Оглавление

Описание наборов данных....1
MNIST....1
Ирисы Фишера....1
Extract Transform Load....2
KNN....2
K-Means....3
Naive Bayes....5
Network....6
Заключение....7

Описание наборов данных

MNIST

Данный набор данных представляет собой набор рукописных цифр. Тренировочный набор состоит из 60000 примеров. Каждый пример представляет собой фотографию 28 на 28 пикселей. Каждый пиксель представляет собой вещественное число от 0 до 255. Нижняя граница(ноль) означает то, что пиксель белый, а верхняя граница(255) означает то, что пиксель черный. Поскольку в этом проекте нас не интересует порядок этих пикселей, то каждый пример представляется в виде вектора длины $28 \cdot 28 = 784$. Можно сказать, что объект фотографии имеет 784 признака — затемненность пикселей, на основе которых нужно предсказать какому классу принадлежит этот объект.

Ирисы Фишера

В этом наборе данных представлена инфрмация о трех видах цветов ирисов. Набор состоит из 150 экземпляров, по 50 экземпляров каждого из трех видов - Iris setosa, Iris virginica, Iris versicolor. Для каждого экземпляра были измерены 4 характеристики (в сантиметрах):

- Длина наружной доли околоцветника (sepal length)

- Ширина наружной доли околоцветника (sepal width)
- Длина внутренней доли околоцветника (petal length)
- Ширина внутренней доли околоцветника (petal width)

На основании этих признаков нужно предсказать какому виду ирисов принадлежит тот или иной экземпляр ириса.

Extract Transform Load

DataHandler — класс отвечающий за считывание данных в общий контейнер. В этом же классе, можно нормализовать данные с помощью минимакс нормализации, которая определяется следующей формулой : $X_n = \frac{(X_i - X_{min})}{(X_{max} - X_{min})}$. т.е. Сначала смещаем, а потом переводим в отрезок [0,1]. т.к. знаменатель есть диапазон значений (или диаметр) исходного множества. Нормализация бывает необходима, т. к. дисбаланс между значениями признаков может вызвать неустойчивость работ модели, ухудшить результаты модели и замедлить процесс моделирования. В этом же классе происходит разбиение данных на тренировочную, тестовую и валидационную выборку. Последняя нужна, чтоб посмотреть как происходит изменение точности модели, какие значения вообще модель умеет предсказывать, скажем так и проч. Элементарным контейнером для экземпляра данных выступает класс Data. В котором хранится вектор признаков, реальный класс которому этот экземпляр принадлежит, расстояние до других экземпляров и др.

KNN

Метод k-ближайших соседей для классификации работает так:

1. Задаем k — количество соседей, которых будем искать.

2. Берем произвольную точку из тренировочный выборки и ищем k ближайших точек. Я делаю это последовательно, т. е. сначала нахожу ближайшую точку m из всего множества M, потом ближайшую точку из множества $M \setminus a$ и т. д. Сложность такого алгоритма есть $O(k \cdot n \cdot m)$, где k — число соседей, n — размер выборки, m - размерность пространства. Если $k \sim n \Rightarrow O(k \cdot n \cdot m) = O(m \cdot n^2)$. Если k и m небольшие, то сложность линейная, но это кажется не всегда так, ибо в MNIST размерность пространства есть число 784, и кажется, что это не маленький коэффициент и нельзя им просто пренебречь.
3. Находим самый популярный класс из ближайших соседей. Этот класс и будет предсказанием модели.
4. Обучаемым, скажем так, параметром здесь является k - количество соседей. Запускаем итеративный процесс и смотрим на результаты модели.

Опытным путем было определено, что для набора MNIST при k = 3 точность модели самая высокая и равняется 95%. Для набора ирисов при k = 4 точность модели составила 95.455% при других k точность была либо такой же либо ниже. Более того в какой-то момент с увеличением k точность падает.

K-Means

Это алгоритм также как и KNN зависит от входного параметра k — число кластеров на которые будет разбито пространство, при этом каждый экземпляр данных относится к тому кластеру, к центру которого этот экземпляр ближе всего. Центр класстеру называется центроидом. В отличии от KNN, используется именно Евклидово расстояние.

Алгоритм:

1. Задаем k — число кластеров.

2. Выбираем k случайных точек — первоначальные центроиды и рассчитываем расстояние кажой точки до каждого центроида, и добавляем эту точку к центроиду расстояние до которого минимально.

3. После 2-го шага мы разбили пространство на k — кластеров. Теперь в каждом класстере высчитываем центр масс множества элементов содержащихся в класстере. Центр масс тут — это среднее арифметическое. Теперь когда центры класстеров обновлены, заново строим кластеры. Делаем так до тех пор пока после обновления центроида, он не изменится.

4. Лучший класс высчитывается также как и в алгоритме KNN

Для набора MNIST почти нереально на домашнем компукторе что-то проверить, ибо модель начинает показывать хороший результат при довольно большом параметре k. А при большом количестве кластеров необходимо делать много вычислений и скорость работы становится довольно долгой. К примеру:

1) при k = 15 точность составила 59.97% (1 min);

2) при k = 100 точность составила 84.94% (5 min);

3) при k = 200 точность составила 87.24% (9 min);

4) при k = 300 точность составила 90.21%(13 min);

4) при k = 400 точность составила 91.08% (17 min);

5) при k = 484 точность составила 91.27% (20 min);

Видно, что с увеличением количества кластеров точность предсказания растет, но и время работы растет. Чтобы проверить диапазон значений от 500 до 600 придется 100 раз запустить алгоритм и каждый запуск займет не менее 20 минут, значит такая проверка займет не менее 33 часов(на самом деле где-то 40). Поэтому я делал это дело с шагом 100 :)

Для ирисов при k = 26, точность составила 95.45%.

Naive Bayes

Наивный байесовский классификатор — это классификатор основанный на применении теоремы Байеса. Наивный он потому-что делается предположение о независимости признаков друг от друга. Хотя, естественно, это может быть и неправдой.

Модель классификатора это условная модель : $P(C|F_1,\dots,F_n)$

т. е. Вероятность того, что объект с свойствами $F_1,\dots,F_n$ принадлежит классу С.

По теореме Байеса

$P(C|F_1,\dots,F_n)=\frac{P(C)\cdot P(F_1,\dots,F_n|C)}{P(F_1,\dots,F_n)}$. если применить теперь предположоение о независимости признаков, то получим

$$P(C|F_1,\dots,F_n)=\frac{1}{P(F_1,\dots,F_n)}P(C)\cdot\prod_{i=1}^{n}P(F_i|C)$$

Алгоритм:

1. Находим среднее арифметическое и стандартное отклонение каждого признака.

2. Разбиваем тренировочную выборку на k(кол-во классов) групп, где в каждой группе объекты только одного класса.

3. находим среднее арифметическое и стандартное отклонение в каждой группе (т. е. Для каждого класса)

4. Вероятность классов расчитываем по формуле последней взяв за условную вероятность значение из нормального распределения

$P(x)=\frac{1}{\sqrt{(2\pi\sigma^2)}}\cdot e^{(\frac{(x-\mu)}{2\sigma^2})}$ ¿

$\mu=\frac{1}{n}\sum x_i$ среднее

$\sigma=\frac{1}{n-1}\cdot\sum(x_i-\mu)$ стандартное отклонение

Для ирисов точность составила 95.45%. Для MNIST надо попробовать

Network

Нейронная сеть — это математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

Нейронные сети состоят из входного слоя — слой в который поступает информация извне, например, вектор объектов, где каждый объект представляет собой вектор признаков(рост, длина, высота, ширина, заработная плата и проч.). Размер входного слоя имеет размер равный размерности пространства объектов, т. е. Если объекты описываются 4 признаками, то объекты лежат в пространстве размерности 4, т. к. каждый объект это вектор длины 4, значит и размер входного слоя будет равен 4. Вслед за входным слоем идут скрытые слои произвольных размеров. Связь между слоями представляет собой полный двудольный граф, т. е. Из каждого нейрона 1-ого слоя выходит ребро в каждый нейрон 2-ого слоя. Причем граф взвешенный, т. е. информация не просто переходит от одного нейрона к другому, а она претерпевает некоторые изменение, что и позволяет нейросети обучатся. После скрытых слоев идет выходной слой, размер которого, в случае классификации объектов, равен количеству классов. В этом слое находится результат работы сети, который сравнивается с известным заранее ответом. Такой подход называется обучением с учителем. Наиболее популярным алгоритмом для обучения таких нейросетей является алгоритм обратного распространения ошибки.

Для того чтобы создать нейросеть необходимо в первую очередь задать несколько параметров, а именно количество нейронов в каждом скрытом слое и скорость обучения.

