МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

«Data Science»

Слушатель Чувакова Любовь Николаевна

Москва, 2023 год.

Содержание

Введение

Тема данной работы - Кластеризация клеток периферической крови от здорового донора после single-cell RNA sequencing.

Single-cell RNA sequencing (scRNA-seq) – относительно новый метод секвенирования РНК, который активно внедряется в научную практику. Этот метод позволяет получать данные от отдельных клеток различной этиологии, оценить, сколько и каких уже известных клеток было в образце, пути дифференциации клеток, а также возможные функции клеток в их микроокружении. Помимо этого, scRNA-seq позволяет выявить, выделить и проанализировать новые популяции клеток. Чтобы выделить популяции клеток, необходимо их кластеризовать, что уже успешно реализуется с помощью методов машинного обучения.

В данной работе использован датасет, полученный в результате scRNA-seq образца периферической крови, и после его картирования на референсный геном. Исходный датасет подвергнут препроцессингу, в виде удаления пустых значений (пустых капель), капель с погибшими клетками (с высоким содержанием митохондриальных генов), дуплетов, нормализации. Проведен кластерный анализ разными методами машинного обучения, для определения количества кластеров популяций клеток крови, с возможностью дальнейшего более глубокого анализа какого-либо кластера.

1. Аналитическая часть.

Полный цикл scRNA-Seq, от пробоподготовки до конечной кластеризации обычно включает следующие этапы: выделение отдельной клетки, экстракция и амплификация нуклеиновых кислот, подготовка библиотеки секвенирования, секвенирование и анализ данных (рис.1)

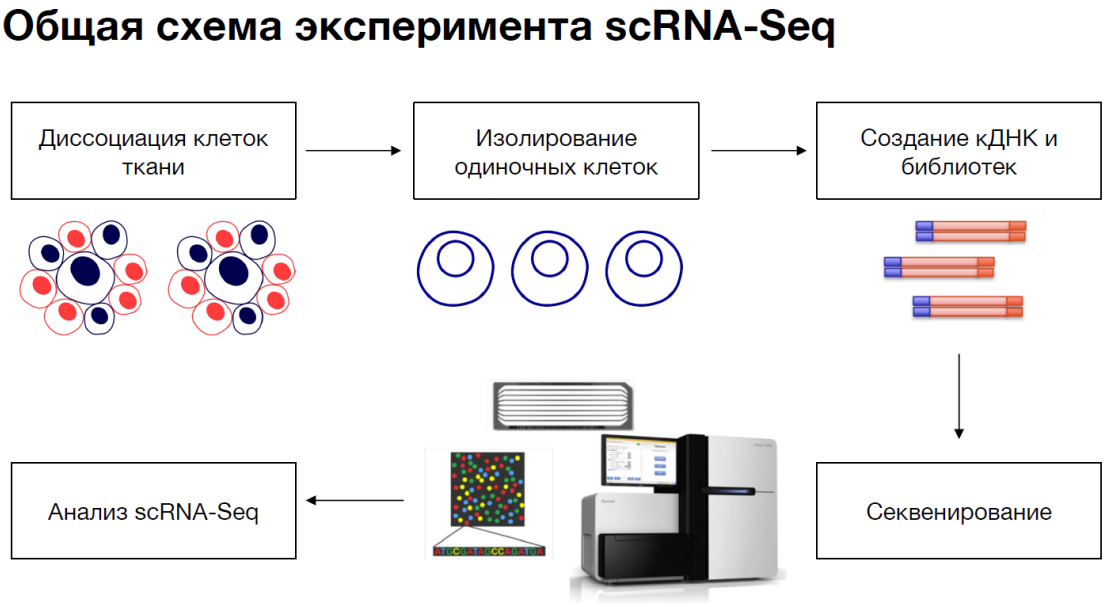


Рис 1. Общая схема эксперимента

Первым шагом метода является разделение и инкапсуляция каждой клетки и подготовка библиотеки. Клетки инкапсулируются в гелевые шарики в эмульсии (GEM) (рис. 2).

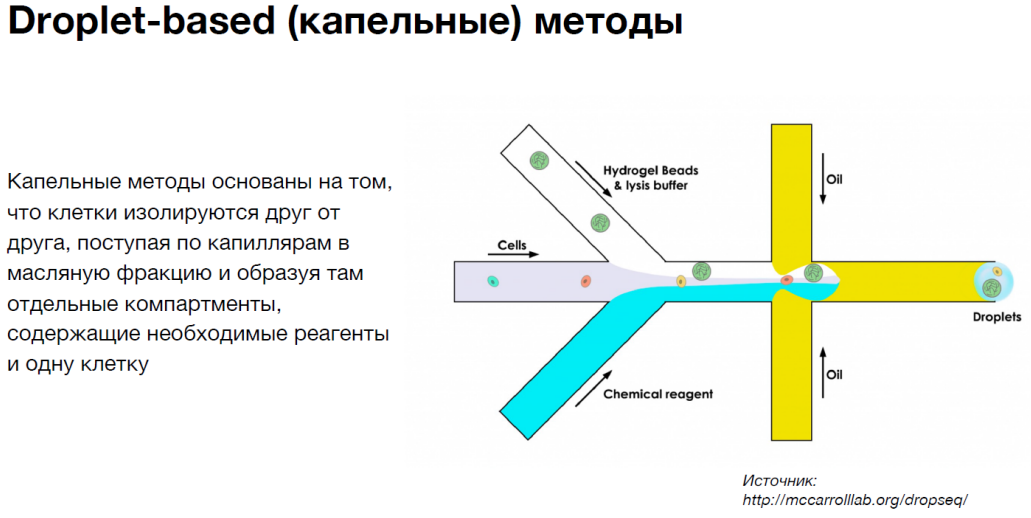


Рис. 2. Изоляция диссоциированных клеток в дроплеты

Каждый функциональный GEM содержит одну ячейку, одну гелевую бусину и реагенты RT. На гелевой бусине связываются олигонуклеотиды, состоящие из 4 отдельных частей: праймер для ПЦР (необходим для секвенирования); 10X олигонуклеотиды со штрих-кодом; последовательность уникального молекулярного идентификатора (UMI); Последовательность PolydT (которая позволяет захватывать полиаденилированные молекулы мРНК) (рис. 3).

Рис. 3. Схема навешивания баркодов и UMI

Внутри каждой реакционной везикулы GEM одна клетка лизируется и подвергается обратной транскрипции. кДНК из одной и той же клетки идентифицируются благодаря общему штрих-коду 10X. Кроме того, количество UMI выражает уровень экспрессии генов, и его анализ позволяет обнаруживать высоковариабельные гены. Эти данные часто используются

либо для классификации клеточных фенотипов, либо для идентификации новых субпопуляций.

Завершающим этапом платформы является секвенирование. Созданные библиотеки можно напрямую использовать для секвенирования всего транскриптома одной клетки или рабочих процессов целевого секвенирования.

В дальнейшем осуществляется непосредственно биоинформатический анализ данных. Из секвенатора выгружаются данные в формате .fastq. Выравнивание и подсчет экспрессии генов на клетку выполняется программой-картировщиком. Золотым стандартом является программа Cell Ranger. Ее достоинством является широкий спектр функций, начиная от автоматичского определения координат координаты баркода / UMI в прочтениях, работает с разными модификациями scRNA-Seq, с помощью флагов можно провести препроцессинг данных. Однако, он очень требовательный к ресурсам (1 Тб дискового пространства, 128 Гб RAM, 16 ядер) и очень долго работает (один образец может рассчитываться 12 часов).

