Архитектура проекта

# Определение этапов:

1. ввод данных
2. разработка (предобработка + обучение базовых + композиция)
3. вывод данных

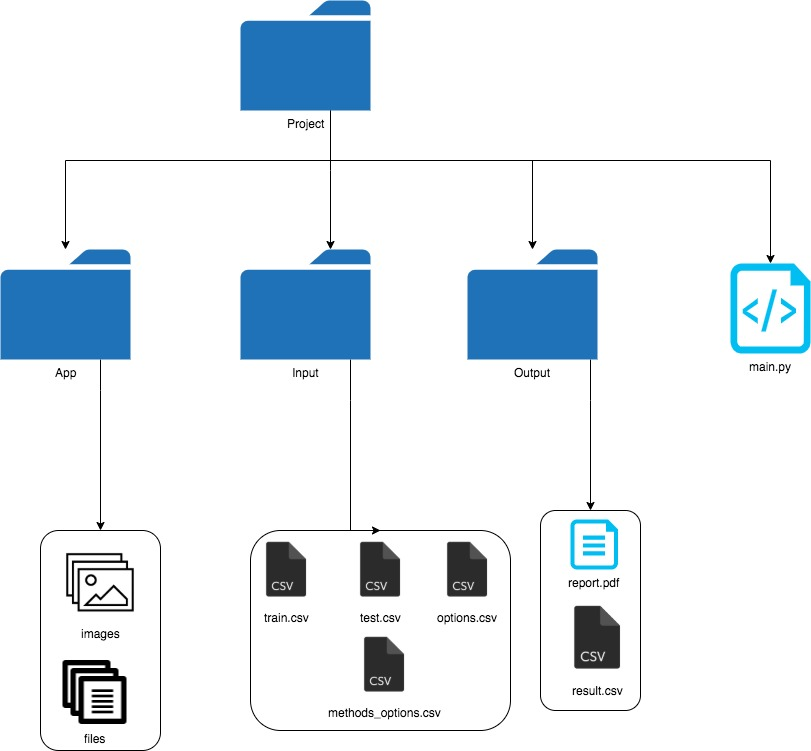
Делитесь по группам в соответствии с этапами и подэтапами.

# Архитектура файлового пространства:

В корневой папке проекта лежат 3 папки:

1. **App** - файлы для выполнения программы (там будут храниться обученные модели в формате pickle (ИЗУЧИТЕ ЕГО), изображения графиков, прочая хрень)
2. **Input** - 3 входных файла:
3. файл обучающей выборки **train.csv** с размеченными данными (признаки+столбец 0/1)
4. файл контрольной выборки **test.csv** с неразмеченными данными (только признаки)
5. файл с **общими** для любого случая параметрами (подробнее про общее/частное см. ниже) **options.csv**

3. **Output** - папка для выгрузки результатов работы программы (там будет pdf-отчет и файл контрольной выборки с дополнительным столбцом - результатом классификации)



# Общая информация для всех групп:

Для поддержания единообразия работы введем следующие объекты:

