Разбор ИИ тура

Никитин Тимофей • Чернов Владимир

15.01.2026

ИИ тур

A. Яндекс.Карты

Описание задачи

Вам предстоит работать с данными, собранными из Яндекс.Карт — популярного сервиса, предоставляющего информацию о тысячах организаций по всей стране. Эти данные содержат информацию о ресторанах и кафе, такую как режим работы, средний чек, рейтинг и другое.

Задача состоит в том, чтобы на основе этих данных ответить на ряд исследовательских вопросов, требующих анализа разнородной информации: от времени работы до статистики по категориям.

Цель задания

Цель – научиться извлекать полезные инсайты из реального набора данных, используя навыки:

- Предварительной обработки данных
- Агрегации и фильтрации
- Работы со временем, текстом и числами

Предоставленные данные

Датасет task_1.csv содержит следующие поля (все данные взяты с Яндекс.Карт):

- `org_id`: Уникальный идентификатор организации
- `city`: Город расположения заведения
- `average_bill`: Средний чек в заведении
- `rating` Рейтинг заведения
- `category_1`: Первая категория заведения
- `category_2`: Вторая категория заведения(если есть)
- `features`: Особенности заведения
- `latitude`: Географическая широта
- `longitude`: Географическая долгота
- `weekday_hours`: Режим работы в будние дни
- `holiday_hours`: Режим работы в выходные/праздничные дни

Формат ответа

Ответы на каждый вопрос необходимо предоставить в виде:

- Числа (например: 1234)
- Краткой строки (например: Бургеры, Средиземноморская кухня)

Формат ответа для каждого вопроса будет указан в условии.

[Датасет] [Jupyter Ноутбук] (Осторожно, сразу начнется скачивание)

A1. Яндекс.Карты

Сколько всего ресторанов и кафе в Москве открыты в обычный четверг после 21:00?

Для начала поймем что значит

- обычный четверг
- открыты после `21:00`
- в Москве
- рестораны и кафе

Посмотрим на данные, что нам приготовили:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
```

```
df = pd.read_csv('task_1.csv').drop(columns = ['Unnamed: 0'])
df.head(3)
```

	org_id	city	average_bill	rating	category_1	category_2	features	Latitude	Longitude	weekday_hours	holiday_hours
0	15903868628669802651	msk	1500.0	4.270968	Ресторан	Кафе	['table_games', 'online_takeaway', 'payment_by...	37.83724	55.87588	09:00-01:00	09:00-23:00
1	16076540698036998306	msk	500.0	4.375000	Быстрое питание	-	['special_menu', 'features_institution', 'coff...	37.37428	55.66106	07:00-21:00	07:00-19:00
2	8129364761615040323	msk	500.0	4.000000	Кофейня	-	['average_bill2', 'breakfast', 'takeaway', 'su...	37.61042	55.90105	08:00-22:00	08:00-20:00

Нетрудно увидеть, что мы получили всю информацию что хотели:

- Обычный четверг — `weekday_hours`
- Открыты после `21:00` — `21:01` принадлежит рабочим часам
- `city = "msk"`
- `category_1.isin(["Ресторан", "Кафе"]) | category_2.isin(["Ресторан", "Кафе"])`

```
# Вариант 1, через присваивания к интам
times = df['weekday_hours'].str.split('-', expand=True)
end_str = times[1].str.replace('24:00', '00:00', regex=False)
end_h = end_str.str[:2].astype(int)
start_str = times[0].str.replace('24:00', '00:00', regex=False)
start_h = start_str.str[:2].astype(int)
mask_time = (end_h > 21) | (end_h <= start_h)

mask_city_cat = (df['city'] == 'msk') & (df['category_1'].isin(['Ресторан', 'Кафе']) |
df['category_2'].isin(['Ресторан', 'Кафе']))

print((mask_time & mask_city_cat).sum())
#df_res = df.loc[mask_time & mask_city_cat, ['city', 'weekday_hours', 'category_1',
'category_2']]
```

17186

```
# Вариант 2, работа с датами
cur_str = "21:01"
cur = pd.to_datetime(cur_str, format="%H:%M").time()
times = df['weekday_hours'].str.split('-', expand=True)
```