В простейшем случае аутпут содержит 4 файла:

1. raw_feature_bc_matrix.tar.gz — матрица со всеми “клетками” из датасета (в дальнейшем и будет анализироваться)

a. barcodes.tsv.gz — названия клеток (баркоды)

b. features.tsv.gz — названия и id генов

c. matrix.mtx.gz — непосредственно матрица экспрессии в sparce-виде

2. filtered_feature_bc_matrix.tar.gz — то же, что и пункт 1, только с уже отфильтрованными клетками (Cell Ranger фильтрует очень неплохо)

a. barcodes.tsv.gz

b. features.tsv.gz

c. matrix.mtx.gz

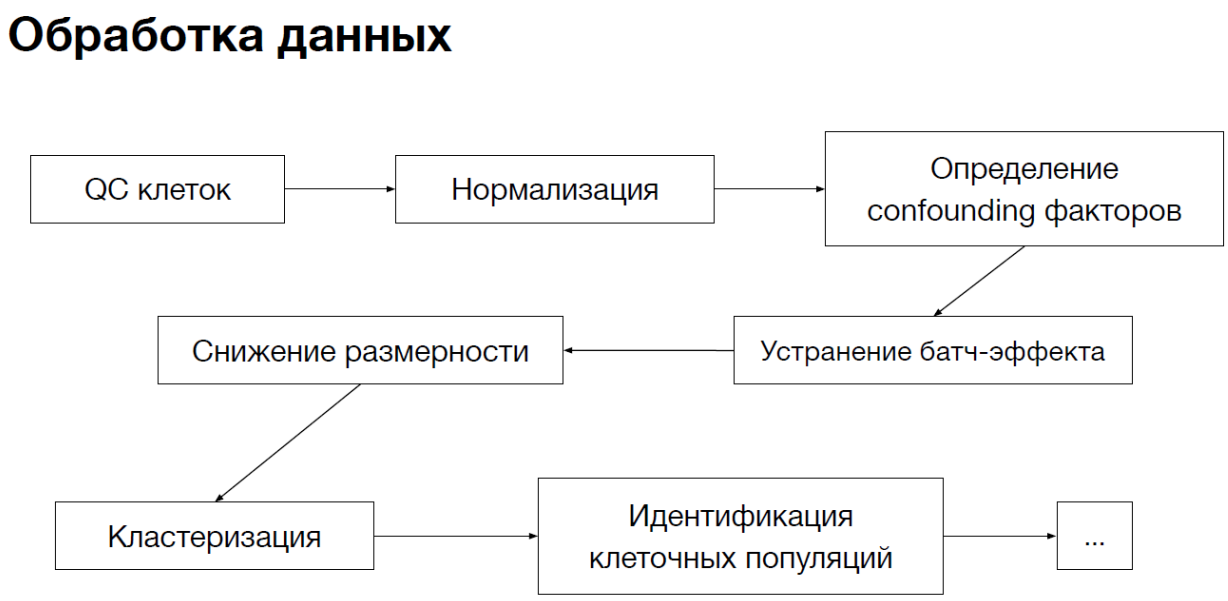
3. metrics_summary.csv — таблица с основными метриками

4. web_summary.html — графический веб-отчёт о качестве выравнивания и т. п.

1.1. Постановка задачи

В папке outs содержатся файлы после отработки CellRanger. Датасет raw_feature_bc_matrix.tar.gz находится на google disk, так как Github не позволяет подгружать большие датасеты с весом, более 25 мб непосредственно в репозиторий и более 100 мб через командную строку.

Для того, чтобы кластеризовать данные после scRNA-Seq и в дальнейшем выбрать и проанализировать интересующие кластеры клеток периферической крови, необходимо сначала провести препроцессинг данных, который включает в себя QC клеток (удаление пустых капель (аналог удаления строк с NA), капель с погибшими клетками (где большое количество митохондриальных генов), дуплетов, нормализацию данных, возможные дополнительные этапы и затем непосредственно кластеризация клеточных популяций (рис.4).



Данные представлены в формате AnnData – Annotated data – это так называемый контейнерный формат данных, где данные взаимосвязаны между собой, но представлены в нескольких таблицах (рис.5).

Class **anndata.AnnData** *(X=None, obs=None, var=None, uns=None, obsm=None, varm=None, layers=None, raw=None, dtype=None, shape=None, filename=None, filemode=None, asview=False, *, obsp=None, varp=None, oidx=None, vidx=None)*

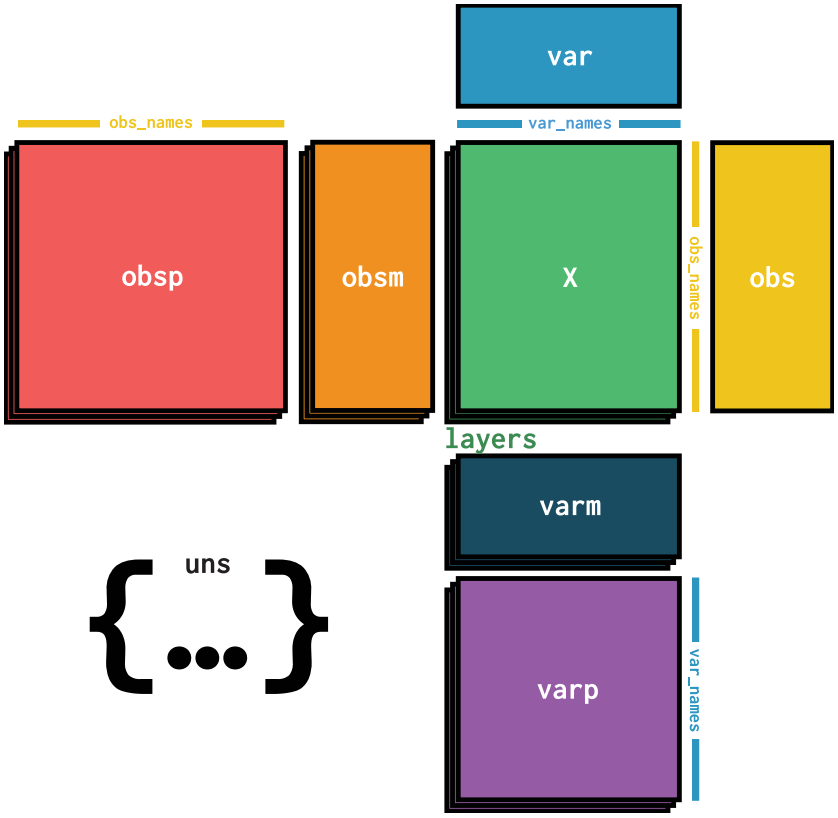


Рис. 5. Формат annData

AnnData хранит в себе матрицу X вместе с аннотациями исследований obs (obsm, obsp), вариативными данными var (varm, varp), и неструктурированными аннотациями uns (https://anndata.readthedocs.io/en/latest/#). Данный формат возможно перевести в data frame, однако часть данных будет потеряна, что не всегда может положительно сказаться на кластерном анализе.

Основной объект AnnData — это матрица, в которой находятся данные по числу UMI каждого гена на каждую клетку. В исходном датасете очень много клеток (больше трёх миллионов). В нашем случае эта матрица имеет размерность число клеток × число генов, то есть 3069478 × 36601. Эта матрица пока находится в sparse-виде. Это такой формат, который подразумевает, что нулей в матрице сильно больше, чем ненулевых элементов — тогда в явном виде можно хранить только координаты ненулевых элементов, это сильно экономит память.

1.2. Описание используемых методов

1.2.1. Scanpy – Single-Cell Analysis in Python

Scanpy — это масштабируемый инструментарий для анализа данных об экспрессии генов в отдельных клетках, созданный совместно с anndata. Он включает в себя предварительную обработку, визуализацию, кластеризацию, вывод траектории и тестирование дифференциальных выражений. Реализация на основе Python эффективно работает с наборами данных, содержащими более миллиона ячеек (Wolf *et al.* (2018), https://scanpy-tutorials.readthedocs.io/en/latest/index.html)

1.2.2. Scrublet

Scrublet используется для удаления капель, где оказалось 2 и более клеток. С такой ячейки приходит смешанная информация, которую затем анализировать не представляется возможным, поэтому данной утилитой строки с дублетами удаляются (рис. 6).