Процесс обучения состоит из таких шагов как:

1. Прямое распространение(front propogation)

2. Обратное распространение

3. Корректировка весов.

Опишем теперь каждый шаг:

1. Прямое распространение.

Во время прямого распространения мы берем один экземпляр данных и пропускаем этот «сигнал» через нейросеть от входного слоя к выходному и на выходе получаем вектор, размер которого, в случае классификации, равен количеству классов. Что же происходит в скрытых слоях? Ответ убил. Скрытые слои иначе называются вычислительными слоями. Вспомним, что в входном слое находится столько нейронов сколько признаков имеют экземпляры данных. В каждом нейроне входного слоя содержится одно число — i-тая координата входного вектора. Теперь мы распространяем числа с входного слоя к 1-ому скрытому. Во время распространения к числу содержащемуся в j-том нейроне входного слоя выполняется домножение на вес и применение сигмоидной функции. И так далее сигнал распространяется до выходного слоя.

2. Обратное распространение:

Сначала вычисляем ошибку предсказания. Теперь эту ошибку надо рапространить к входному слоя от выходного. Бдуме подправлять веса после каждого обучающего примера. Для этого надо вычислить пару производных применяя много раз цепное правило(позже напишу формулки). И таким образом мы вычислим коэффициент необходимый для корректировки веса.

3. Корректировка веса. И этим все сказано.

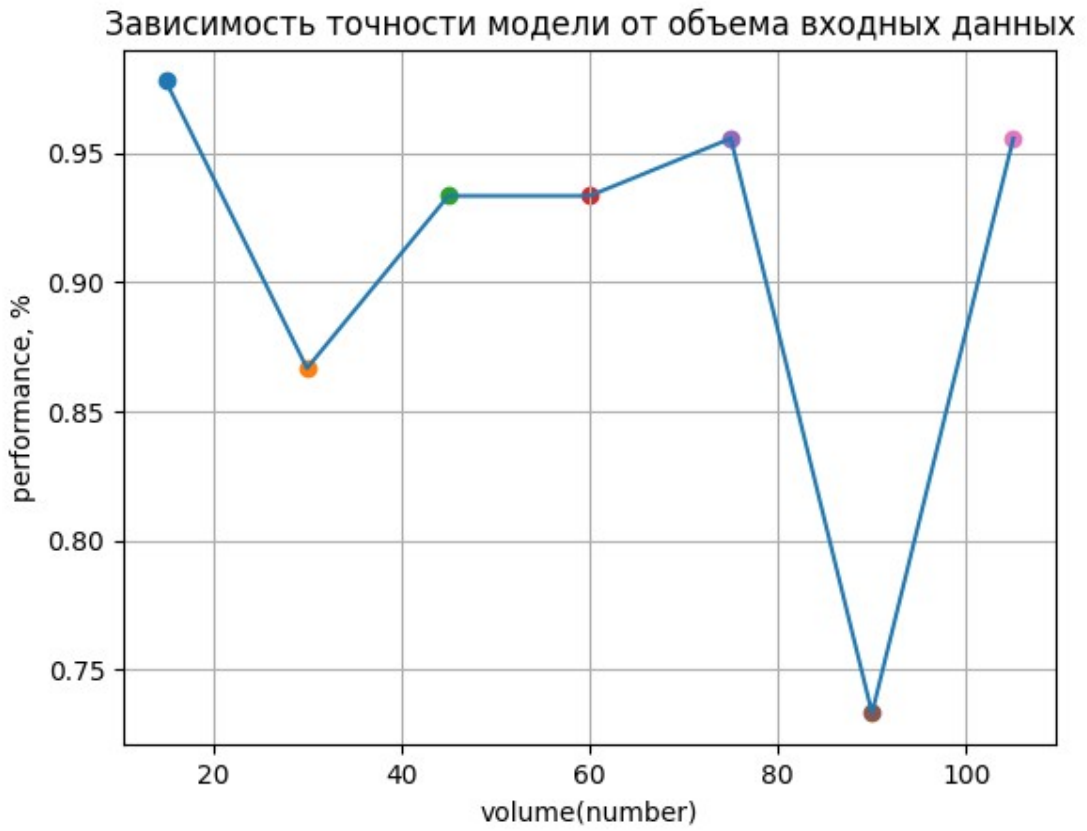
Таким образом мы пропускаем все тренировочные данные через этот «маршрут»

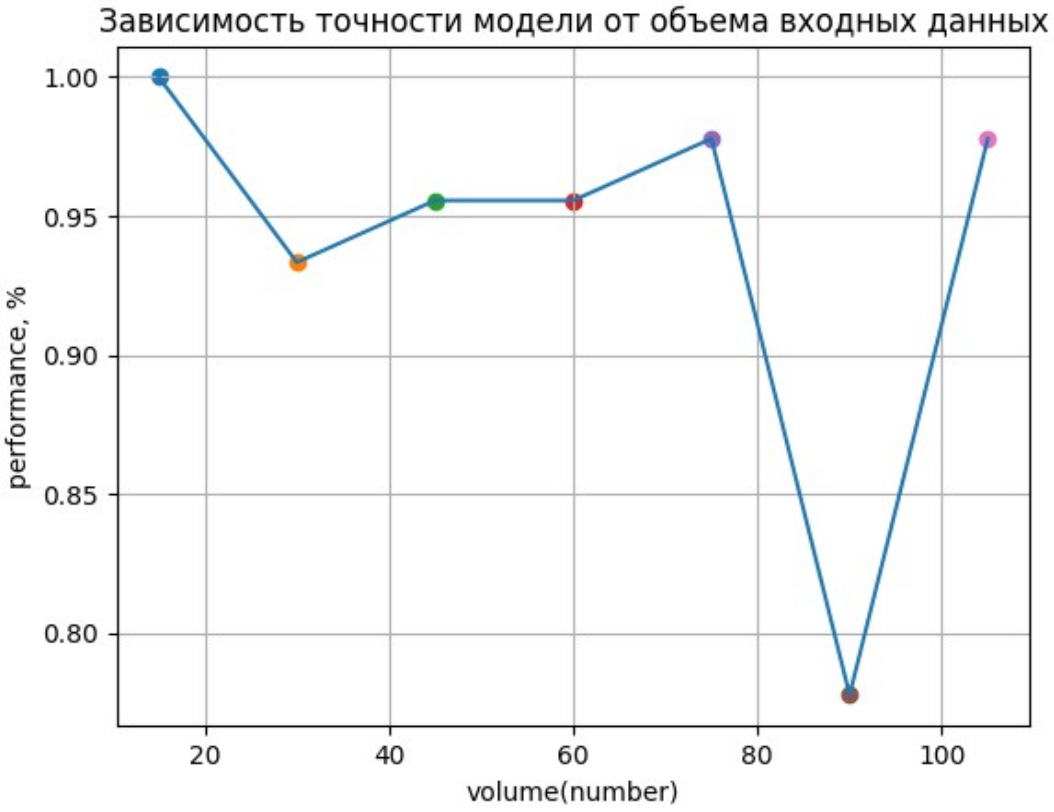
Заключение

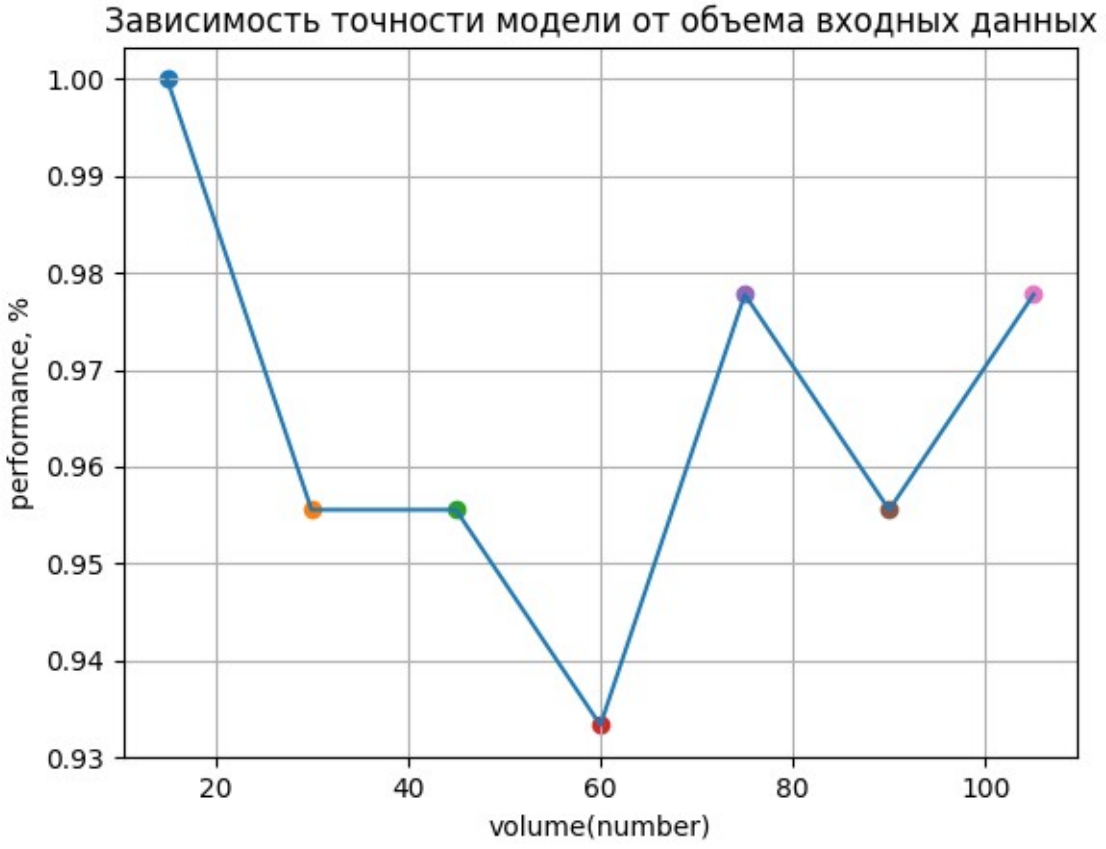
В заключение посмотрим на результаты модели и попробуем их проанализировать. Я обучил каждую модель на каждом наборе