```
start = pd.to_datetime(times[0].str.replace('24:00','00:00', regex=False),
                       format="%H:%M", errors='coerce').dt.time
end   = pd.to_datetime(times[1].str.replace('24:00','00:00', regex=False),
                       format="%H:%M", errors='coerce').dt.time

mask_normal = (start < end) & (start < cur) & (cur < end)
mask_overnight = (start > end) & ((cur > start) | (cur < end))

mask_time = mask_normal | mask_overnight | (start == end)
mask_city_cat = (df['city'] == 'msk') & (df['category_1'].isin(['Ресторан', 'Кафе']) |
df['category_2'].isin(['Ресторан', 'Кафе']))

print((mask_time & mask_city_cat).sum())
# df.loc[mask_time & mask_city_cat, ['city', 'weekday_hours', 'category_1', "category_2" ]]
```

17186

A2. Яндекс.Карты

Какой процент пиццерий в Санкт-Петербурге имеют оценку больше 4.5?

Что нужно заметить в данной задаче?

- Имеют оценку **больше** 4.5, то есть строго больше, а не $\geq$.
- Пиццерий (`category_1.isin(["Пиццерия"]) | category_2.isin(["Пиццерия"])`)
- В Санкт-Петербурге. (`city = "spb"`)

Посчитаем количество пиццерий в Санкт-Петербурге и количество пиццерий в Санкт-Петербурге с рейтингом > 4.5. Поделим второе число на первое – это и есть процент. Осталось грамотно округлить. Если вдруг забылись все правила округления, в `python` есть функция `round(x, digits)`, которая округляет до `digits` знаков после запятой.

```
mask1 = (df['city']=='spb') &\
    (df['category_1'].isin(['Пиццерия']) |\
    df['category_2'].isin(['Пиццерия']))
mask2 = df['rating'] > 4.5
print(round((mask1 & mask2).sum()/(mask1).sum() * 100, 1))
# df.loc[mask, ['rating', 'city', 'category_1', 'category_2']]
```

16.4

A3. Яндекс.Карты

Сколько кофеен, булочных/пекарен и кондитерских в Москве находятся в радиусе не более 1.5 км от отделения ШАД?

Координаты ШАД:

- `Longitude` (долгота) = 55.733705
- `Latitude` (широта) = 37.589482

Примечание:

Формула для рассчета расстояние между двумя точками:

$$d = \arccos\big(\sin(\varphi_\alpha) \cdot \sin(\varphi_\beta) + \cos(\varphi_\alpha) \cdot \cos(\varphi_\beta) \cdot \cos(\lambda_\alpha - \lambda_\beta)\big)$$

Где $\varphi_\alpha.\varphi_\beta$ – широты **в радианах**; $\lambda_\alpha, \lambda_\beta$ – долготы **в радианах**, d – расстояние между точками.

Для перевода координат в радианы используйте функцию `math.radians()`.

Таким образом:

- $\varphi_\alpha = \text{math.radians}(\text{Latitude}_\alpha)$
- $\varphi_\beta = \text{math.radians}(\text{Latitude}_\beta)$
- $\lambda_\alpha = \text{math.radians}(\text{Longitude}_\alpha)$
- $\lambda_\beta = \text{math.radians}(\text{Longitude}_\alpha)$

Для расчета `sin` используйте функцию `math.sin()`, для расчета `cos` используйте функцию `math.cos()`, для расчета `arccos` используйте функцию `math.acos()`.

Расстояние в километрах вычисляется по формуле:

$$L = d \cdot R$$

Где $R = 6371$ км — средний радиус Земли.

Формулу для подсчета расстояния нам любезно дали, то есть можно просто ее реализовать, осталось понять, что такое `кофеен, булочных/пекарен и кондитерских`. Потому что «в Москве» не должно составлять проблем.

Чтобы перевести все данные сразу в радианы, воспользуемся не `math.radians`, а `np.radians`.

```
SHAD_phi = math.radians(37.589482)
SHAD_lam = math.radians(55.733705)
mask = (df['city'] == 'msk') & (df['category_1'].isin(['Кофейня', "Булочная, пекарня",
"Кондитерская"]) | df['category_2'].isin(['Кофейня', "Булочная, пекарня", "Кондитерская"]))
lat = np.radians(df['Latitude'])
lon = np.radians(df['Longitude'])
mask2 = (np.arccos(
    np.sin(lat) * math.sin(SHAD_phi) +
    np.cos(lat) * math.cos(SHAD_phi) * np.cos(lon - SHAD_lam)
) * 6371) <= 1.5
print((mask & mask2).sum())
# df.loc[mask & mask2, ['rating', 'city', 'category_1', 'category_2']]
```

38

A4. Яндекс.Карты

Объекты какой категории имеют наименьший средний по категории чек?

В качестве категории учитывать только category_1

Это довольно простая задача, нужно сгруппировать данные по `category_1` и посмотреть на среднее значение по полю `average_bill`, а после найти минимум:

```
print(df[df['average_bill'].notna()].groupby('category_1')['average_bill'].mean().idxmin())
```

Быстрое питание

A5. Яндекс.Карты

В какой категории больше всего объектов с наличием настольных игр(table_games)

В качестве категории учитывать только category_1

В данной задаче нужно найти среди `features` название `table_games`