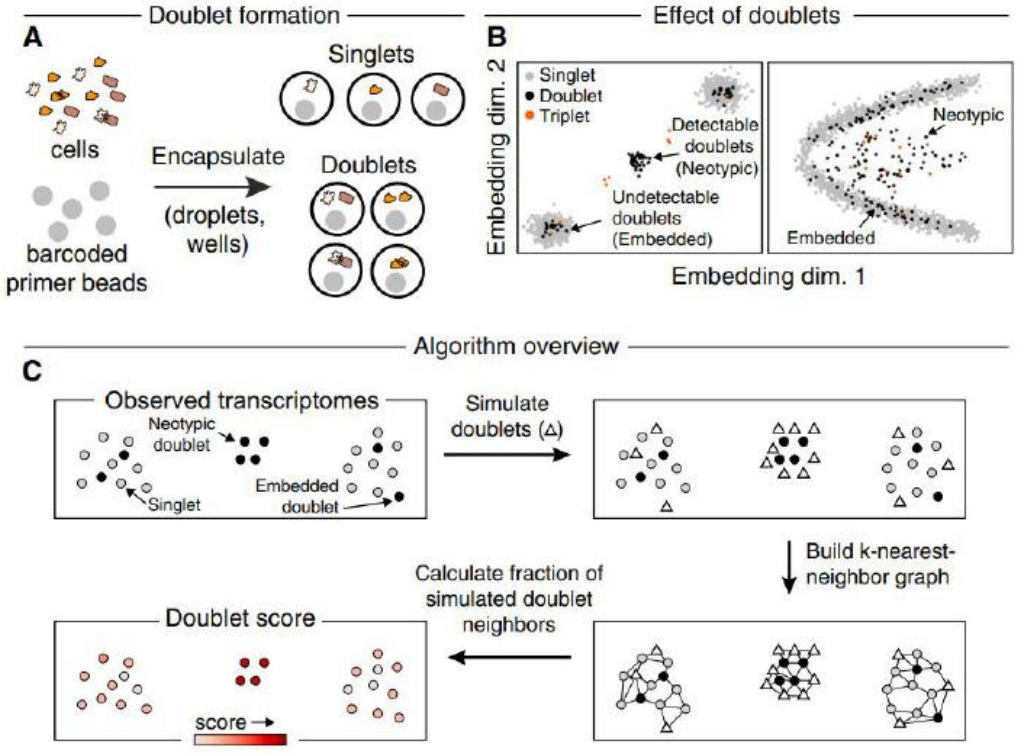


Рис. 6. Принцип работы Scrublet

Подход достаточно простой, сначала на строках генерируются фантомные дублеты и с ними сравниваются другие строки, и где оказывается похожее распределение по баркодам и UMI, те строки помечаются как дублеты и в дальнейшем из датасета удаляются.

1.2.3. PCA

Метод главных компонент (PCA) - один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации, а также снизить шум в данных, который может помешать адекватной кластеризации данных.

1.2.4. Метод k-ближайших соседей

Метод k-ближайших соседей (k Nearest Neighbors) - метод классификации, который адаптирован для регрессии. Суть метода проста: посмотри на соседей вокруг, какие из них преобладают, таковым ты и являешься. Для реализации метода необходима метрика расстояния между объектами. Используется, например, эвклидово расстояние для количественных признаков или расстояние Хэмминга для категориальных.

1.2.5. tSNE

tSNE - это алгоритм машинного обучения для визуализации, является техникой нелинейного снижения размерности, хорошо подходящей для вложения данных высокой размерности для визуализации в пространство низкой размерности (двух- или трехмерное). В частности, метод моделирует каждый объект высокой размерности двух- или трёхмерной точкой таким образом, что похожие объекты моделируются близко расположенными точками, а непохожие точки моделируются с большой вероятностью точками, далеко друг от друга отстоящими.

1.2.6. Leiden

Алгоритм Лейдена использует идею ускорения локального перемещения узлов и идею перемещения узлов к случайным соседям (Traag, et. al, 2019).

1.2.7. UMAP

Uniform Manifold Approximation and Projection (UMAP) — алгоритм машинного обучения, выполняющий нелинейное снижение размерности, похожий на t-SNE, но с более сильным математическим обоснованием.

При снижении размерности UMAP сначала выполняет построение взвешенного графа, соединяя ребрами только те объекты, которые являются ближайшими соседями. Множество из ребер графа — это нечёткое множество с функцией принадлежности, она определяется как вероятность существования ребра между двумя вершинами. Затем алгоритм создает граф в низкоразмерном пространстве и приближает его к исходному.

1.2.8. KMeans

Kmeans – метод, который разбивает множество элементов векторного пространства на заранее известное число кластеров k. Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение на точках каждого кластера. Основная идея заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике. Алгоритм завершается, когда на какой-то итерации не происходит изменения кластеров. Проблемы алгоритма k-means: необходимо заранее знать количество кластеров, алгоритм очень чувствителен к выбору начальных центров кластеров.

1.2.9. Случайный лес (RandomForest)

Случайный лес (RandomForest) - представитель ансамблевых методов.

Для определения входных данных каждому дереву используется метод случайных подпространств. Базовые алгоритмы обучаются на различных подмножествах признаков, которые выделяются случайным образом.

Преимущества случайного леса:

– высокая точность предсказания;

– редко переобучается;

– практически не чувствителен к выбросам в данных;

– одинаково хорошо обрабатывает как непрерывные, так и дискретные признаки, данные с большим числом признаков;

– высокая параллелизуемость и масштабируемость.

Из недостатков можно отметить, что его построение занимает больше времени. Так же теряется интерпретируемость.

1.2.10. Нейронная сеть

Нейронная сеть — это последовательность нейронов, соединенных между собой связями. Структура нейронной сети пришла в мир программирования из биологии. Вычислительная единица нейронной сети — нейрон или персептрон.

У каждого нейрона есть определённое количество входов, куда поступают сигналы, которые суммируются с учётом значимости (веса) каждого входа.

Смещение – это дополнительный вход для нейрона, который всегда равен 1 и, следовательно, имеет собственный вес соединения.

Так же у нейрона есть функция активации, которая определяет выходное значение нейрона. Она используется для того, чтобы ввести нелинейность в нейронную сеть. Примеры активационных функций: relu, сигмоида.

У полносвязной нейросети выход каждого нейрона подается на вход всем нейронам следующего слоя. У нейросети имеется:

– входной слой — его размер соответствует входным параметрам;

– скрытые слои — их количество и размерность определяем специалист;

– выходной слой — его размер соответствует выходным параметрам.

Прямое распространение – это процесс передачи входных значений в нейронную сеть и получения выходных данных, которые называются прогнозируемым значением.

Прогнозируемое значение сравниваем с фактическим с помощью функции потери. В методе обратного распространения ошибки градиенты (производные значений ошибок) вычисляются по значениям весов в направлении, обратном прямому распространению сигналов. Значение градиента вычитают из значения веса, чтобы уменьшить значение ошибки. Таким образом происходит процесс обучения. Обновляются веса каждого соединения, чтобы функция потерь минимизировалась.

Для обновления весов в модели используются различные оптимизаторы.

Количество эпох показывает, сколько раз выполнялся проход для всех примеров обучения.

Нейронные сети применяются для решения задач регрессии, классификации, распознавания образов и речи, компьютерного зрения и других. На настоящий момент это самый мощный, гибкий и широко применяемый инструмент в машинном обучении.

2. Практическая часть

2.1. Предобработка данных

В исходном датасете очень много клеток (больше трёх миллионов).