данных с разыми размерами. Для MNIST это были 6000, 18000, 30000 и 42000 фотографий и каждая модель тестировалась на 300 примерах. Для ирисов это были 15, 45, 75 и 105 цветочков и каждая модель тестировалась на 45 примерах. Посмотрим сначала на точности моделей при разных входных данных и на время их работы.

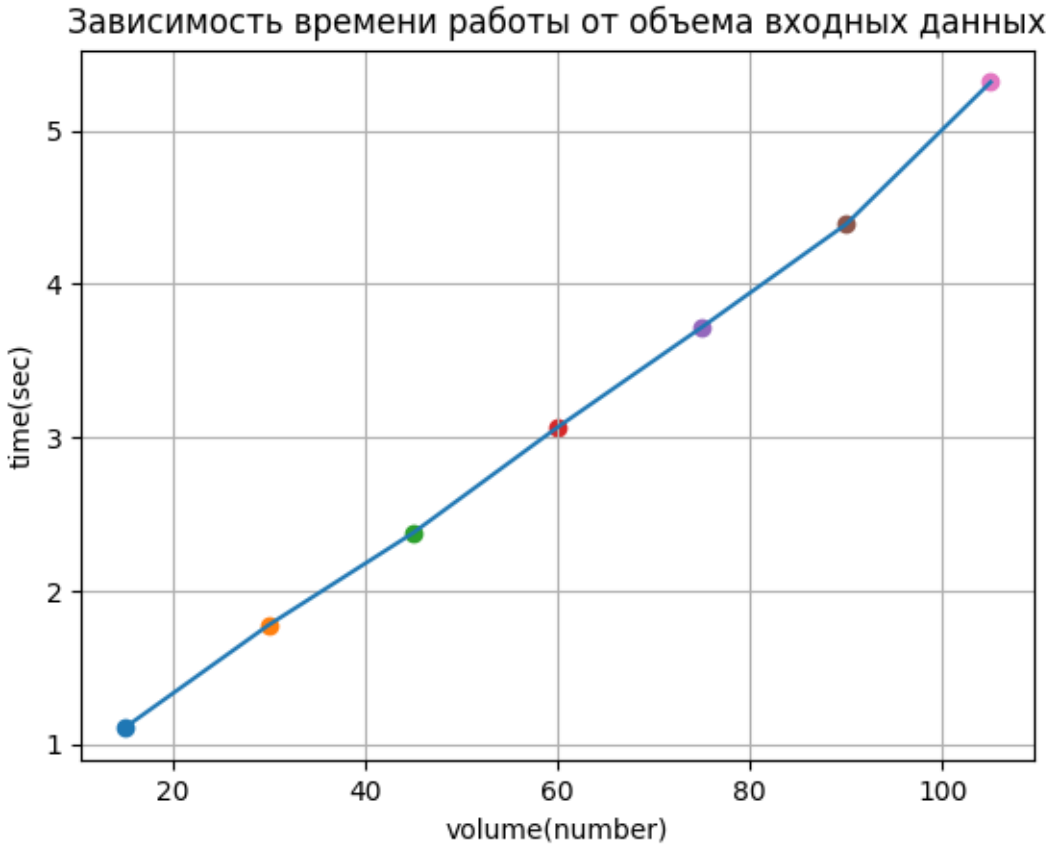
Начнем с нейронной сети. Ниже представлены результаты ее работы для параметров epochs = 15, 25, 35 на датасете ирисов соответственно. Рейтинг обучения(коэффициент скорости обучения) на всех примерах был равен 0.25 и нейронная сеть представляла собой один скрытый слой, размер которого менялся от 10 до 200 и выбирался лучший для конкретного размера входа в сеть.



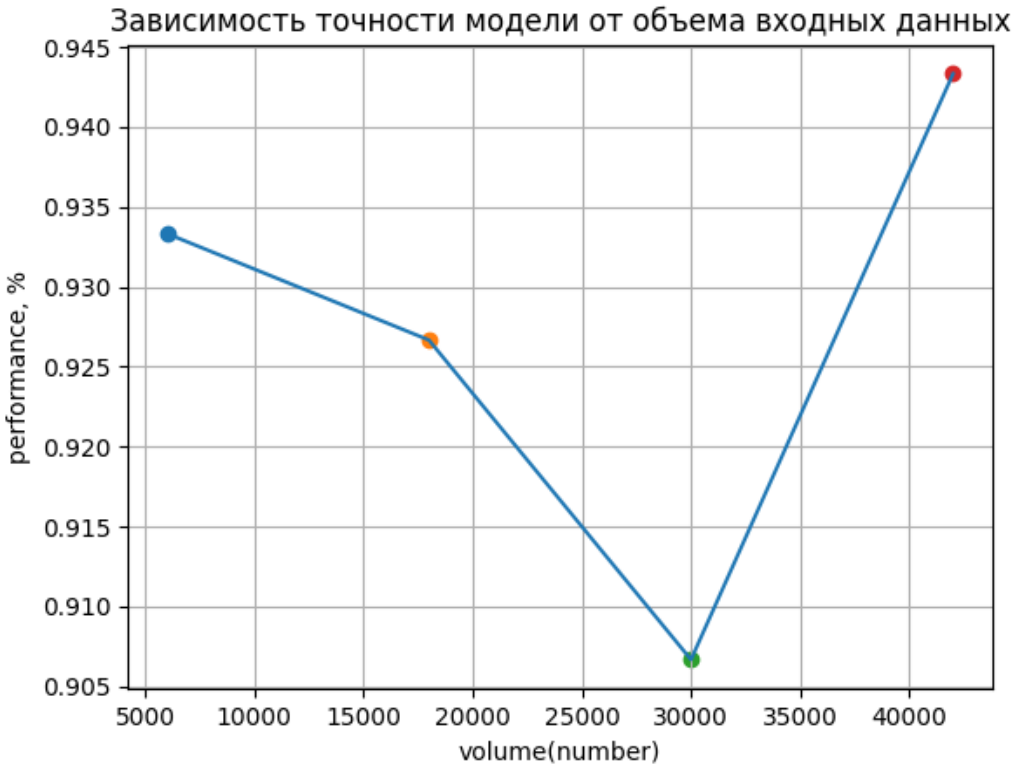

Зависимость точности модели от объема входных данных
1.00
0.95
0.90
0.85
0.80
performance, %
20
40
60
80
100
volume(number)