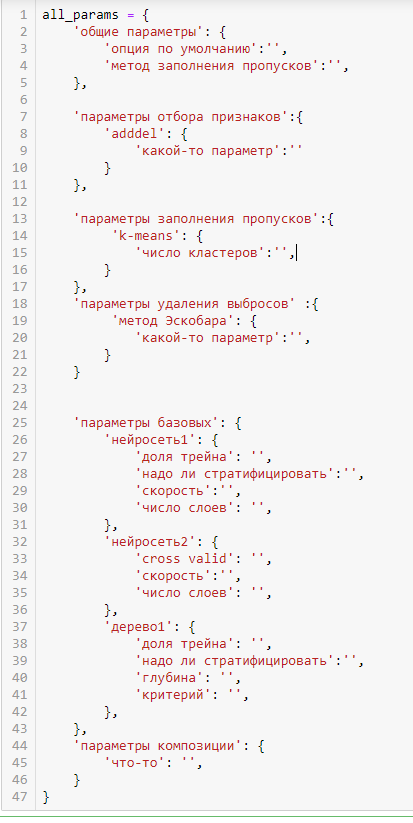
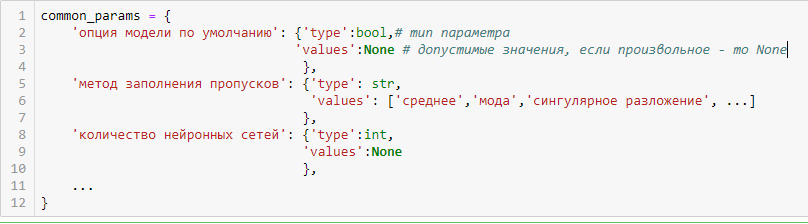
1. Словарь общих параметров common\_params для любого случая. В этом объекте находятся ,например следующие параметры: модель по умолчанию или нет(boolean), метод заполнения пропусков(str), метод удаления выбросов(str), тип композиции(БЕЗ ПАРАМЕТРОВ)(str), к-во различных нейронных сетей(int), к-во различных деревьев решений(int)...(если добавятся другие типы базовых алгоритмов, то делаем аналогично).

Примечание, надо сказать Домашовой: Общее количество моделей - вроде как лишний параметр, если у нас есть инфа про количество моделей каждого алгоритма

Пример этого словаря:

**Примечание:** многие примеры параметров в гуглдоке взяты от балды, правильные смотрите в ТЗ, на рисунках главное показать структуру

1. Словарь параметров methods\_params для каждого ТИПА базового алгоритма(нейросеть, дерево решений, …)+для каждого конкретного метода удаления пропусков, удаления выбросов и отбора признаков , где ключ - название алгоритма, а значение - как весь словарь из предыдущего пункта, только параметры не общие, а ВСЕВОЗМОЖНЫЕ характерные для каждого конкретного алгоритма (смотрите в ТЗ, стр 5 и далее). ЕСЛИ У МЕТОДА НЕТ ДОППАРАМЕТРОВ, ОН НЕ ВКЛЮЧАЕТСЯ В СЛОВАРЬ. Возможно, есть смысл при составлении methods\_params опираться на названия из sklearn.
2. Словарь, содержащий параметры ВСЕХ моделей (общие параметры, параметры для удалений пропусков/выбросов/отбора признаков, параметры базовых алгоритмов, параметры композиции). Позволяет “удобно” передавать параметры с одного этапа на другой. Имеет следующую структуру:



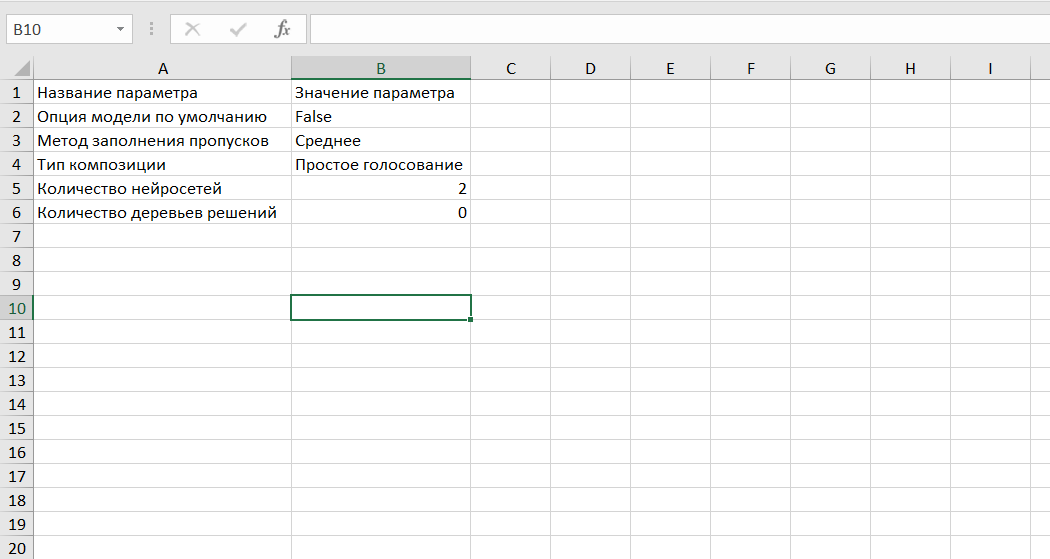
Первые два словаря существуют изначально, третий должен сформироваться после этапа ввода.