Матрица имеет размерность число клеток × число генов, то есть 3069478 × 36601.

```
adata_unfiltered = sc.read_10x_h5("outs/raw_feature_bc_matrix.h5")
print(adata_unfiltered)
```

```
reading outs/raw_feature_bc_matrix.h5
 (0:00:21)
AnnData object with n_obs × n_vars = 3069478 × 36601
    var: 'gene_ids', 'feature_types', 'genome'
```

Проверяем, что разные обьекты в AnnData содержат информацию

```
adata_unfiltered.obs.head() # Шапка датафрейма с аннотацией клеток, кроме баркодов пока ничего нет
```

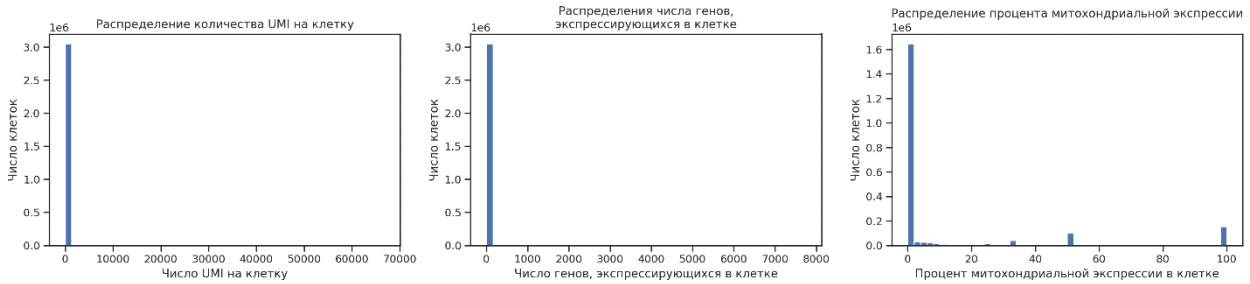
AAACCCAAGAAACACT-1
AAACCCAAGAAACCAT-1
AAACCCAAGAAACCCA-1
AAACCCAAGAAACCTG-1
AAACCCAAGAAACTAC-1

```
adata_unfiltered.var.head() # Шапка датафрейма с аннотацией генов
```

	gene_ids	feature_types	genome
MIR1302-2HG	ENSG00000243485	Gene Expression	GRCh38
FAM138A	ENSG00000237613	Gene Expression	GRCh38
OR4F5	ENSG00000186092	Gene Expression	GRCh38
AL627309.1	ENSG00000238009	Gene Expression	GRCh38
AL627309.3	ENSG00000239945	Gene Expression	GRCh38

Строим графики:

1) отрисовываем распределение общего числа UMI на клетку,
2) отрисовываем распределение числа генов по клеткам и
3) отрисовываем распределение митохондриальной экспрессии.

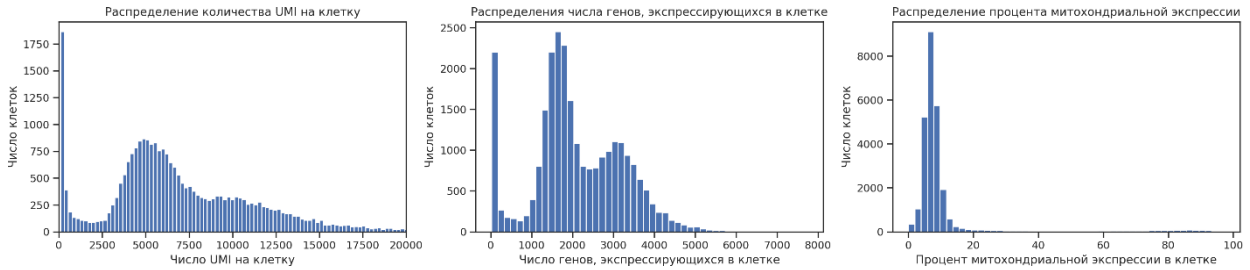


2.1.1. Удаление пустых клеток

При визуализации исходных данных видно, что подавляющее большинство «клеток» имеют околонулевую экспрессию. Фильтруем клетки, оставляем только те, что экспрессируют ≥ 100 UMI на клетку, то есть являются реальными клетками с мРНК, а не загрязнением внешней РНК при пробоподготовке.

```
adata_unfiltered = adata_unfiltered[adata_unfiltered.obs.total_counts >= 100]
```

И снова строим графики



Картина сильно улучшилась. Однако мы до сих пор видим сильный пик в начале. Для того, чтобы от него избавиться, можно просто вручную посмотреть минимум между двумя пиками (например, по числу генов) — здесь он находится где-то на 500 генах.

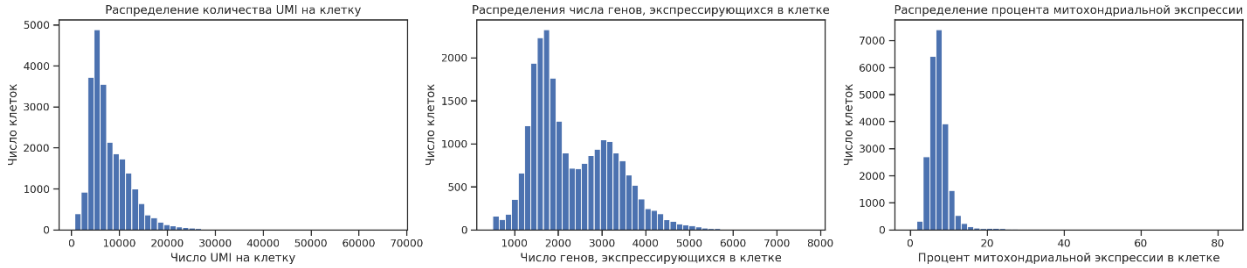
2.1.2. Удаление погибших клеток с митохондриальными генами.

Если клетки погибли до или во время диссоциации ткани, то в таких клетках будет огромное количество дифференциально экспрессных генов митохондрий, которые своим фоном практически закрывают экспрессию

мРНК клетки, что затрудняет процесс определения принадлежности клетки к какому-либо кластеру, поэтому от таких строк с клетками, тоже необходимо избавляться.

```
adata_unfiltered = adata_unfiltered[adata_unfiltered.obs.n_genes_by_counts >= 500]
adata = adata_unfiltered[adata_unfiltered.obs.pct_counts_mt <= 20]
```

И также визуализируем полученные данные препроцессинга



Записываем в формат AnnData наши данные, чтобы была возможность их в любой момент прочитать

```
adata.write_h5ad("10x.h5ad")
```

```
adata = sc.read_h5ad("10x.h5ad")
```

2.1.3. Удаление дублетов

Помимо пустых капель у нас существует и другая проблема — дублеты из нескольких клеток. Для того, чтобы их выявить и удалить, мы используем пакет scrublet.

```
sce.pp.scrublet(adata)
```

```
Running Scrublet
filtered out 12617 genes that are detected in less than 3 cells
normalizing counts per cell
    finished (0:00:00)
extracting highly variable genes
    finished (0:00:04)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
normalizing counts per cell
    finished (0:00:00)
normalizing counts per cell
    finished (0:00:00)
Embedding transcriptomes using PCA...
Automatically set threshold at doublet score = 0.19
Detected doublet rate = 5.2%
Estimated detectable doublet fraction = 61.2%
Overall doublet rate:
        Expected   = 5.0%
        Estimated  = 8.5%
    Scrublet finished (0:01:07)
```

```
adata.obs.to_csv("adata_obs.tsv", sep="\t")
```

```
obs = pd.read_csv("adata_obs.tsv", sep="\t", index_col=0)
adata.obs = obs
```