```
df_with_games = df[df['features'].str.contains('table_games', na=False, case=False)].copy()

games_by_category = df_with_games['category_1'].value_counts()
games_by_category
```

category_1
Ресторан 171
Кафе 166
Кофейня 28
Быстрое питание 13
Булочная, пекарня 8
Пиццерия 7
Кондитерская 7
Столовая 7
Кофе с собой 6
Суши-бар 2
Name: count, dtype: int64

Таким образом ответом на это задание было бы `Ресторан`, если бы разработчики этого задания были бы чуть-чуть внимательнее.

B. Архив немой симфонии

В начале XXXI века в одном из европейских архивов цифрового искусства была обнаружен гигантский массив аудиофайлов — фрагменты утраченной симфонии неизвестного композитора. Музыка была полностью оцифрована, а каждый фрагмент представлен векторным описанием: эмбеддингом, извлечённым старой, но очень качественной нейросетью анализа звука.

Эта система создавалась десятилетия назад и использовалась для каталогизации музыки по трём музыкальным темам. Однако современные музыковеды выяснили: композитор изначально писал симфонию, состоящую из пяти фундаментальных музыкальных тем, а не трёх. Просто поздние редакторы архива упростили классификацию, сведя всё к трёхклассовой схеме, в которой 4 и 5 классы включены в первые три. При этом сами векторные представления содержат информацию о всех пяти темах.

К счастью, искусствоведы дополнительно обнаружили небольшую коллекцию музыкальных фрагментов в личных записях композитора. Эмбеддинги этих фрагментов помечены всеми пятью классами. Также были обнаружены музыкальные фрагменты, эмбеддинги которых не были классифицированы. Ваша задача — на основе имеющихся данных помочь искусствоведам классифицировать неразмеченные музыкальные фрагменты на пять классов.

Формат ввода

К задаче прикреплены файлы:

- **train_class123.csv** – обучающая выборка эмбеддингов, размеченных на 3 класса. Известно, что часть из эмбеддингов этой выборки принадлежит классам 4 и 5. Колонка `id` — уникальный `id` объекта, feat_i – i-я компонента эмбеддинга объекта, `label` – класс объекта.
- **train_class45.csv** – обучающая выборка эмбеддингов, размеченных на классы 4 и 5. Колонка `id` - уникальный `id` объекта, feat_i — i-я компонента эмбеддинга объекта, `label` – класс объекта.
- **test.csv** - тестовая выборка. Колонка id – идентификатор объекта. feat_i – i-я компонента эмбеддинга объекта
- **baseline.ipynb** – ноутбук с базовым решением задачи.
- **submission.csv** – пример решения, которое вам нужно отправить в тестирующую систему.

Формат вывода

Файл `submission.csv` должен содержать две колонки:

- `id` - идентификатор объекта из `test.csv`
- `label` - предсказанная вами целевая переменная

Система оценивания

Максимум за задачу — 100 баллов. Данные тестовой выборки разделены на публичную и приватную части. После отправки решения система показывает результат на публичной части. Окончательный результат после завершения контеста будет рассчитан по приватной части. После окончания этапа ваша метрика будет приведена к 100-балльной шкале по следующему правилу:

- результат baseline-решения оценивается в 0 баллов;
- результат авторского решения оценивается в 100 баллов;
- результаты между этими точками распределяются линейно.

В качестве метрики качества используется Micro F1, рассчитанное по всей матрице ошибок.

Как и всегда, сначала import библиотек. В данной задаче нам не нужно сильно много методов, потому что понятно, что основные методы — KNN, потому что похожесть аудиофайлов оценивается каким-то образом в расстоянии в 15-мерном пространстве.

```
import pandas as pd
import numpy as np
from catboost import CatBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

Читаем данные и масштабируем

```
df_noisy = pd.read_csv('train_class123.csv')
df_clean = pd.read_csv('train_class45.csv')
df_test = pd.read_csv('test.csv')

# Сдвиг меток для удобства (0..4)
# 1,2,3 -> 0,1,2
# 4,5   -> 3,4
df_noisy['label'] = df_noisy['label'] - 1
df_clean['label'] = df_clean['label'] - 1

feature_cols = [c for c in df_test.columns if c.startswith('feat_')]

X_noisy = df_noisy[feature_cols].values
y_noisy = df_noisy['label'].values

X_clean = df_clean[feature_cols].values
y_clean = df_clean['label'].values

X_test = df_test[feature_cols].values
```

```
scaler = StandardScaler()
# Обучаем скейлер на всем, чтобы пространство было единым
X_full = np.vstack([X_noisy, X_clean])
scaler.fit(X_full)

X_noisy_s = scaler.transform(X_noisy)
X_clean_s = scaler.transform(X_clean)
X_test_s = scaler.transform(X_test)
```