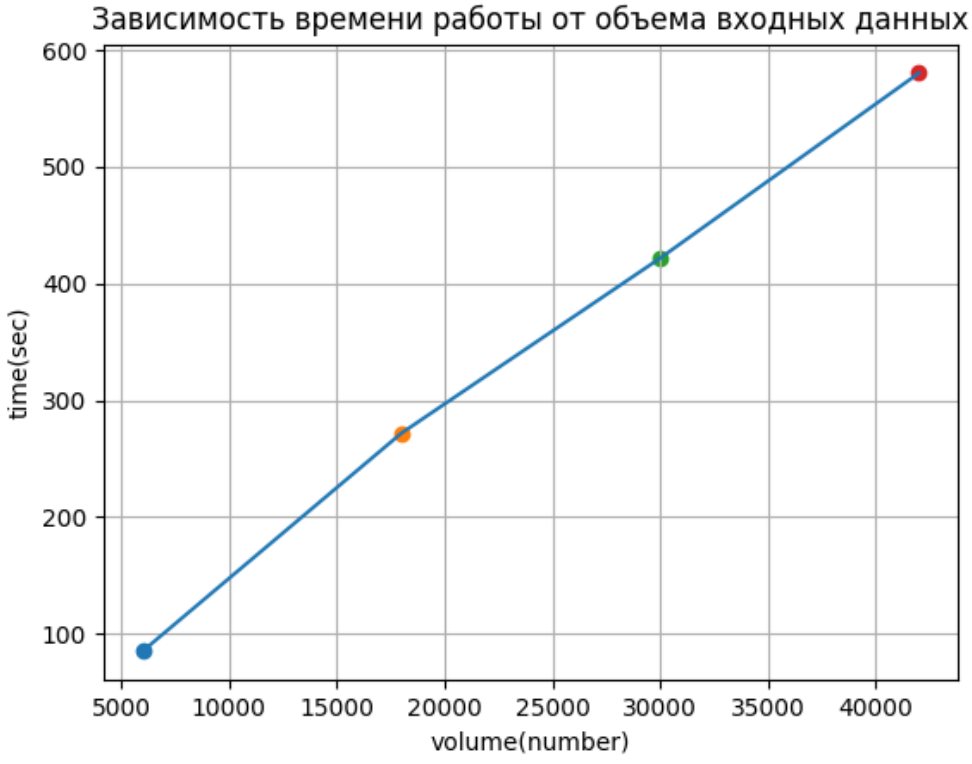

Зависимость точности модели от объема входных данных
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
performance, %
20
40
60
80
100
volume(number)

Из данных графиков видно, что модель нейронной сети для датасета ирисов не чувствительна к увеличению тренировочного набора. Так как для каждого размера входа был найден оптимальный размер скрытого слоя такой что модель с абсолютной точность предсказывала классы для тестового набора. Конечно, это не значит вовсе, что она никогда не будет ошибаться,т.к. она была протестирована всего лишь на 45 примерах, но это все равно отличный результат. Также можно обратить внимание на нижний предел на графиках. С увеличение количества эпох нижний предел поднимался, то есть нейросеть, которая обучалась больше эпох показывала результат гораздо лучше на тех же самых данных. Графики роста времени работы в зависимости от объема входа все очень похожи, поэтому представлю здесь один из них, соответствующий количеству эпох равному 25. Самое большое время работы для 15 и 35 отличается примерно на 1.5с в меньшую и большую сторону соответсвенно.



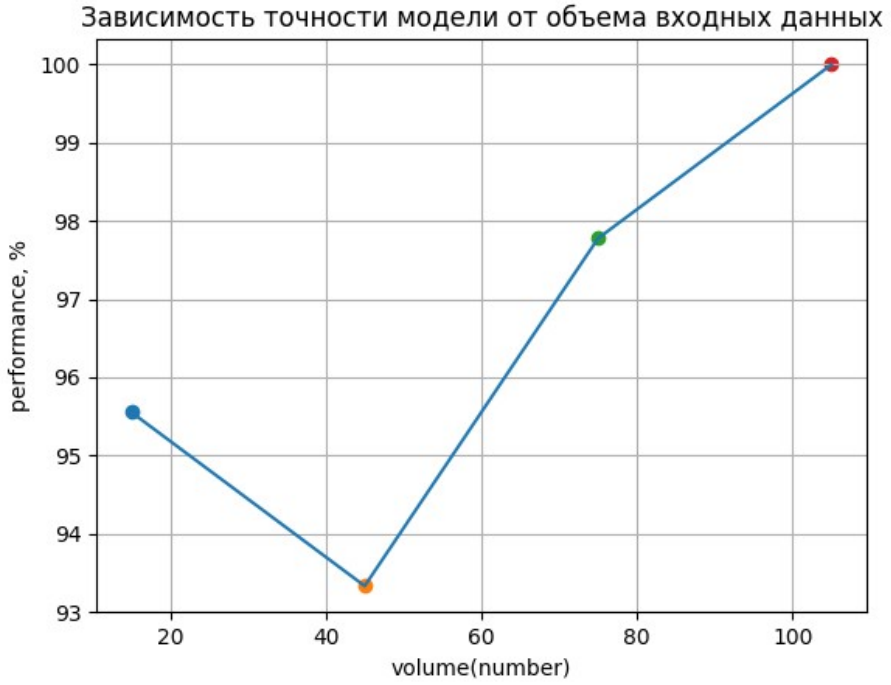
Проанализируем теперь результат работы нейросети на датасете MNIST. Параметрами выступали рейтинг равный 0.25, количество эпох равное 25 и размер скрытого слоя равный 100. Я эти параметры не менял, потому что датасет большой очень...

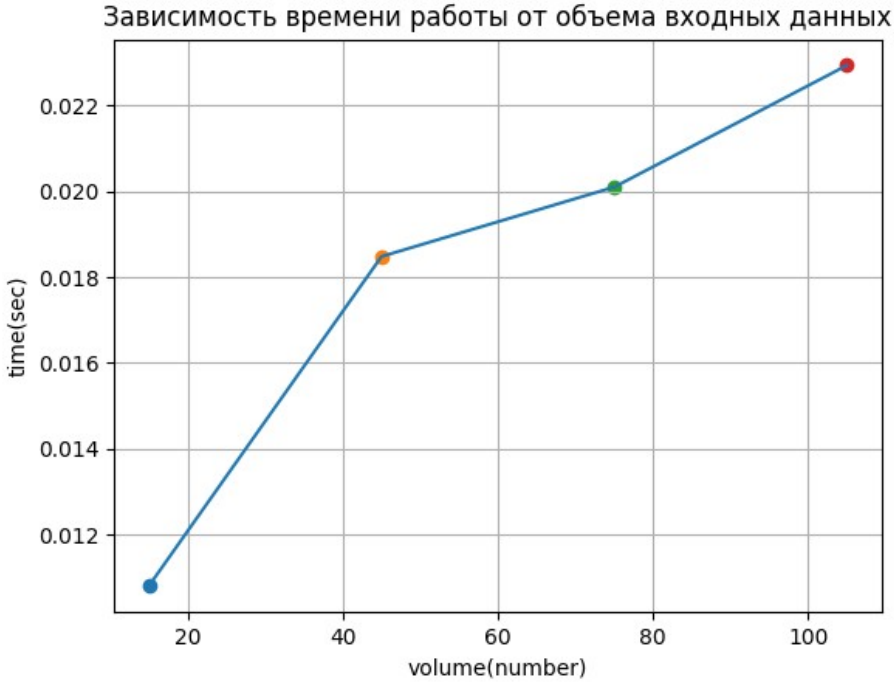




Во-первых видим сильное отличие от датасета ирисов в времени работы. Если для ирисов модель работала 7 секунд при 35 эпохах обучения, то тут при 25 эпохах время работы для самого большого входа уде 10 минут, почти в 85 раз больше. Однако и датасет больше в 400 раз. Я думаю, что на время работы сильно сказывается большая размерность пространства признаков, ведь картинку 28 на 28 пикселей мы представляем в виде вектора длиной 784. Ну и количество входящих данных, естественно тоже, ведь на рисунке прямая зависимость представлена. Если посмотреть на 1-ый график, то наблюдаем ситуацию похожую с картинкой для ирисов. т. е. При относительно небольшом входе модель показывает неплохой результат, но если для ирисов результат вообще был лучшим, то тут все таки лучший результат модель показала с самым былышим входом в нее. На самом деле результат модели как-то неудовлетворяет. Что можно было бы доработать так это то, что параметры были выбраны почти случайно, потому что на домашнем компукторе запускать при разных параметрах алгоритмы которые работают больше 10 минут тяжеловато. Ну а еще лучше так это использовать некие сверточные нейронные сети, которые как-то круто работают для изображений и всего такого.

Дальше рассмотрим результат работы алгоритмы k-ближайших соседей. Сначала на примере ирисов. Разбиение всюду одинаковое, т. е. такое же как для нейросети.