Теперь в adata.obs у нас появилась дополнительная колонка: predicted_doublets

```
adata.obs.predicted_doublet
```

```
AAACCCAAGAGCAGCT-1    False
AAACCCACAATACCTG-1    False
AAACCCACACAACCGC-1    False
AAACCCACACACAGAG-1     True
AAACCCACAGATCATC-1    False
                      ...
TTTGTTGGTTCAAGGG-1    False
TTTGTTGTCACCTGGG-1    False
TTTGTTGTCATTGAGC-1    False
TTTGTTGTCCGATGTA-1    False
TTTGTTGTCGTGGCTG-1    False
Name: predicted_doublet, Length: 23232, dtype: bool
```

Теперь можно удалить дублеты (1213 дублетов из 23232 клеток)

```
print(f"Обнаруженное число дублетов: {sum(adata.obs.predicted_doublet)} (из {len(adata)} клеток всего)")
adata = adata[~adata.obs.predicted_doublet]
print(adata)
```

```
Обнаруженное число дублетов: 1213 (из 23232 клеток всего)
View of AnnData object with n_obs × n_vars = 22019 × 36601
    obs: 'n_genes_by_counts', 'total_counts', 'total_counts_mt', 'pct_counts_mt', 'doublet_score', 'predicted_doublet'
    var: 'gene_ids', 'feature_types', 'genome', 'mt', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts', 'total_counts'
    uns: 'scrublet'
```

2.1.4. Нормализация и логарифмирование данных

Наши данные из-за особенностей процедуры имеют слишком высокую техническую дисперсию (график не показан). Вероятностная модель, которая описывает распределение числа UMI какого-то гена на клетку, представляет из себя обратное биномиальное распределение

$$n(\text{UMI}) \sim NB(\mu; \mu + \frac{\mu^2}{\theta}).$$

Для контроля над дисперсией можно использовать несколько различных подходов. Самый простой — мы сначала нормируем количество каунтов каждой клетки так, чтобы это значение было равно какой-то константе (сейчас, как правило, это число — это медианное число каунтов в клетках датасета), а потом логарифмируем отнормированные каунты (а точнее берём функцию log(x + 1), чтобы избежать проблемы с нулями и иметь только положительные значения в результате).

```
print("Суммы UMI на клетку до нормировки:", adata.X.sum(axis=1).T)

sc.pp.normalize_total(adata) # Нормировка
print("Суммы UMI на клетку после нормировки:", adata.X.sum(axis=1).T)
```

```
Суммы UMI на клетку до нормировки: [[ 9203.  5705.  8877. ...  6433. 12527.  4419.]]
normalizing counts per cell
    finished (0:00:00)
Суммы UMI на клетку после нормировки: [[6415.     6414.9995 6415.     ... 6414.999  6415.0005 6414.999 ]]
```

```
sc.pp.log1p(adata) # логарифмирование данных
```

2.2. Анализ данных

Подготовка к кластеризации и снижению размерности. Выделяем самые высоко вариабельные гены. Это необходимо, чтобы амплифицировать сигнал, которая у нас есть в нашем датасете.

```
sc.pp.highly_variable_genes(adata, n_top_genes=3000)
```

```
If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
    finished (0:00:02)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
```

Теперь в описании adata.var есть отдельное поле, которое сообщает нам, является ли ген высоко вариабельным.

```
adata.var.head() # Выводим верхушку adata.var
```

	gene_ids	feature_types	genome	mt	n_cells_by_counts	mean_counts	pct_dropout_by_counts	total_counts	highly_variable
MIR1302-2HG	ENSG00000243485	Gene Expression	GRCh38	False	0	0.000000	100.000000	0.0	False
FAM138A	ENSG00000237613	Gene Expression	GRCh38	False	0	0.000000	100.000000	0.0	False
OR4F5	ENSG00000186092	Gene Expression	GRCh38	False	0	0.000000	100.000000	0.0	False
AL627309.1	ENSG00000238009	Gene Expression	GRCh38	False	79	0.000026	99.997426	79.0	False
AL627309.3	ENSG00000239945	Gene Expression	GRCh38	False	12	0.000004	99.999609	12.0	False

Сохраняем исходные данные в отдельный объект, чтобы к нему всегда можно было вернуться

```
adata.raw = adata #Сохраняем исходные данные в отдельный объект
```

А также перезаписываем данные для дальнейшей работы

```
adata.write_h5ad("10xafterscrublet.h5ad")
```

```
adata = sc.read_h5ad("10xafterscrublet.h5ad")
```

2.2.1. Снижение размерности – Метод главных компонент

Дальше оставляем в нашем объекте только гены, которые мы отнесли к высоко вариабельным (HVG), то есть матрица становится 22019x3000

```
adata = adata[:, adata.var.highly_variable] # Оставляем только подмножество генов, которые относятся к HVG
print(adata)
```

```
View of AnnData object with n_obs × n_vars = 22019 × 3000
    obs: 'n_genes_by_counts', 'total_counts', 'total_counts_mt', 'pct_counts_mt', 'doublet_score', 'predicted_doublet'
    var: 'gene_ids', 'feature_types', 'genome', 'mt', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts', 'total_counts'
    uns: 'scrublet', 'log1p', 'hvg'
```

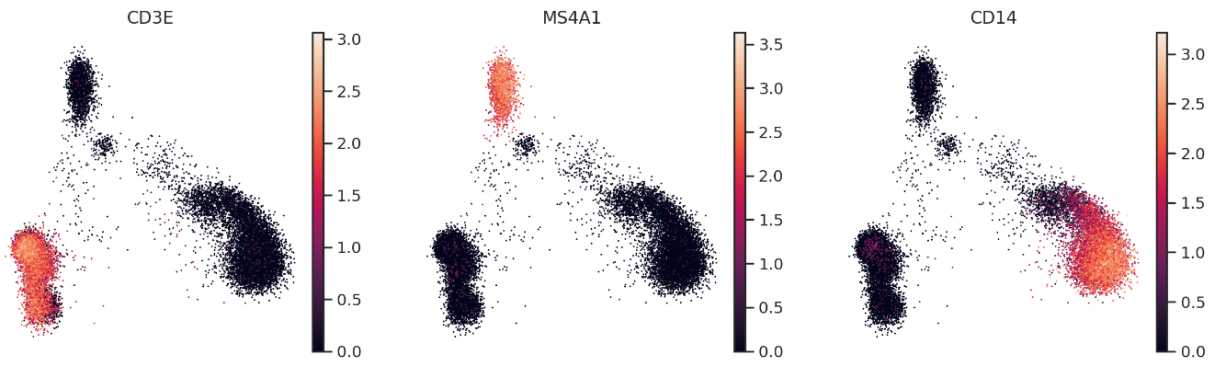
Проводим анализ методом главных компонент. Это позволит получить первичное представление о том, как данные распределены в пространстве, Первые N компонент PCA сохраняют максимальную биологическую значимость и в это же время устраняют шумы.

```
sc.tl.pca(adata) # Вычисляем PCA
sc.pl.pca(adata, frameon=False) # Отрисовываем PCA
```

```
computing PCA
    on highly variable genes
    with n_comps=50
    finished (0:00:06)
```


Часть клеток можно визуализировать, если знать маркерные гены и уровень их экспрессии:

