    new_column = "neg  NV12GY_1"
    processed_ind = []
    for itr in range(len(x[proccessed_column])):
        obj = x.iloc[itr][proccessed_column]
        if obj < 0:
            processed_ind += [1]
        else:
            processed_ind += [0]
    x[new_column] = processed_ind

    proccessed_column = "NV10GY_1"
    new_column = "neg NV10GY_1"
    processed_ind = []
    for itr in range(len(x[proccessed_column])):
        obj = x.iloc[itr][proccessed_column]
        if obj < 0:
            processed_ind += [1]
        else:
            processed_ind += [0]
    x[new_column] = processed_ind

process_dataset(X_train)
X_test = pd.read_csv("X_test.csv", encoding="windows-1251")
process_dataset(X_test)

columns_used = ['Localization',
                'CANCER_HISTOLOGY',
                'V1',
                'PD1',
                'PI1',
                'MV12GY_1',
                'MV10GY_1',
                'NV12GY_1',
                'NV10GY_1',
                'neg  NV12GY_1',
                'neg NV10GY_1']

X_train = X_train[columns_used]
X_test = X_test[columns_used]

used_cols = X_train.columns


#X_train = KNN(k=3).complete(X_train)
#X_train = pd.DataFrame(data=X_train, columns=used_cols)
#from fancyimpute import KNN##
#
#class_0_sample = sample_full[sample_full[0] == 0]
#del class_0_sample[0]
#class_1_sample = sample_full[sample_full[0] == 1]
#del class_1_sample[0]###

#print(class_0_sample.shape, class_1_sample.shape)
#KNN_0 = KNN(k=5)
#KNN_1 = KNN(k=5)
#class_0_sample = pd.DataFrame(data=KNN_0.complete(class_0_sample), columns=used_cols)
#class_1_sample = pd.DataFrame(data=KNN_1.complete(class_1_sample), columns=used_cols)
#X_train = pd.concat((class_0_sample, class_1_sample), axis=0)
```

```
#X_train.shape

import numpy as np
for col in X_train.columns:
    for pol in range(2, 9):
        col_name = col + " ^{}".format(pol)
        X_train[col_name] = (X_train[col] + 1) ** pol
        X_test[col_name] = (X_test[col] + 1) ** pol

    col_name = col + " cos".format(pol)
    X_train[col_name] = np.cos(X_train[col])
    X_test[col_name] = np.cos(X_test[col])
    col_name = col + " cos + 1".format(pol)
    X_train[col_name] = np.cos(X_train[col] + 1)
    X_test[col_name] = np.cos(X_test[col] + 1)

import xgboost
print(xgboost.__version__)
from xgboost import XGBClassifier
xgb = XGBClassifier(learning_rate=0.01, n_estimators=100, seed=42)
xgb.fit(X_train, y_train, sample_weight = [x*11 + 1 for x in y_train[0]])
y_test = xgb.predict(X_test)
save_predicted_data(y_test)
```

```
0.7

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-2969.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
y_test
```

```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)