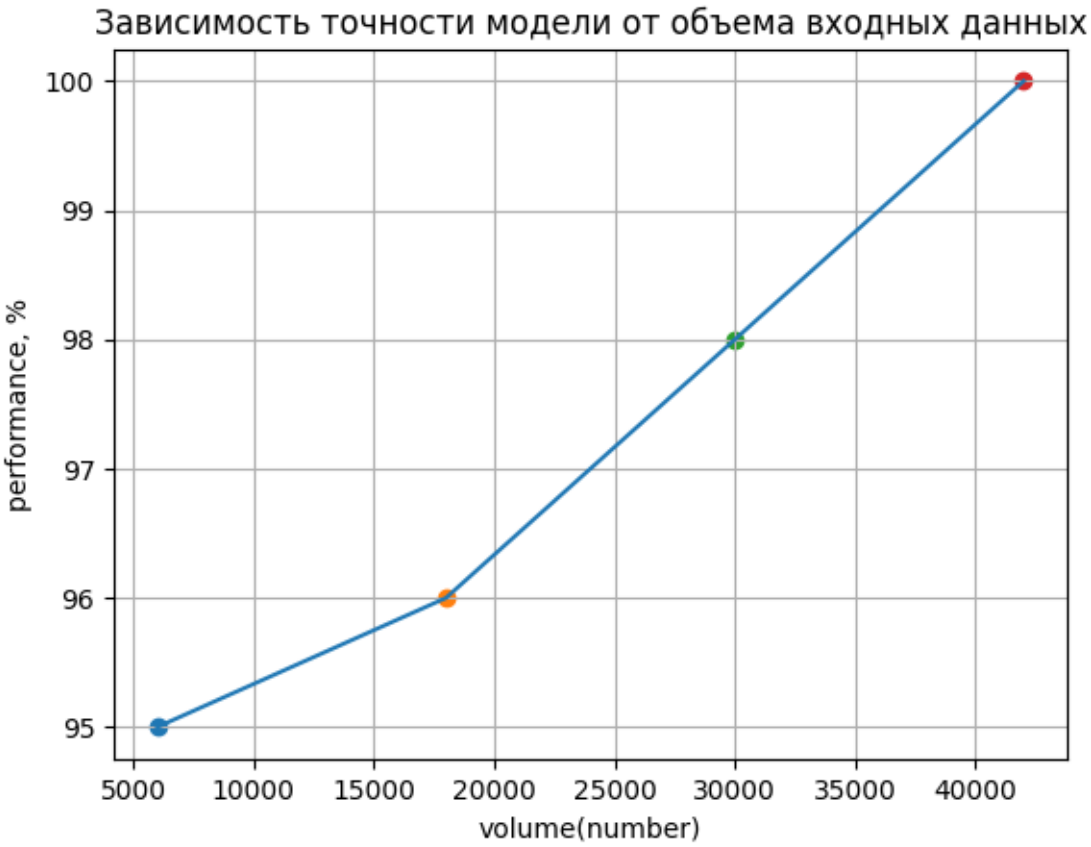


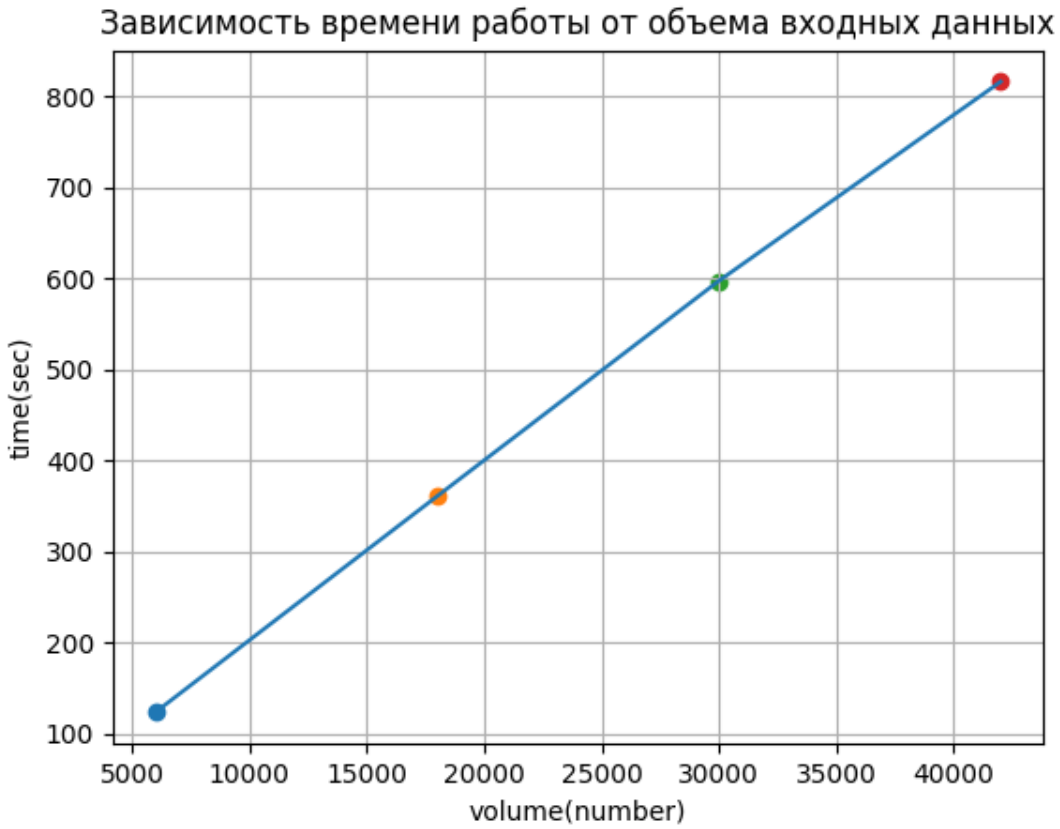


Ну тут кажется все просто: чем больше примеров, тем лучше результат. Возможно, если мало примеров, то слово ближайшее в названии алгоритма немного(или даже много) искажается, т. е. Если на самом деле существует много прям БЛИЖАЙШИХ соседей, на основании которых можно что-то предсказать, а мы этих соседей

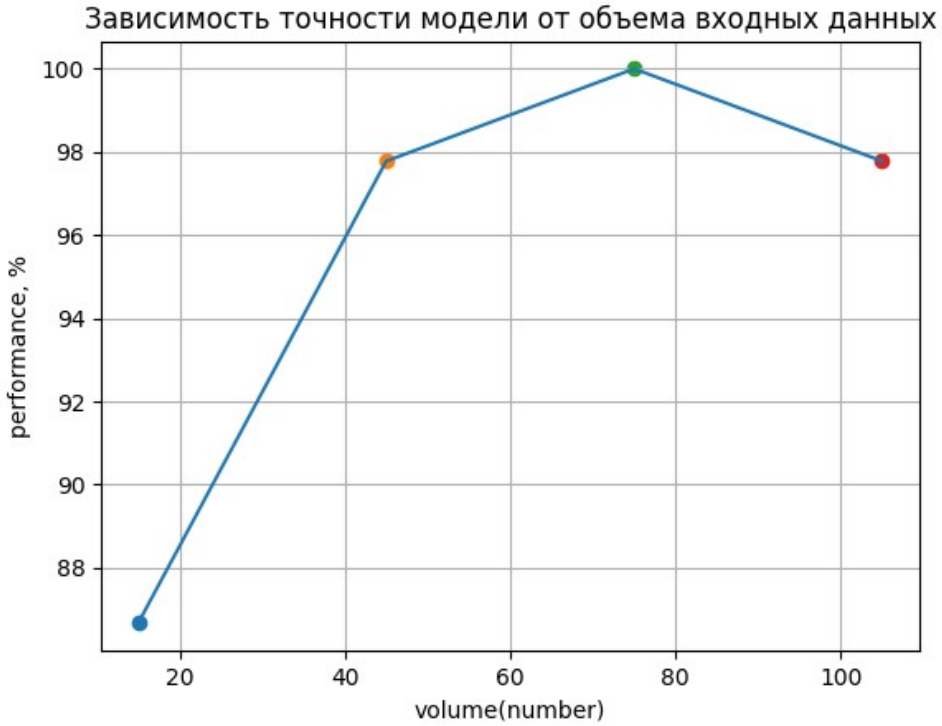
выбрасываем и рассматриваем какую-то горсть, никак или почти никак не связанных, объектов, то понятно, что результат должен быть хуже. Возможно, это не всегда правда, но применимо к этому датасету. Хотя при маленьком входе, результат все равно неплох. Что стоит еще отметить, так это то, что данная модель работает сильно быстрее нейросети.

Рассмторим теперь работу модели на MNIST датасете.





Тут ситуация аналогичная с датасетом ирисов. Чем больше примеров, тем лучше результат. Модель работает дольше нейросети, но дает результат лучше. Я думаю большое преимущество алгоритма k-ближайших соседей это простота реализации и всего лишь один входной параметр(хотя наверное есть какие-то модифицированные версии), в отличии от нейросети, где нужно задать много параметром, таких как: количество эпох, скорость обучения и вообще архитектура нейросети.