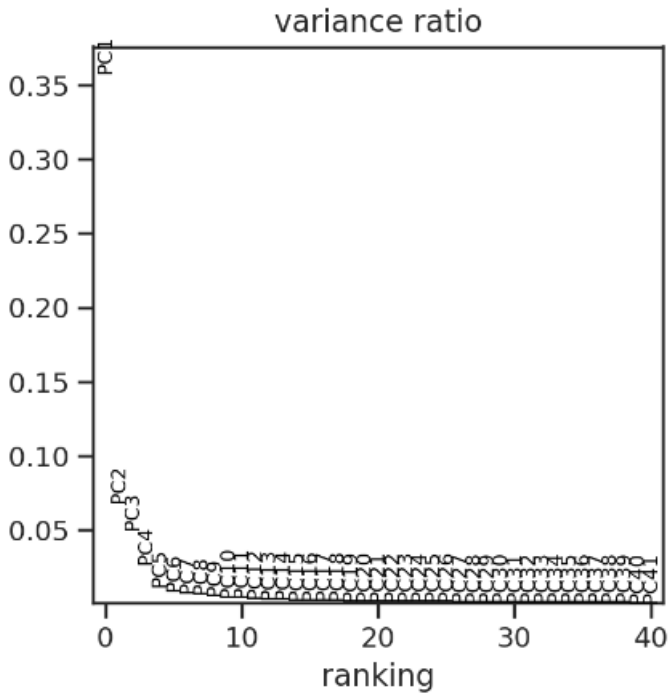
```
sc.pl.pca(adata, color=["CD3E", "MS4A1", "CD14"], frameon=False) # Отрисовываем PCA по 3 типам клеток
```



Пока это далеко от красивой картинки, на которой отчётливо видны различные клеточные популяции. Но это нормально, потому что

кластеризацию и анализ популяций мы будем проводить на UMAP и tSNE. UMAP обычно строится на каком-то количестве первых компонент. Для того, чтобы их подобрать, нарисуем ElbowPlot.

```
sc.pl.pca_variance_ratio(adata, log=False, n_pcs=40)
```



Теперь приступим к снижению размерности при помощи UMAP и t-SNE. Параметры для запуска UMAP и t-SNE мы возьмём некоторые дефолтные

2.2.2. Метод k-ближайших соседей

Вычисление графа k-ближайших соседей является важным этапом перед кластеризацией методами tSNE, UMAP и алгоритмом Leiden.

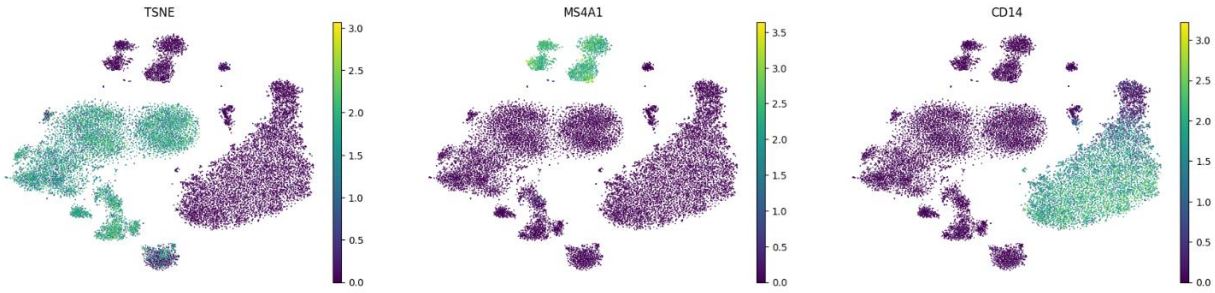
```
sc.pp.neighbors(adata, n_pcs=30) # Вычисление графа соседей
```

```
computing neighbors
    using 'X_pca' with n_pcs = 30
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:42)
```

2.2.3. tSNE и UMAP

Кластеризация и визуализация некоторых высокоэкспрессных генов методом tSNE

```
sc.tl.tsne(adata)
sc.pl.tsne(adata, color=["CD3E", "MS4A1", "CD14"], frameon=False, title="TSNE")
```

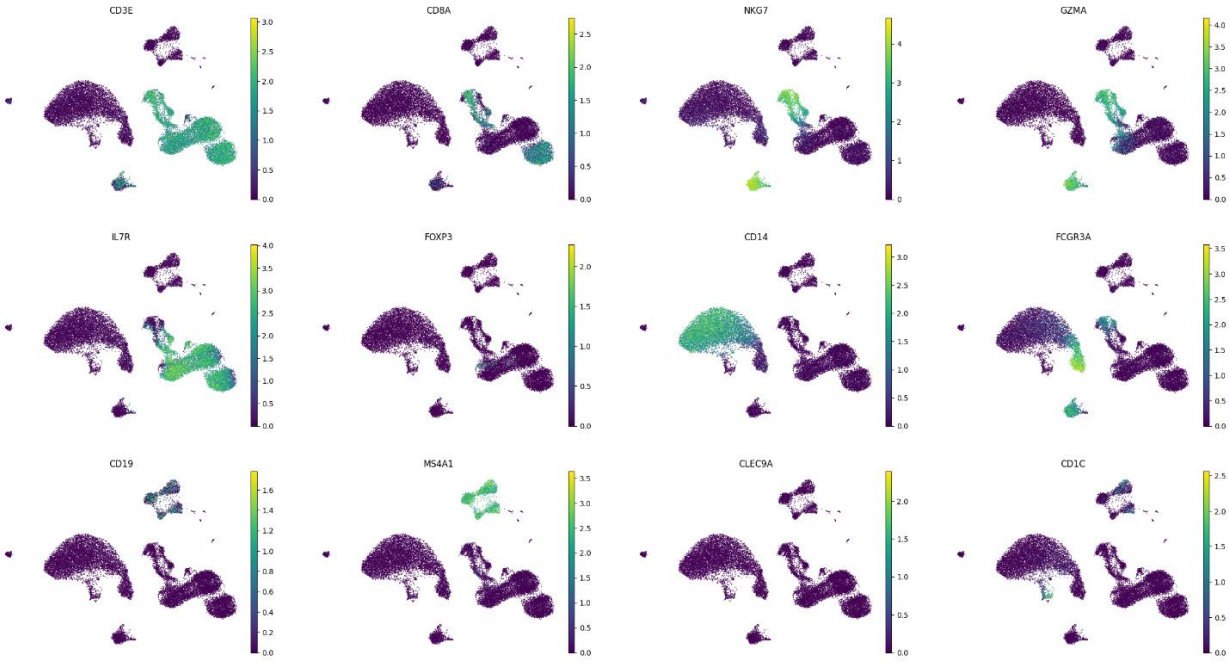


Мы уже видим, что кластеров достаточно много

Пробуем сделать визуализацию UMAP

```
sc.tl.umap(adata) # Рассчёт UMAP
```

```
sc.pl.umap(
    adata,
    color=[
        "CD3E", "CD8A", "NKG7", "GZMA", "IL7R",
        "FOXP3", "CD14", "FCGR3A", "CD19", "MS4A1",
        "CLEC9A", "CD1C", "TCF4",
      ],
    frameon=False
)
```



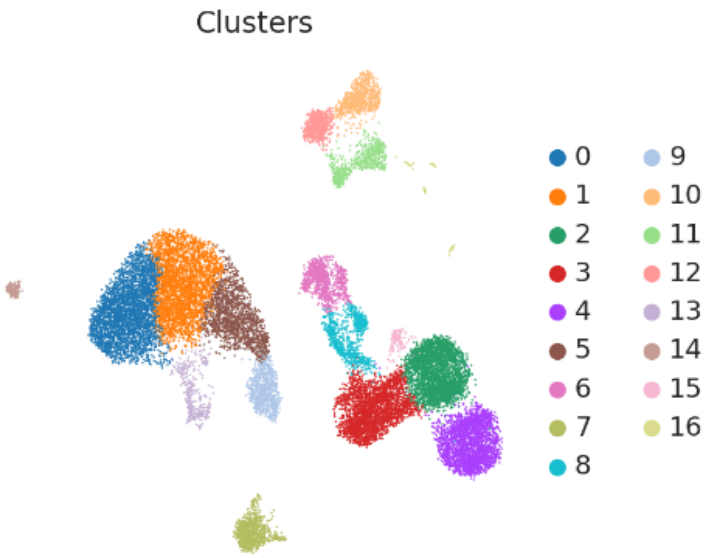
2.2.4. Leiden кластеризация

Некоторым стандартом кластеризации является алгоритм leiden

```
sc.tl.leiden(adata) # Кластеризация алгоритмом Leiden
```

```
running Leiden clustering
    finished: found 17 clusters and added
    'leiden', the cluster labels (adata.obs, categorical) (0:00:05)
```

```
sc.pl.umap(adata, color="leiden", frameon=False, title="Clusters", s=3) # Отрисовываем UMAP
```



Возможное продолжение работы:

В дальнейшем необходимо понять, какой кластер соответствует какому типу клеток. Для этого мы можем:

1. определить, какие гены дифференциально экспрессированны в данном кластере, а потом, исходя из этого, определить тип клетки;
2. посмотреть, какие заранее известные гены-маркеры в каком кластере экспрессируются, и исходя из этого определить тип клетки.