0.7
```

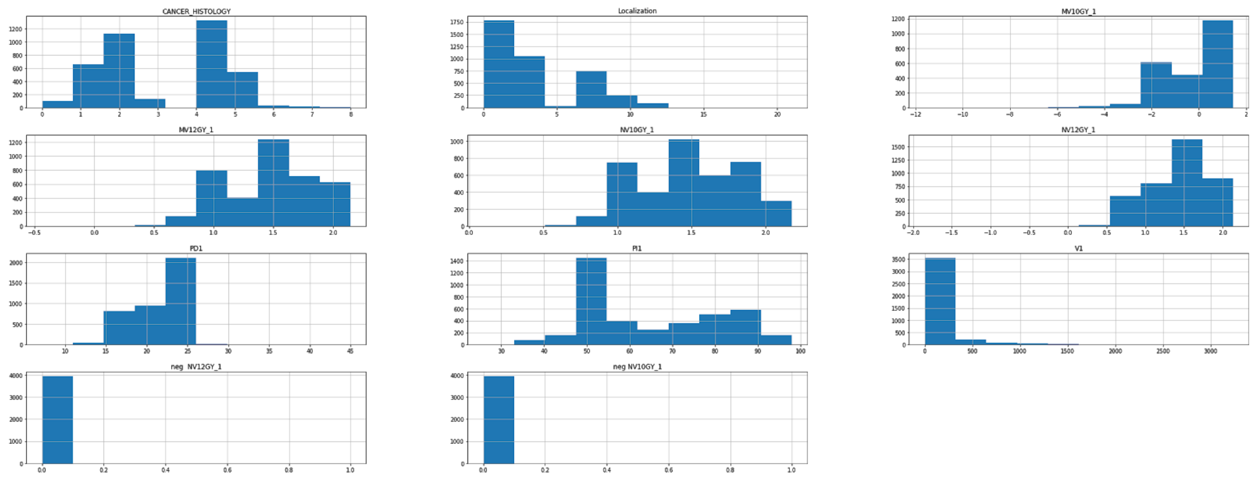
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-8391.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True


0.7

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-2971.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True


0.7

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
saved as  y_test_data-1270.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
#logarithm_features = [
#    "NV12GY_1",
#    "NV10GY_1",
#    "MV10GY_1",
#    "MV12GY_1",
#]
#for feature in logarithm_features:
#    log_feature = np.log(X_train[feature])
#    X_train[feature] = log_feature
#
#    log_feature = np.log(X_train[feature])
#    X_train[feature] = log_feature

X_train.columns
```

```
Index(['Localization', 'CANCER_HISTOLOGY', 'V1', 'PD1', 'PI1', 'MV12GY_1',
       'MV10GY_1', 'NV12GY_1', 'NV10GY_1', 'neg  NV12GY_1', 'neg NV10GY_1'],
      dtype='object')
```

```
import seaborn as sns
%matplotlib inline
raspr = X_train.hist(figsize=(40,15))
```



```
0.7

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-3247.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
#for col in used_cols:
#    for col1 in used_cols:
#        col_name = "{} + {}".format(col1, col)
#        alt_col_name = "{} + {}".format(col, col1)
#        if (not col_name in X_train.columns) and (not alt_col_name in X_train.columns):
#            X_train[col_name] = X_train[col] + X_train[col1]
#
#        if (not col_name in X_test.columns) and (not alt_col_name in X_test.columns):
#            X_test[col_name] = X_test[col] + X_test[col1]
#
#        col_name = "{} * {}".format(col1, col)
#        alt_col_name = "{} * {}".format(col, col1)
#        if (not col_name in X_train.columns) and (not alt_col_name in X_train.columns):
#            X_train[col_name] = X_train[col] * X_train[col1]
#
#        if (not col_name in X_test.columns) and (not alt_col_name in X_test.columns):
#            X_test[col_name] = X_test[col] * X_test[col1]
#
#
#        col_name = "{} - {}".format(col1, col)
#        alt_col_name = "{} - {}".format(col, col1)
#
#        if (not col_name in X_train.columns) and (not alt_col_name in X_train.columns):
#            X_train[col_name] = X_train[col] - X_train[col1]
#
#        if (not col_name in X_test.columns) and (not alt_col_name in X_test.columns):
#            X_test[col_name] = X_test[col] - X_test[col1]
#
#        print(X_train.shape)
```

```
%matplotlib inline
%notebook inline
#import seaborn as sns
#res = X_train.hist(figsize=(40,15))
```

```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
rf = RandomForestClassifier(n_estimators=1000, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
#rf.fit(X_train, y_train)
#y_test = rf.predict(X_test.as_matrix())
#save_predicted_data(y_test)

dff = pd.DataFrame(X_resampled)
dff.columns = ['Localization', 'CANCER_HISTOLOGY', 'V1', 'PD1', 'PI1', 'MV12GY_1',
'MV10GY_1', 'NV12GY_1', 'NV10GY_1', 'neg  NV12GY_1', 'neg NV10GY_1']

X_train = X_train[used_cols]

best_params = {'colsample_bytree': 0.5,
 'gamma': 1,
 'learning_rate': 0.01,
 'max_depth': 3,
 'min_child_weight': 6,
 'n_estimators': 100,
 'scale_pos_weight': 13,
 'subsample': 1}
```

```
0.7

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-4655.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
rf = ExtraTreesClassifier(n_estimators=1000, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf1 = ExtraTreesClassifier(n_estimators=1000, max_depth = 4, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf3 = ExtraTreesClassifier(n_estimators=1000, max_depth = 5, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf4 = ExtraTreesClassifier(n_estimators=1000, max_depth = 6, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
```

```
from mlxtend.classifier import StackingClassifier
eclf1 = StackingClassifier(classifiers=[rf, rf3, rf4],
                          meta_classifier=xgb)
eclf1 = eclf1.fit(X_train, y_train)
save_predicted_data(eclf1.predict(X_test))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\classifier\stacking_classification.py:126:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  clf.fit(X, y)
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\classifier\stacking_classification.py:126:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  clf.fit(X, y)
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\classifier\stacking_classification.py:126:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  clf.fit(X, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-6828.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
from sklearn.linear_model import SGDClassifier
cls = SGDClassifier(loss='squared_hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
epsilon=0.1,
                n_jobs=-1, random_state=42,
                learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight={1: 12},
average=False, n_iter=2000)

cls1 = SGDClassifier(loss='squared_hinge', penalty='l1', alpha=0.001, l1_ratio=0.15,
epsilon=0.1,
                n_jobs=-1, random_state=42,
                learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight={1: 12},
average=False, n_iter=2000)

cls2 = SGDClassifier(loss='squared_hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
epsilon=0.1,
                n_jobs=-1, random_state=42,
                learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight={1: 12},
average=False, n_iter=2000)
#cls.fit(X_train, y_train)

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
rf = RandomForestClassifier(n_estimators=100, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf1 = RandomForestClassifier(n_estimators=1000, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf2 = RandomForestClassifier(n_estimators=5000, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf3 = RandomForestClassifier(n_estimators=8000, max_depth = 4, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight={1:12})
rf4 = RandomForestClassifier(n_estimators=10000, max_depth = 5, random_state=42,
n_jobs=-1, oob_score = True, bootstrap = True, class_weight={1:12})
rf5 = RandomForestClassifier(n_estimators=10000, max_depth = 6, random_state=42,
n_jobs=-1, oob_score = True, bootstrap = True, class_weight={1:12})
rf6 = RandomForestClassifier(n_estimators=10000, max_depth = 7, random_state=42,
n_jobs=-1, oob_score = True, bootstrap = True, class_weight={1:12})
rf7 = RandomForestClassifier(n_estimators=10000, max_depth = 8, random_state=42,
n_jobs=-1, oob_score = True, bootstrap = True, class_weight={1:12})



#rf.fit(X_train, y_train)
#y_test = rf.predict(X_test.as_matrix())
#save_predicted_data(y_test)

#save_predicted_data(cls.predict(X_test))
best_params = {'colsample_bytree': 0.5,
 'gamma': 1,
 'learning_rate': 0.01,
 'max_depth': 3,
 'min_child_weight': 6,
 'n_estimators': 100,
 'scale_pos_weight': 13,
 'subsample': 1}
#GridSearchCV
import xgboost
#print(xgboost.__version__)
from xgboost import XGBClassifier
xgb = XGBClassifier(**best_params, seed=42)
xgb2 = XGBClassifier(**best_params, seed=42)
xgb3 = XGBClassifier(**best_params, seed=42)
#xgb.fit(X_train, y_train)
#y_test = xgb.predict(X_test)
#save_predicted_data(y_test)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
```

```
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
```

```
from catboost import CatBoostClassifier
cat = CatBoostClassifier(depth=3, learning_rate=0.01, n_estimators=1000, random_seed=42)
#cat.fit(X_train, y_train, sample_weight = [x*11 + 1 for x in y_train[0]])

y_test = cat.predict(X_test)
save_predicted_data([int(i) for i in y_test])

from sklearn.decomposition import PCA

def add_pca(x):
   pca = PCA(n_components=3, svd_solver='full', whiten=True, tol=0.0, iterated_power='auto',
random_state=42)
   pca_feats = pd.DataFrame(pca.fit_transform(x)*100)
   return pd.concat((x, pca_feats), axis=1)
X_train = add_pca(X_train)
X_test = add_pca(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
```

```
from mlxtend.classifier import StackingClassifier
eclf1 = StackingClassifier(classifiers=[rf2, rf3, cls],
                          meta_classifier=xgb)
eclf1 = eclf1.fit(X_train, y_train)
save_predicted_data(eclf1.predict(X_test))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:547:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
```

```
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\classifier\stacking_classification.py:126:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  clf.fit(X, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\classifier\stacking_classification.py:134:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  self.meta_clf_.fit(meta_features, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

saved as  y_test_data-6900.txt

True
eclf1 = VotingClassifier(estimators=[('sgd', rf2), ('rf', rf3), ('xgb', cls)])
eclf1 = eclf1.fit(X_train, y_train)
save_predicted_data(eclf1.predict(X_test))

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-7701.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:73:
DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use
max_iter and tol instead.
  DeprecationWarning)



C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:95:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:128:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

saved as  y_test_data-7646.txt

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151:
DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in
future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:

True
```