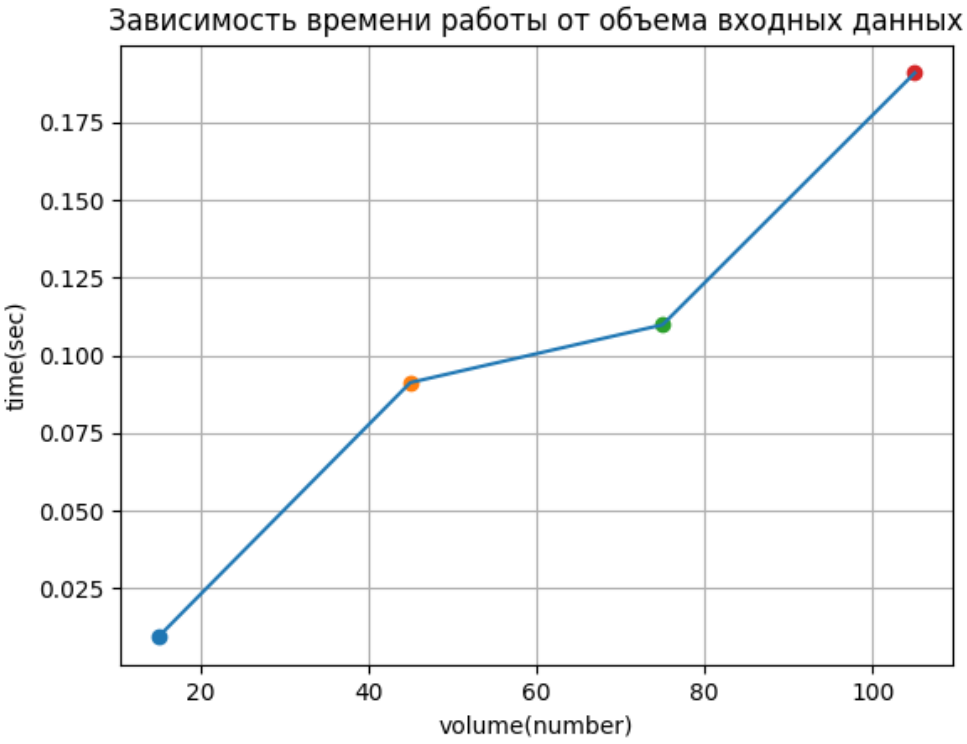


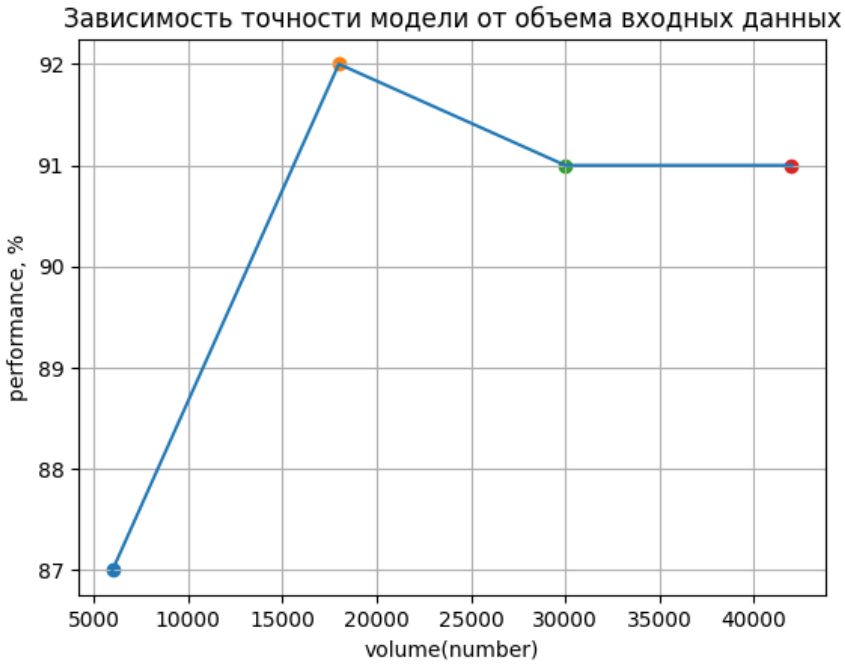
Посмотрим теперь на модель с алгоритмом K-средних. Начнем как обычно с ирисов. Обучить модель значит найти оптимальное количество кластеров.

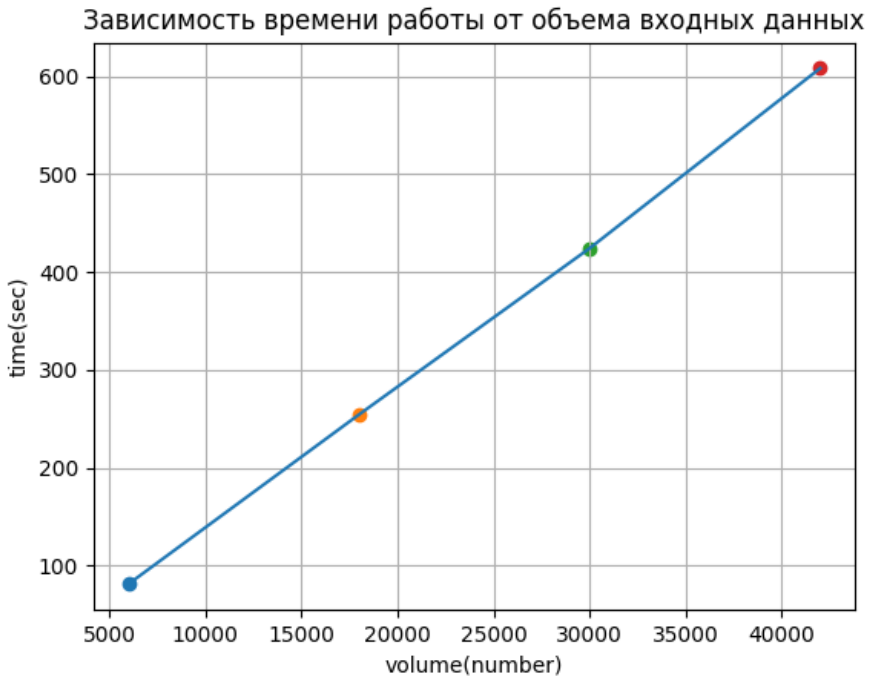
Для ирисов количество класстеров для каждого объема входа рассчитывался. Просто запускаем алгоритм начиная с 4 до 105.

Легко видеть, что при небольшом входе и результат не выдающйися. Но и при самом большом входе результат не лучше. Лучший результат затесался где-то посередине. Вывод который можно сделать — запускать использовать хотя бы 50% датасета ирисов для обучения. Скорость работы быстрее чем у сети, но медленее чем KNN.

Посмотрим теперь на то, как алгоритм справился с MNIST. Подбирать параметр числа кластеров крайне тяжело(компуктеру), поэтому я просто начал с 10 и до 500 с шагом 100 запускал алгоритм. При K = 484 алгоритм работал 20 минут… Когда же я тестировал алгоритм для разных объемов входа я положил K = 300.