   Ниже приведен пример фильтрации данных 0 кластера

```
DEG = sc.get.rank_genes_groups_df(adata, group="0")
DEG[DEG.logfoldchanges > 2.5]
```

	names	scores	logfoldchanges	pvals	pvals_adj
62	IL7R	73.588806	2.593273	0.000000e+00	0.000000e+00
79	CCR7	59.787647	2.685055	0.000000e+00	0.000000e+00
95	LEF1	50.137753	2.549269	0.000000e+00	0.000000e+00
115	FHIT	36.958820	3.028283	2.112866e-245	7.851065e-244
123	CHRM3-AS2	33.802128	2.736526	5.483530e-212	1.813033e-210
...	...	...	...	...	...
5090	AC008581.1	0.817699	2.514848	4.135972e-01	6.747524e-01
5091	TEX101	0.817415	2.550114	4.137581e-01	6.749547e-01
5092	AC073525.1	0.817241	2.548816	4.138575e-01	6.750869e-01
5094	AC019257.2	0.816243	2.541338	4.144279e-01	6.758825e-01
5095	HPN-AS1	0.816241	2.541331	4.144286e-01	6.758825e-01

614 rows × 5 columns

2.2.5 KMeans

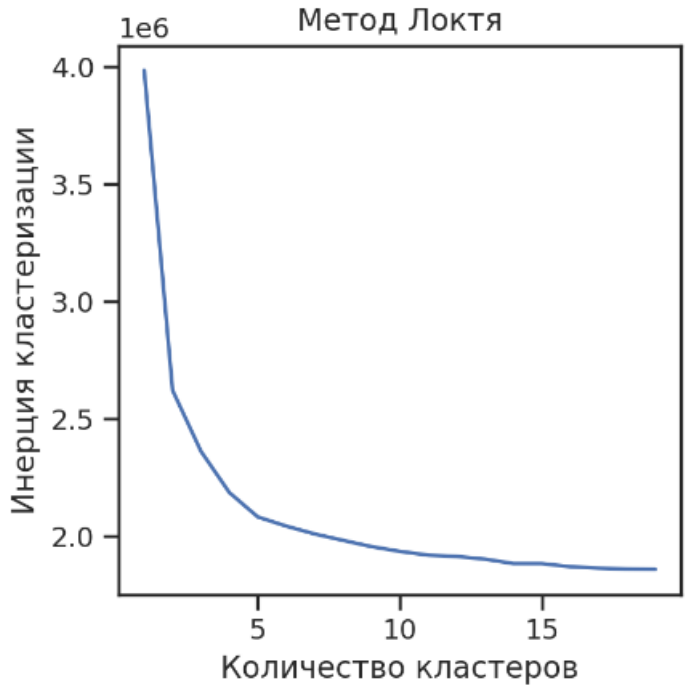
KMeans не смог отработать на данных AnnData, поэтому для начала необходимо было данные перевести в Data Frame. А затем использовав метод локтя определить количество кластеров для обучения

```
X = adata.to_df(layer=None) #переход от annData в дата фрейм
```

```
X
```

	HES4	ISG15	TNFRSF18	TNFRSF4	AL391244.2	MMP23B	PLCH2	MEGF6	CCDC27	TNFRSF25	...	PNMA6F	PDZD4	HCFC1-AS1
AAACCCAAGAGCAGCT-1	0.0	1.128548	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0
AAACCCACAATACCTG-1	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.178318	...	0.0	0.0	0.0
AAACCCACACAACCGC-1	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0
AAACCCACAGATCATC-1	0.0	0.415752	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0
AAACCCACAGCGCGTT-1	0.0	1.482164	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
TTTGTTGGTTCAAGGG-1	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.227463	...	0.0	0.0	0.0

```
#определение количества кластеров для обучения k-means
n_inertia=[]
for i in range(1, 20):
  model = KMeans(n_clusters=i).fit(X)
  n_inertia.append(model.inertia_)

fig, ax = plt.subplots()
sns.lineplot(x=np.arange(1, 20), y=n_inertia, ax=ax)
ax.set_title('Метод Локтя')
ax.set_xlabel('Количество кластеров')
ax.set_ylabel('Инерция кластеризации')
```



Здесь видно, что уже после 17 кластеров изменений практически нет, поэтому запустим KMeans с 17 кластерами (такое же количество кластеров мы получили алгоритмом кластеризации Leiden)

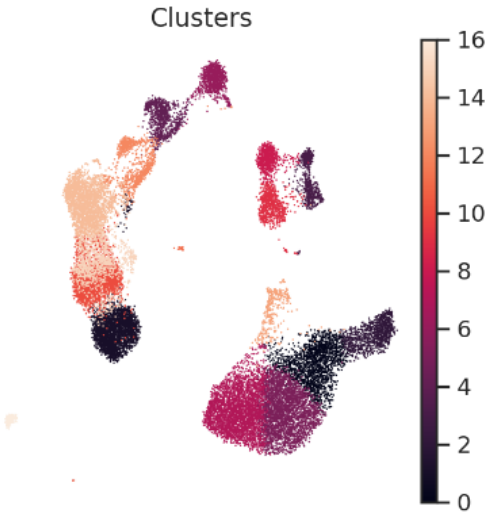
```
model = KMeans(n_clusters=17).fit(X)

# Предсказание на всем наборе данных
all_predictions = model.predict(X)

# Выводим предсказания
print(all_predictions)
```

```
[ 5  1  7 ... 10  7  4]
```

Для того, чтобы визуализировать полученные данные, мы обратно data frame переводим в формат AnnData и отрисовываем UMAP

```
adata1 = sc.AnnData(X)
adata1.obs['cluster'] = all_predictions
sc.pp.neighbors(adata1)
sc.tl.umap(adata1, n_components=2)
sc.pl.umap(adata1, color=['cluster'], frameon=False, title="Clusters", s=3) #отрисовываем UMAP
```

```
computing neighbors
WARNING: You're trying to run this on 3000 dimensions of `.X`, if you really want this, set `use_rep='X'`.
         Falling back to preprocessing with `sc.pp.pca` and default params.
computing PCA
    with n_comps=50
    finished (0:00:10)
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:01:04)
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:19)
```



2.2.6. Random forest

Для метода случайного леса пришлось в качестве предсказываемого кластера (y) использовать данные, полученные методом kmeans. Данный метод обучился предсказывать класс (принадлежность к популяции) с точностью 82%

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# разбиваем данные на обучающую и тестовую выборки
y = all_predictions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# создаем и обучаем модель на обучающей выборке
clf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
clf.fit(X_train, y_train)

# оцениваем качество модели на тестовой выборке
accuracy = clf.score(X_test, y_test)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.82
```

2.2.7. Нейронная сеть

Для того, чтобы кластеризовать данные с помощью нейронной сети, необходимо было добавить колонку с предсказываемыми данными в data frame

```
#Подготовка дата фрейма
X['cluster'] = all_predictions
X1 = X.copy()
y = X['cluster']
del X['cluster']
```

Затем мы разделили данные на тренировочную и тестовую выборки и скомпилировали нейронную сеть, в которой было 2 слоя с активацией relu и softmax. Во втором слое количество выходных нейронов должно соответствовать количеству кластеров.