# Этапы:

## Ввод

На вход попадают три файла, лежащие в папке input - **train.csv, test.csv, options.csv** ( описание **train.csv** и **test.csv** есть ранее).

**Описание options:** первый столбец - название параметров (это ключи из common\_params). Второй столбец - одно из допустимых значений из common\_params (если в common\_params по параметру по ключу values написан None, то любое значение типа type). Пример файла:



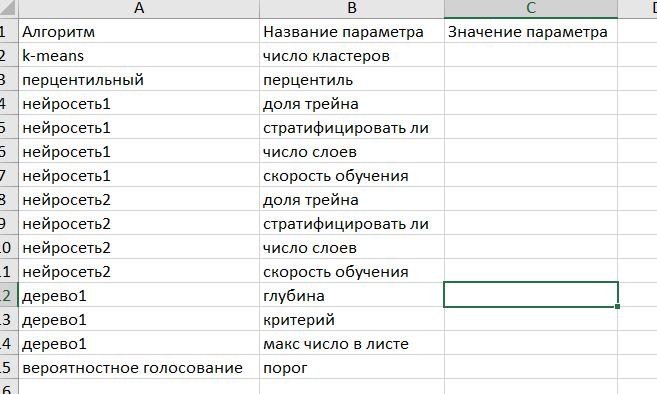
Примечание: если тип алгоритма не используется, то пишем 0 (считаем, что количество разных типов алгоритмов ограничено - пока только нейросети и деревья решений).

Дальнейший ход работы:

читаем файл options.csv, с помощью словаря common\_params проверяем корректность заданных параметров (надо сделать проверку на дурака, что все параметры заполнены и заполнены верно).

Далее надо сформировать файл methods\_options.csv, в котором мы потребуем от пользователя параметры для каждого базового алгоритма+алгоритмов удаления пропусков/выбросов/отбора признаков , указанных в options.csv. + параметры композиции (ЕСЛИ ЭТИ ПАРАМЕТРЫ ВООБЩЕ ЕСТЬ). Список параметров для каждого базового алгоритма и методов предобработки мы получаем из словаря **methods\_params**. Файл **methods\_params.csv** должен быть СГЕНЕРИРОВАН теми разрабами , которые делают ввод.

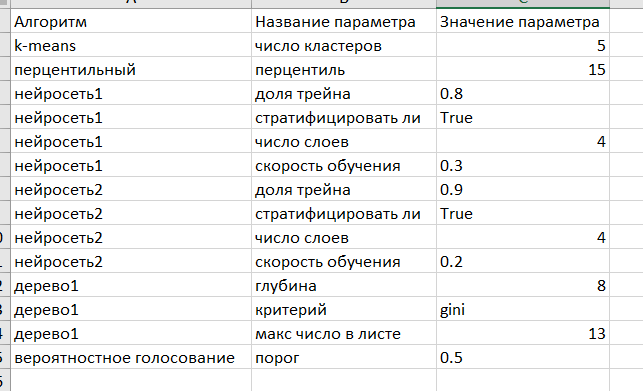
Пример такого файла:



**Примечание:** в первом столбце перемешаны все названия алгоритмов. Желательно, чтобы шло по порядку - заполнение пропусков->удаление выбросов->отбор признаков->базовые алгоритмы->композиция. Если будет допколонка с названием этапа, вообще супер.