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0, ratio="auto")
X_resampled, y_resampled = ros.fit_sample(X_train, y_train)
#from imblearn.over_sampling import SMOTE, ADASYN
#X_resampled, y_resampled = SMOTE().fit_sample(X_train, y_train)
#X_resampled, y_resampled = ADASYN().fit_sample(X_train, y_train)
#print(X_resampled.shape)
#from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=1000, max_depth = 3, random_state=42, n_jobs=-1,
oob_score = True, bootstrap = True, class_weight="balanced")
#rf.fit(X_resampled, y_resampled)

#y_test = rf.predict(X_test.as_matrix())
#save_predicted_data(y_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:547:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
from sklearn.linear_model import SGDClassifier
```

```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
cv_params = {'loss': ['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'],
            'penalty': ['none', 'l2', 'l1', 'elasticnet'],
            'alpha': [0.1, 0.5, 1, 5, 10],
            'class_weight': [{1:12}, "balanced"]
           }
ExtraTrees_reg = GridSearchCV(SGDClassifier(), cv_params, scoring ='f1', n_jobs = -1)
ExtraTrees_reg.fit(X_train.as_matrix(), y_train.as_matrix().reshape(len(y_train)))
```

```
from sklearn.model_selection import cross_val_score
cv = cross_val_score(xgb, X_train.as_matrix(), y_train.as_matrix().reshape(len(y_train)), cv=5,
scoring='f1')
print(cv)
cv.mean()
#0.20459782250174768 1:16
# 1:10 17.4
#0.24306072389077488 1:10 17.218543046357617
#0.2598858045911596 1:8
```

```
#process_dataset(X_test)
```

```
X_test.keys()
```

```
null_columns=X_test.columns[X_test.isnull().any()]
X_test[null_columns].isnull().sum()
```