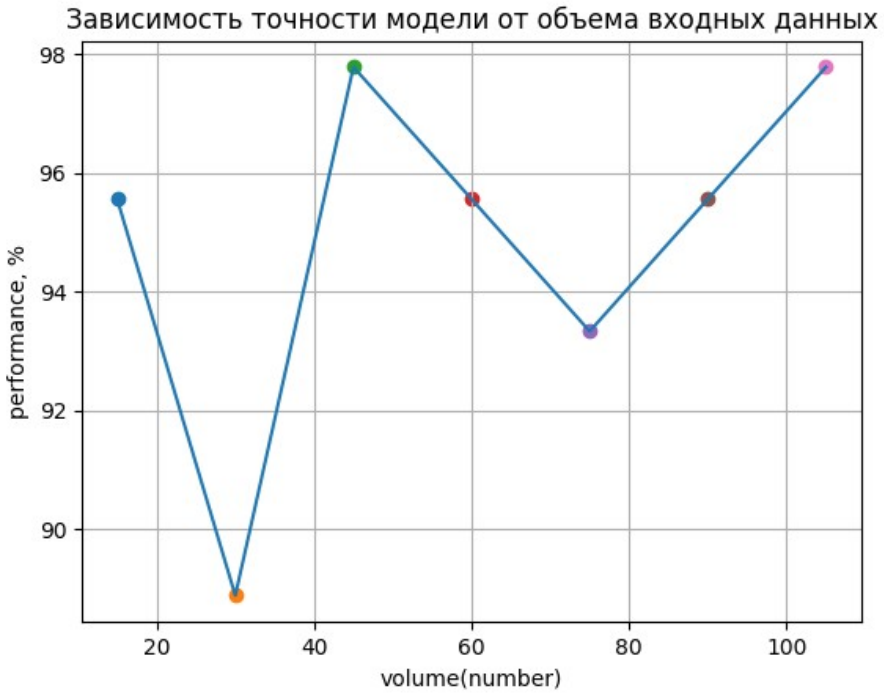




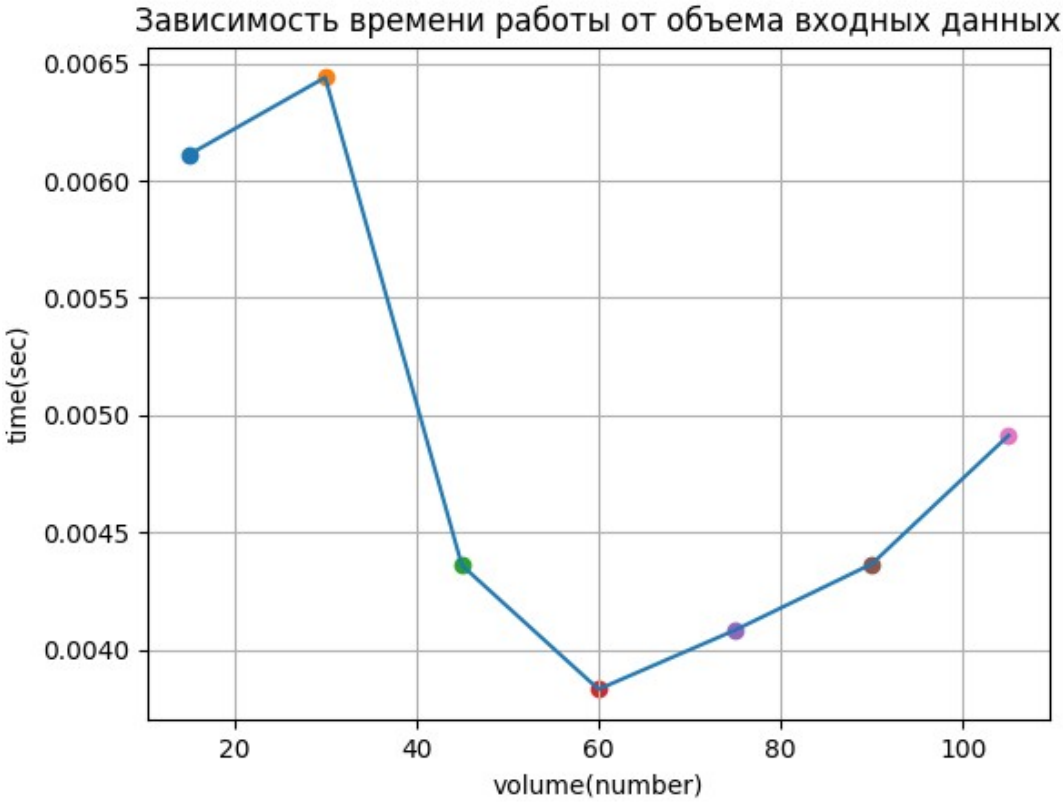


Результат работы не радует, всего лишь 92% и 10 минут работы. Это худший результат на данный момент. Кажется этот алгоритм тоже чувствителен к объему входных данных, как KNN, но если предположить, что между 30000 и 42000 тысячами не происходит каких-то невероятных скачков, то наступает момент, когда алгоритм совсем, ну или почти совсем, не различает эти ситации и тогда нет смысла давать ему больше, чем он может съесть. На самом деле я думаю, что нельзя строго судить о этой модели, потому что много зависит от количеств кластеров, потому что также была замечена зависимость, что чем больше кластеров, тем лучше результат. По крайней мере эта логика применима к отрезку от 10 до 500. Если положить K = 1000, то результат, возможно, будет лучше. Да и вообще задача класстеризации, кажется, более сложной нежели просто классификация. А и еще к тому же на время работы, по-моему, сильно влияет то, как изначально были определеные центры класстеров. Можно сделать как-то по-умному, но у меня они задаются случайно. Так тоже делают, но это вносит свои коррективы.

Остался только Наивный Байесовский классификатор. Начнем с ирисов:

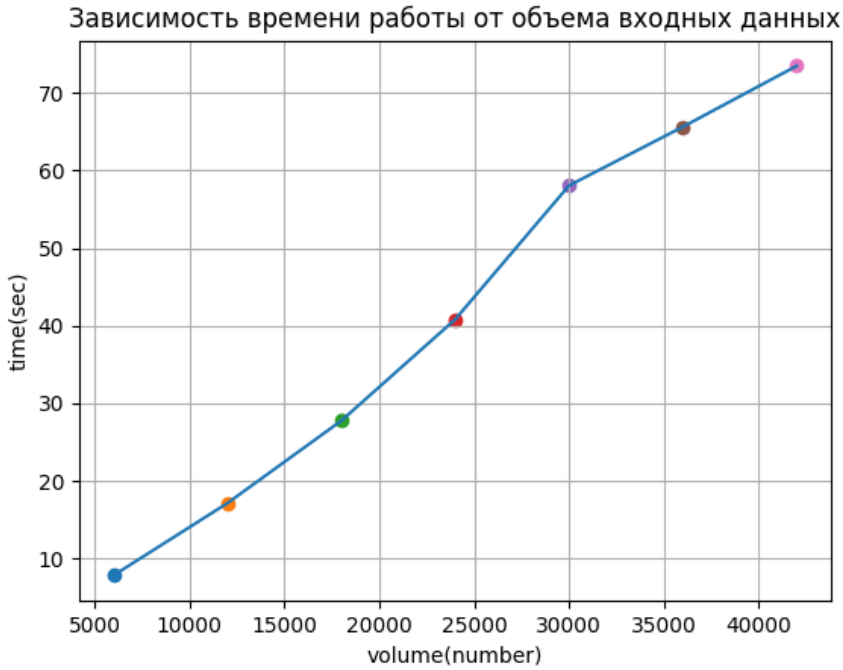


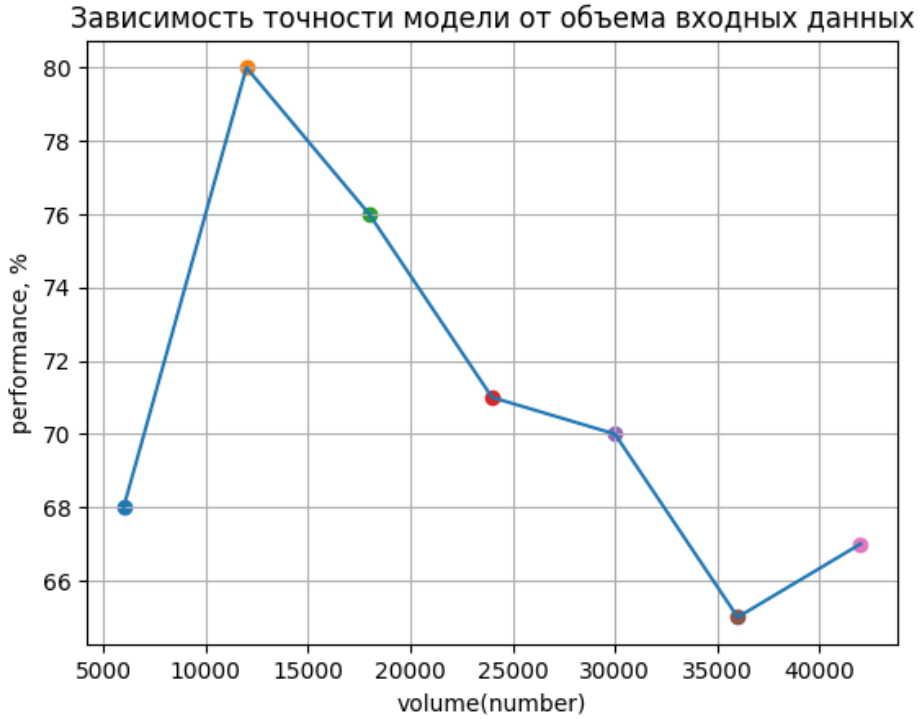
Выглядит очень интересно. Рискну предположть, что от объема входа вообще ничего не зависит. Алгоритм и при больших и при небольших входах может показывать себя и хорошо и плохо.



Скорость работы алгоритма просто потрясающая! Ну оно и не удивительно, ведь все что нужно сделать это вычислить некоторые статистики.