Этот файл генерируется программой и сохраняется в папке **input**, а программа встает на паузу (метод os.system("pause") работает в .py, но не работает в юпитере, можно поставить input()). После того как пользователь заполнил СВОИМИ РУКАМИ файл **methods\_params.csv**, он дает команду программе продолжить (для паузы - библиотека os, метод pause).

Пример заполненного файла:



С помощью словаря **methods\_params** проверяем корректность заданных параметров (надо сделать проверку на дурака, что все параметры заполнены и заполнены верно)

После получения всех параметров (общих и для каждого алгоритма отдельно) формируется словарь всех параметров **all\_params** (см. выше его описание).

## Разработка

На входе есть словарь all\_params и два файла train.csv и test.csv (необработанные).

### Предобработка

#### Удаление пропусков (есть готовые решения)

**На входе**: файл (датафрейм) **train.csv** БЕЗ УЧЕТА СТОЛБЦА С МЕТКАМИ КЛАССОВ и **test.csv** соответствующая строка из словаря all\_params.

По строке выбирается метод, применяется к датафрейму,

**На выходе**: очищенный датафрейм.

Как сделать наиболее гибко (создавать одну главную функцию с if метод==то-то, выполнить такую-то функцию или как-то по-другому, решайте сами).

#### Удаление выбросов (есть готовые решения) - наверное теперь участвует только датафрейм от train.csv

Аналогично удалению пропусков.

#### Отбор признаков - есть готовые решения - на выходе усеченный датафрейм.

### Обучение базовых алгоритмов (есть готовые решения, или у Домашовой или в библиотеках питона).

### для каждого базового алгоритма делаем:

#### Кросс-валидация (есть готовые решения)

**На входе:** датафрейм, соответсвующая строка из all\_params.

**На выходе:** два датафрейма - один будет служить непосредственно для обучения, второй для проверки качества.

Просто train\_test\_split, единственное, посмотреть stratify или нет

#### Обучение базового алгоритма (есть готовые решения, желательно из библиотек питона - главное чтоб был метод обучения (fit) и предсказания (predict))

**На входе**: датафрейм и соответствующие данные из словаря all\_params.

Каждая из моделей, перечисленных в all\_params, получает свои параметры из all\_params. Идет **обучение модели на датафрейме для обучения**. Обученная модель сохраняется в файле формата .pkl (pickle) в папке app/models. На **датафрейме для предсказаний делается предсказывание** смотрится качество обучения.. Правила называния файлов придумываете сами.

На выходе: два словаря (первый - мб датафрейм, решайте сами):

1. Ключ - идентификатор объекта, значение - ( словарь, где ключ - название алгоритма (напр, нейросеть3), значение - 0 или 1 (или вероятность отнесения к классу, лучше даже ее)
2. Ключ - название алгоритма, значение - (словарь, где ключ - название метрики, значение - либо число, либо матрица, либо график (в каком-то формате, чтоб потом можно было открыть и посмотреть)

Второй словарь надо как-то сохранить.

### Композиция (пока простое голосование, дальше посмотрим).

**На входе:** параметры композиции, результаты первого словаря из предыдущего пункта.

Идет простое голосование

**На выходе:** словарь (мб датафрейм), где ключи - id объектов, значения - 0 или 1.

Метрики, таблица, мб графики (вроде как их уже не получится)

### Предсказание

Для теста делаем предсказание - выгружаем каждую из обученных моделей из app/models/, сохраненных в pkl, делаем предсказание, в датафрейм добавляем столбец с предсказаниями.

## Вывод

Надо сформировать два файла:

1. отчет в формате .pdf (кто это делает, спрашиваете у Домашовой, что туда надо) - на основе того, что есть в папке app ( или осталось прям в оперативке)
2. Файл test.csv с предсказанным столбцом.

# 

# <https://github.com/MisterMaks/classification-project-5-course>