Так как данные второго файла корректны на все 100%, будем обучать KNN лишь по одному соседу, потому что в таких данных **явно** выражены все «кластеры» и неразберихи не произойдет.

```
# Мы используем KNN как классификатор, обученный только на классах 4 и 5 (id 3 и 4)
knn_model = KNeighborsClassifier(n_neighbors=1, p=2) # 1 сосед, Евклидово расстояние
knn_model.fit(X_clean_s, y_clean)
```

Теперь можно искать «потерянные» аудиозаписи (те, которые принадлежат не своему классу)

```
# Ищем для каждой шумной точки ближайшего соседа из чистых
# distances - расстояние до соседа
distances, _ = knn_model.kneighbors(X_noisy_s)
# preds - какой это класс (3 или 4)
```

```
preds = knn_model.predict(X_noisy_s)

# Чтобы понять, какое расстояние считать "близким", посмотрим,
# насколько близко точки чистых классов лежат друг к другу.
clean_dists, _ = knn_model.kneighbors(X_clean_s, n_neighbors=7)
# clean_dists[:, 1] - расстояние до ближайшего соседа (не считая себя)
threshold = np.percentile(clean_dists[:, 1], 90)

print(f"Рассчитанный порог расстояния: {threshold:.4f}")

y_corrected = y_noisy.copy()
count_corrections = 0

for i in range(len(y_noisy)):
    dist = distances[i][0]
    predicted_clean_label = preds[i]

    # Если точка из 1-го датасета лежит к точке из 2-го датасета так же близко,
    # как точки 2-го датасета лежат друг к другу -> мы верим, что это скрытый класс
    if dist < threshold:
        # Проверяем, отличается ли метка
        if y_corrected[i] != predicted_clean_label:
            y_corrected[i] = predicted_clean_label
            count_corrections += 1

print(f"Найдено и исправлено меток: {count_corrections}")
```

Поиск скрытых классов в шумной выборке...
Рассчитанный порог расстояния: 4.5366
Найдено и исправлено меток: 624
Теперь можно обучать модель 🐱boost.

```
# Склеиваем исправленные шумные + чистые
X_final = np.vstack([X_noisy, X_clean])
y_final = np.concatenate([y_corrected, y_clean])
final_model = CatBoostClassifier(
    iterations=1500,
    learning_rate=0.02,
    depth=6,
    loss_function='MultiClass',
    verbose=100,
    random_state=42,
    allow_writing_files=False,
    l2_leaf_reg=3
)
final_model.fit(X_final, y_final)
```

Такое решение набирает порядка **95** баллов. Существуют эвристики, которые позволяют поднять этот балл вплоть до **98**, но они не будут рассмотрены, просто потому что они

1. Неочевидны
2. Высосаны из пальца
3. Не несут учебного смысла

C. Прогноз температуры по метеостанциям

Никита увлекается машинным обучением и мечтает поехать на межнар по ИИ. Местная метеослужба обратилась к нему с просьбой помочь оценить возможности машинного обучения в долгосрочном прогнозе погоды. Они предоставили исторические данные о температуре с различных метеостанций и хотят понять, насколько точно современные методы ML могут предсказывать температуру. Никита уверен, что может построить модель предсказания температуры, которая продемонстрирует метеорологам потенциал машинного обучения для прогнозирования погоды.

Формат ввода

К задаче прикреплены файлы:

- `train.csv` – обучающая выборка. Колонка `temperature` – целевая переменная. Остальные колонки – признаки (`timestamp`, `station_id`, `humidity`, `wind`).
- `test.csv` – тестовая выборка. Колонка `ID` – идентификатор объекта. Остальные колонки – признаки (`timestamp`, `station_id`, `measure_type`).
- `baseline.ipynb` – ноутбук с базовым решением задачи.
- `submission.csv` – пример решения, которое вам нужно отправить в тестирующую систему.

Формат вывода

Файл `submission.csv` должен содержать две колонки:

- `ID` – идентификатор объекта из `test.csv`
- `temperature` – предсказанная вами целевая переменная

Система оценивания

Максимум за задачу — 100 баллов. Данные тестовой выборки разделены на публичную и приватную части. После отправки решения система показывает результат на публичной части. Окончательный результат после завершения контеста будет рассчитан по приватной части. После окончания этапа ваша метрика будет приведена к 100-балльной шкале по следующему правилу:

- результат baseline-решения оценивается в 0 баллов;
- результат авторского решения оценивается в 100 баллов;
- результаты между этими точками распределяются линейно.