```python
#нейронная сеть
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
n_classes = 17
y_train = keras.utils.to_categorical(y_train, n_classes)
y_test = keras.utils.to_categorical(y_test, n_classes)
n_genes = X.shape[1]

#Структура нейронной сети
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(n_genes, )),
    tf.keras.layers.Dense(n_classes, activation='softmax') # n_clusters - количество кластеров для предсказания
])

# Компиляция модели
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# обучаем модель
history = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_split=0.1)

#оцениваем обученную модель
model.evaluate(X_test, y_test, verbose=1)
```

На 20 эпохе точность предсказания составила 94,8%

```
Epoch 20/20
109/109 [==============================] - 1s 6ms/step - loss: 0.0102 - accuracy: 0.9999 - val_loss: 0.1389 - val_accuracy: 0.9481
207/207 [==============================] - 0s 2ms/step - loss: 0.1398 - accuracy: 0.9479
[0.13979719579219818, 0.9479261040687561]
```

Архитектура модели

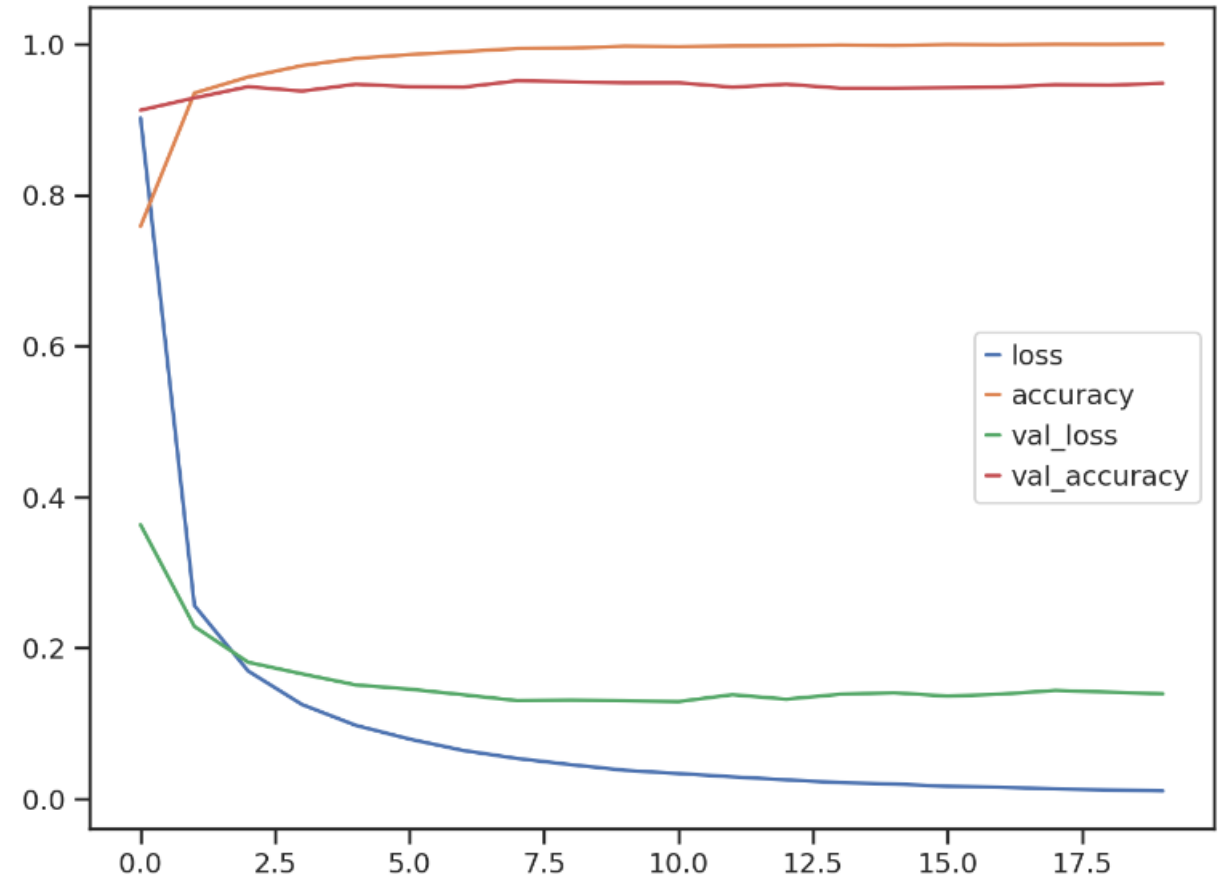
```python
model.summary() # архитектура модели
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 32)                96032

 dense_7 (Dense)             (None, 17)                561

=================================================================
Total params: 96,593
Trainable params: 96,593
Non-trainable params: 0
_________________________________________________________________
```

Метрики и функции потерь

```
pd.DataFrame(history.history).plot(figsize=(8,6)) # метрики и функции потерь
```

```
<Axes: >
```



2.3. Разработка приложения

Разработка приложения, которое будет проводить препроцессинг данных, осуществлялась, однако, так и не была реализована на платформе. Причин несколько: 1) при запуске CellRanger возможно с помощью флагов прописать все необходимые этапы препроцессинга и получить уже готовые к кластеризации данные. 2) уже вышло приложение Azimuth, которое на вход принимает файл после картирования (в т.ч. после отработки CellRanger) в формате .h5, самостоятельно осуществляет препроцессинг, картирование и визуализацию данных (особенно касается часто секвенируемых фрагментов органов или тканей). 3) Если же транскриптом мало изучен, тогда есть смысл

все этапы проводить вручную, чтобы контролировать все этапы подготовки данных.

2.4. Создание удаленного репозитория

Для данного исследования был создан удаленный репозиторий на GitHub, который находится по адресу https://github.com/LyubovChuvakova/vkr_work. На него были загружены результаты работы: исследовательский notebook, пояснительная записка, презентация.

Заключение

В данной работе было определено 17 кластеров популяций клеток, которые в последующем предстоит проанализировать. Однако для данного анализа подходят только те методы, которые не требуют «y» (т.е. данные, которые мы и должны определить и распределить между клетками) на вход, а сами кластеризуют данные.

В ходе выполнения данной работы мы рассмотрели большую часть операций и задач, которые приходится выполнять специалисту по работе с данными.

Этот поток операций и задач включает:

– изучение теоретических методов анализа данных и машинного обучения;

– изучение основ предметной области, в которой решается задача;

– извлечение и трансформацию данных. Здесь нам был предоставлен реальный набор данных, поэтому через трудности работы с разными источниками и парсингом данных мы еще не соприкоснулись;

– выполнение предобработки (препроцессинга) данных для обеспечения корректной работы моделей;

– построение аналитического решения. Это включает выбор алгоритма решения и модели, сравнение различных моделей;

– визуализация модели и оценка качества аналитического решения;

Я проделала максимум исследований, которые в моей компетенции как начинающего дата-сайентиста и ученого с биоинформатическим анализом данных, я применила большую часть знаний, полученных в ходе прохождения курса.

Дальнейшие возможные пути решения этой задачи могли бы быть:

– углубиться в изучение нейросетей, попробовать различные архитектуры, параметры обучения и т.д.;

Список литературы:

1) https://anndata.readthedocs.io/en/latest/#

2) Wolf et al (2018*),* Scanpy: large-scale single-cell gene expression data analysis*,* Genome Biology https://doi.org/10.1186/s13059-017-1382-0.

3) Traag, V.A., Waltman, L. & van Eck, N.J. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep 9, 5233 (2019). https://doi.org/10.1038/s41598-019-41695-z