Метрика в задаче представляет собой **MAE (Mean Absolute Error)** – среднюю абсолютную ошибку. MAE вычисляется по формуле:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \tilde{y}_i|$$

где:

- n - количество объектов в тестовой выборке
- y_i - истинное значение температуры для i-го объекта
- $\tilde{y}_i$ - предсказанное значение температуры для i-го объекта

Чем меньше значение MAE, тем лучше качество модели. MAE измеряется в тех же единицах, что и целевая переменная (в данном случае — градусы Цельсия), что делает её интерпретацию интуитивно понятной.

Для начала, как всегда import библиотек и чтение данных. Также полезно посмотреть данные:

```
import pandas as pd
import numpy as np
from catboost import CatBoostRegressor

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
submission = pd.read_csv('submission.csv')
```

Теперь будем чистить данные, так как очевидно, что есть выбросы. Какие данные точно будет неверными?

- Температура, которая слишком большая или очень маленькая, то есть: > 53 или < -60
- Влажность меньше 30 или больше 70. (Спойлер: таких в действительности нет)
- Температуру перепутали с влажностью, то есть `temperature` $\approx$ `humidity`

```
# Чистим данные
print("Примеры выбросов (temp ≈ humidity):")
outlier_examples = train[np.abs(train['temperature'] - train['humidity']) < 1.0]
print(outlier_examples.head(10))

# Способ 1: Убираем строки где temperature слишком близка к humidity
mask_bug = np.abs(train['temperature'] - train['humidity']) < 1.0
print(f"\nНайдено строк с багом (temp ≈ humidity): {mask_bug.sum()}")

# Способ 2: Убираем экстремальные температуры (физически невозможные для погоды)
mask_extreme = (train['temperature'] < -60) | (train['temperature'] > 53)
print(f"Найдено экстремальных значений: {mask_extreme.sum()}")

# Комбинируем маски
mask_outliers = mask_bug | mask_extreme
train = train[~mask_outliers].copy()
```

Примеры выбросов (temp ≈ humidity):

	timestamp	station_id	temperature	humidity	wind
46	2000-04-08 16:00:00	1	69.217277	69.822281	4.244093
7847	2003-10-30 20:00:00	1	31.505778	32.482166	1.273716
14392	2006-11-16 16:00:00	1	36.866894	36.010833	1.055764
14452	2006-11-26 16:00:00	1	38.000527	38.665965	1.004166
16421	2007-10-20 20:00:00	1	31.771824	31.088072	1.472223
29535	2013-10-14 12:00:00	1	31.096265	30.517991	1.613438
38369	2017-10-25 20:00:00	1	30.944841	31.717440	1.366924
49515	2022-11-26 12:00:00	1	37.971906	38.665965	1.004166
52891	1999-04-23 04:00:00	2	67.887853	68.614479	4.581891
54328	1999-12-18 16:00:00	2	46.335474	45.561570	1.097887

Найдено строк с багом (temp ≈ humidity): 30
Найдено экстремальных значений: 1004

Дальше давайте распарсим данные в более удобный формат и перестроим таблицу, чтобы в ней осталось только то, что нам непосредственно нужно.

```
# Парсим даты
train['dt'] = pd.to_datetime(train['timestamp'])
train['date_only'] = train['dt'].dt.date

# Группируем по (Дата, Станция) и считаем min и max за сутки
daily_agg = train.groupby(['date_only', 'station_id'])['temperature'].agg(['min',
'max']).reset_index()

# Сейчас у нас колонки: [date, station, min, max].
# А нам нужно, чтобы было как в тесте: одна строка - одно измерение (либо min, либо max).
# Используем pd.melt
train_transformed = pd.melt(
    daily_agg,
    id_vars=['date_only', 'station_id'],
    value_vars=['min', 'max'],
    var_name='measure_type',
    value_name='temperature'
)

train_transformed.sample(5)
```

	date_only	station_id	measure_type	temperature
8306	2005-07-17	3	min	-10.000000
34209	2019-09-27	4	min	10.596067
47403	2003-07-14	4	max	24.611569
12232	2007-09-10	4	min	9.438218
58906	2009-11-12	5	max	1.392465

Какие признаки могут влиять на погоду? Месяц, время дня, день в году, вероятно, год. Ну и конечно наш самый главный параметр: `min/max`.

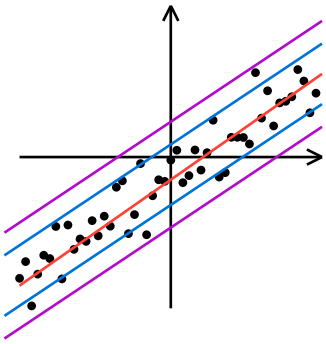
```
# Добавим timestamp в train_transformed для удобства генерации фичей
train_transformed['timestamp'] = pd.to_datetime(train_transformed['date_only'])

def create_features(df): # чтобы не писать два раза
    df_feat = df.copy()

    df_feat['timestamp'] = pd.to_datetime(df_feat['timestamp'])
    df_feat['year'] = df_feat['timestamp'].dt.year
    df_feat['month'] = df_feat['timestamp'].dt.month
    df_feat['day_of_year'] = df_feat['timestamp'].dt.dayofyear
    # 0 - min, 1 - max
    df_feat['is_max'] = (df_feat['measure_type'] == 'max').astype(int)

    return df_feat


X_train_df = create_features(train_transformed)
X_test_df = create_features(test)

# Признаки для модели
features = ['station_id', 'year', 'month', 'day_of_year', 'is_max']
target = 'temperature'
```

Теперь применим любимую модель, 🐱boost, не забыв указать категориальные признаки

```
cat_features = ['station_id']

model = CatBoostRegressor(
    iterations=2000,
    learning_rate=0.03,
    depth=6,
    loss_function='MAE', # Оптимизируем MAE, как в задаче
    verbose=200,
    random_seed=42,
    cat_features=cat_features
)

model.fit(X_train_df[features], X_train_df[target])
```

Такое решение набирает не менее **92.3** балла. Теперь время использовать всю мощь ИИ — **магию**. Здесь начинается случайный лес случайного подбора гиперпараметров и недоказываемых эвристик.

Замечаем, что мы в модель пытаемся скормить что-то наподобие такой картинки:



Формально, мы хотим получить в ответе две фиолетовые прямые, но в среднем мы получаем что-то ближе к красной прямой (синие прямые). Просто потому что так устроены модели, они хорошо ищут среднее, а края нет. Таким образом можно начать подбирать параметры, что мы будем «прибавлять к синим прямым, чтобы они стали фиолетовыми».

Теперь можно заметить, что так как наша оригинальная программа имеет MAE ≈ 1.73, из этого можно сделать вывод, что мы либо перебираем с температурой на 0.73, либо недобираем. Переберем два этих случая, учитывая, что если `min` надо уменьшать в среднем, а `max` увеличивать.

Теперь, можно получить, если увеличивать `max` и уменьшать `min` примерно оптимальные параметры будут:

```
predictions = model.predict(X_test_df[features])

submission['temperature'] = predictions
is_max_mask = X_test_df['is_max'] == 1
is_min_mask = X_test_df['is_max'] == 0
submission.loc[is_max_mask, 'temperature'] += 2.0 # Олимпиадная хитрость
submission.loc[is_min_mask, 'temperature'] -= 1.20
submission.to_csv('submission_cheat.csv', index=False)
```

То есть надо к максимуму прибавлять ~ 2.0, а из минимума вычитать ~ 1.2.

Такое **эвристическое** решение получает **99.94** балла.

D. Классификация на плоскости

У вас есть две таблицы в формате CSV.

- **train.csv**: столбцы `id`, `x`, `y`, `label` – это обучающие примеры с известным классом.
- **test.csv**: столбцы `id`, `x`, `y` — это объекты без меток.

Задача

По координатам (x, y) научиться присваивать один из восьми классов $\{1, 2, 3, 4, 5, 6, 7, 8\}$ и предсказать метку `label` для каждой строки из `test.csv`, используя примеры из `train.csv`.

Для вашего удобства вам дан **baseline.ipynb**.

Формат ввода

`train.csv` содержит следующую информацию:

- `x` — первая координата точки
- `y` — вторая координата точки
- `label` — целевая метка класса (категориальный признак, 8 значений)

`test.csv` содержит следующую информацию:

- `id` — уникальный идентификатор точки
- `x` — первая координата точки
- `y` — вторая координата точки

Формат вывода

Необходимо сдать csv файл с двумя колонками:

- `id` — идентификатор точки из `test.csv`;
- `label` — предсказанный вами класс для объекта из `test.csv`.

Система оценивания

Максимум за задачу — 100 баллов. Данные тестовой выборки разделены на публичную и приватную части. После отправки решения система показывает результат на публичной части. Окончательный результат после завершения контеста будет рассчитан по приватной части. После окончания этапа ваша метрика будет приведена к 100-балльной шкале по следующему правилу:

- результат baseline-решения оценивается в 0 баллов;
- результат авторского решения оценивается в 100 баллов;
- результаты между этими точками распределяются линейно.

Метрика:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} 1[\hat{y}_i = y_i]$$

Разберем несколько разных подходов набирания баллов в данной задаче:

1. Самый тупой метод решения — голый KNN:

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

X_train = train_df[['x', 'y']].values
y_train = train_df['label'].values
X_test = test_df[['x', 'y']].values

knn = KNeighborsClassifier(
    n_neighbors=3,
    p=2, #Евклидово расстояние
)
knn.fit(X_train, y_train)
test_df["label"] = knn.predict(X_test)
sub = test_df[['id', 'label']]
sub.to_csv('submission.csv', index=False)
```

Такое решение набирает, внимание, **39.62** балла. Это **надо** написать.

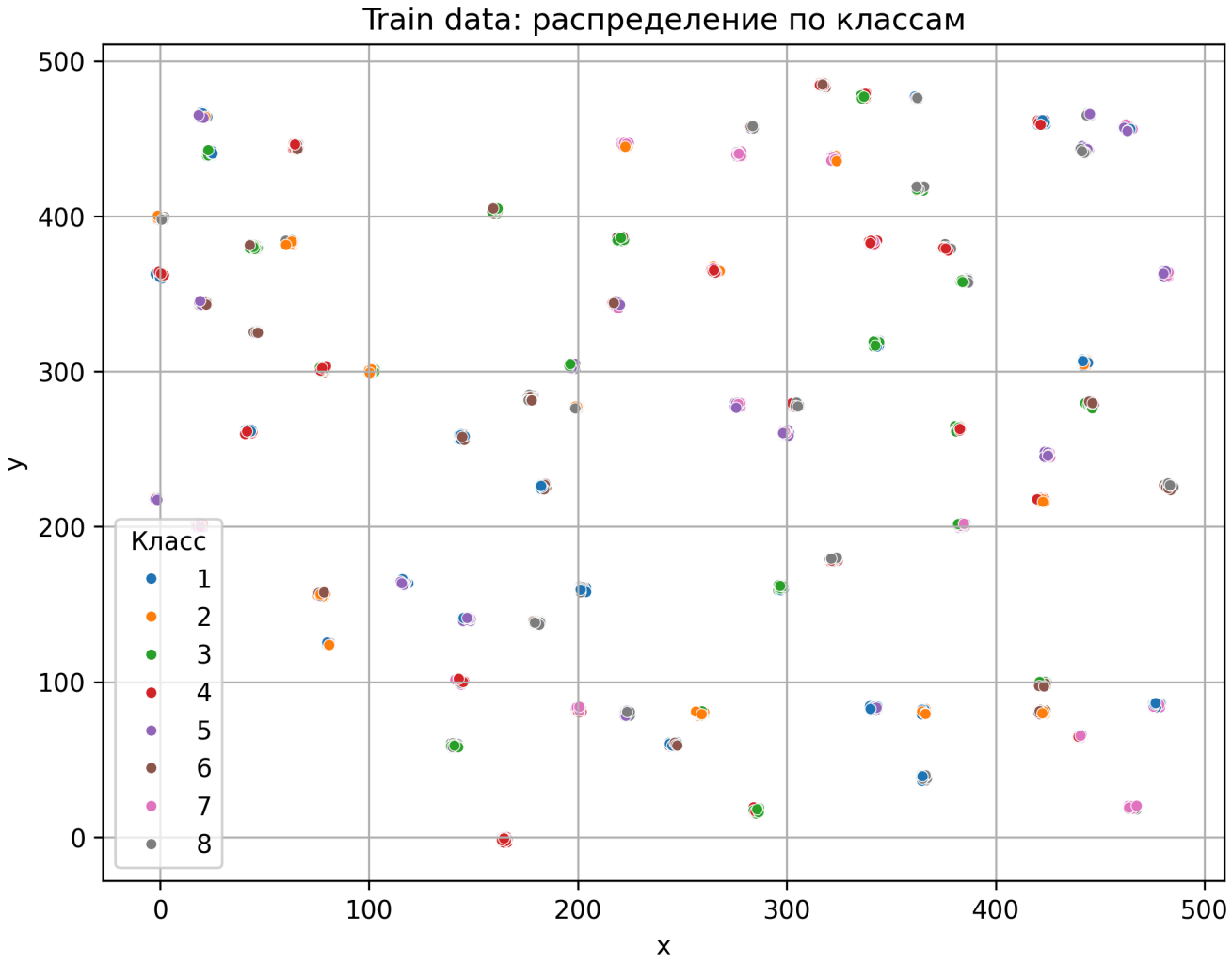
2. Чуть более тупой метод — голый KNN с одним соседом.

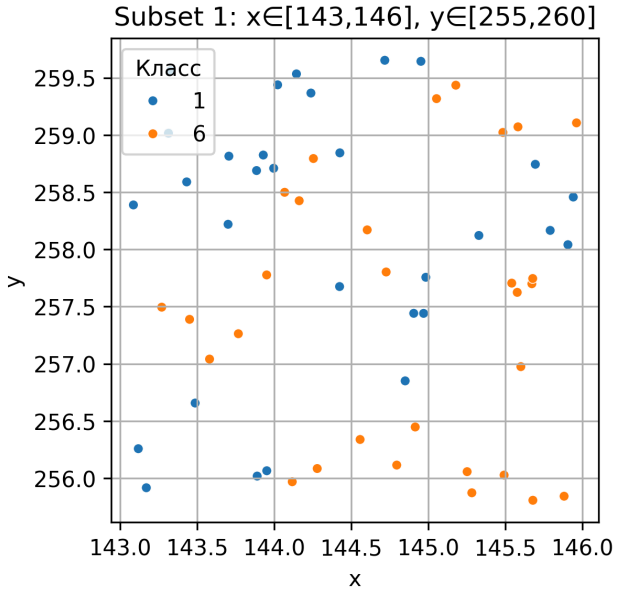
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

X_train = train_df[['x', 'y']].values
y_train = train_df['label'].values
X_test = test_df[['x', 'y']].values

knn = KNeighborsClassifier(
    n_neighbors=1,
    p=2, #Евклидово расстояние
)
knn.fit(X_train, y_train)
test_df["label"] = knn.predict(X_test)
sub = test_df[['id', 'label']]
sub.to_csv('submission.csv', index=False)
```

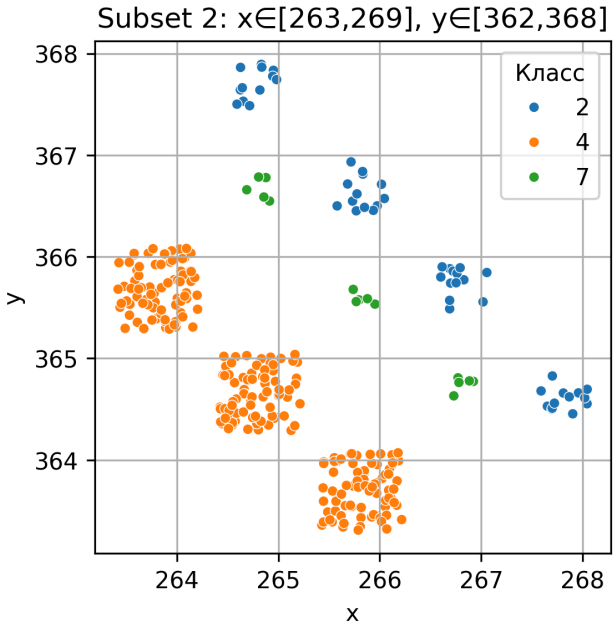
Такое решение набирает, внимание, **42.09** балла. До такой тупости нужно додуматься, берите на вооружение

А ничего более умного не набирает больше **42** баллов) Веселая задача, согласитесь.

Давайте теперь визуализируем данные, посмотрим, что от нас спрятали разработчики этой задачи:







Теперь понятно, почему так хорошо работает KNN. Довольно очевидно, что нам здесь понадобится KNN. Но когда его использовать? Предлагается использовать когда в круге вокруг очередной точки

не больше 2 разных типов точек. А что делать в других случаях? Воспользуемся логистической регрессией.

Почему применима логистическая регрессия? Если в окрестности точки много разных классов, KNN не помогает, значит можно воспользоваться вероятностным подходом определения класса.

Решение:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

X_train = train_df[['x', 'y']].values
y_train = train_df['label'].values
X_test = test_df[['x', 'y']].values

def predict(pt, X_tr, y_tr, radius=3, C_val=1):
    dists = np.sqrt(np.sum((X_tr - pt)**2, axis=1))
    mask = dists <= radius
    local_X, local_y = X_tr[mask], y_tr[mask]

    if len(local_X) == 0:
        return 1

    n_classes = len(np.unique(local_y))
    if n_classes == 1: # однозначно определяется класс точки
        return local_y[0]

    if n_classes >= 3:
        model = LogisticRegression(C=C_val, max_iter=1000, random_state=42)
    else:  # 2 класса -> KNN k=3 или KNN k = 1
        model = KNeighborsClassifier(n_neighbors=3)

    model.fit(local_X, local_y)
    return model.predict([pt])[0]

# Генерация
preds = [predict(pt, X_train, y_train, radius=3, C_val=1) for pt in X_test]

# Сохранение
sub = pd.DataFrame({'id': test_df['id'], 'label': preds})
sub.to_csv('submission.csv', index=False)
```

Такое решение наберет целых **100** баллов. Но получить по этой задаче хороший балл от **25**, применив почти любой бустинг, просто логистическую регрессию или другой алгоритм