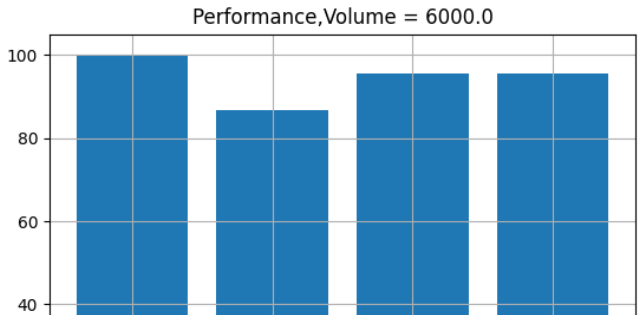
Обратимся к MNIST датасету

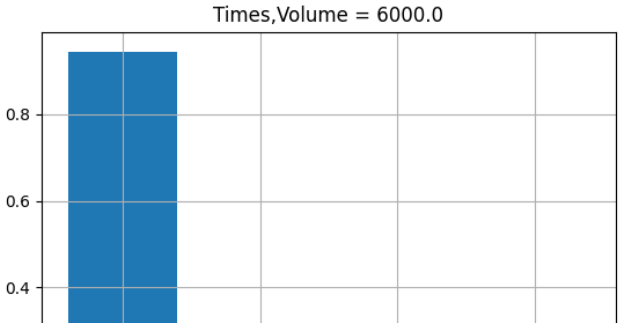


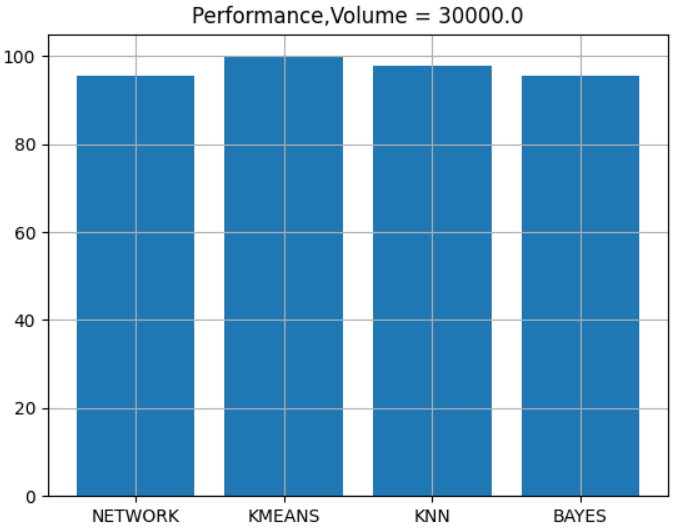


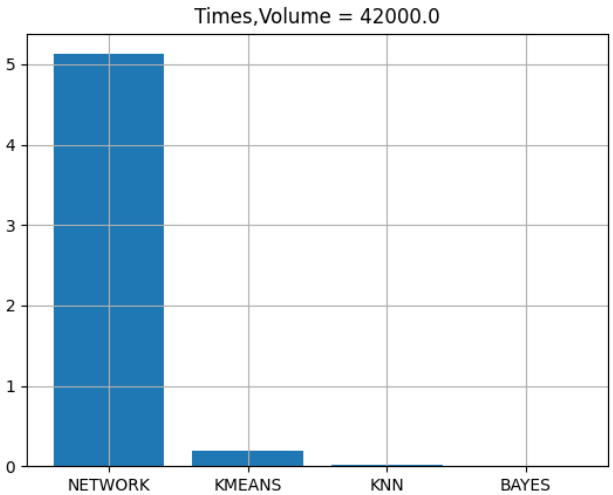
В алгоритме для MNIST я столкнулся с проблемой которую пока что не решил. Проблема заключается в том, что статистики типа стандартного отклонения и среднего вычисляются по признакам. Признаки у нас это числа от 0 до 1 после нормализации. Но дело в том, что на самом деле существуют полность нулевые признаки поэтому и среднее и стандартное отклонение будут нулевыми, но в расчете предсказания используется формула для вычисления функции распределения вероятностей, унас это функция Гаусса, происходит деление на ноль. Возможно для решения нужно полность выкинуть нулевые столбцы, или полагать стандартное отклонение равным какой-то константе. В текущем варианте я положит стандартное отклонение равным 1. Скорее всего график точности неправда, потому что копится ошибка из-за неправдивого стандартного отклонения.

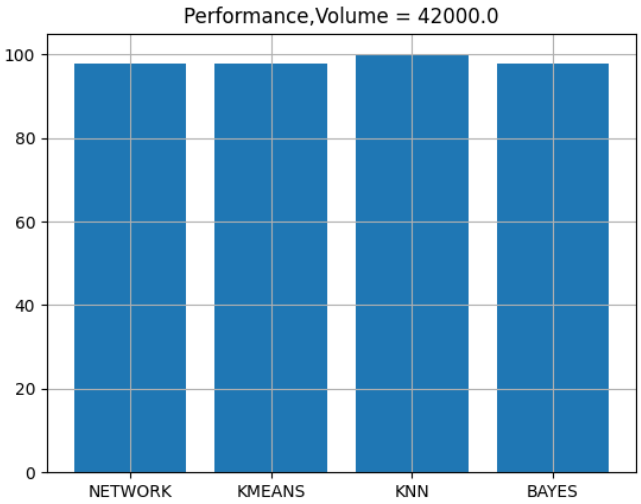
Представлю теперь для каждого разбиения данных сравнение точности и времени(тут ошибка нужно чтобы сверху было 15, 45, 75, 105):

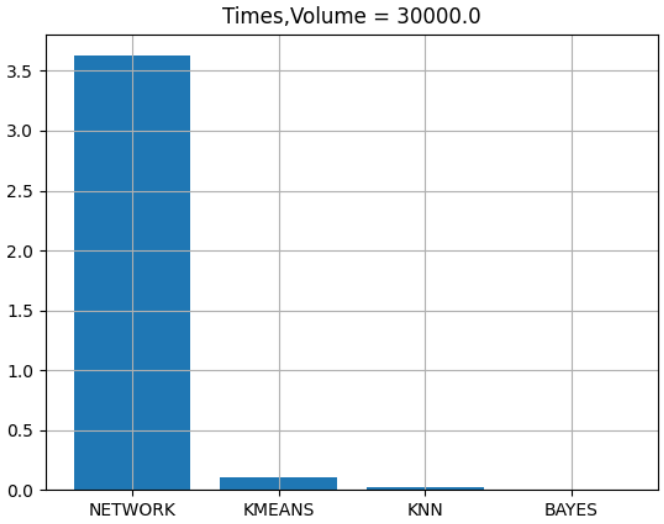










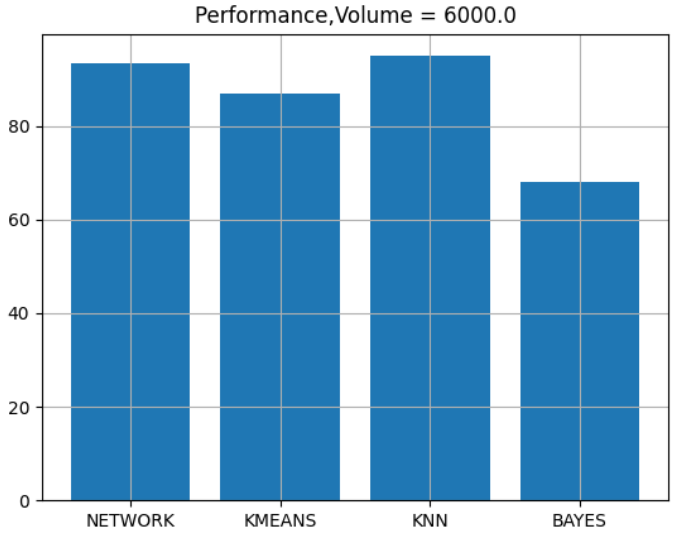


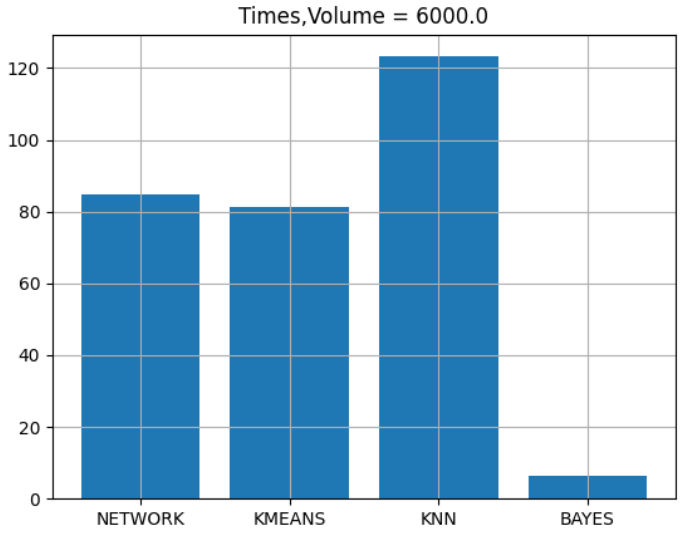
Делаем выводы касательно ирисов(да и вообще похожих на этот датасет датасетов):

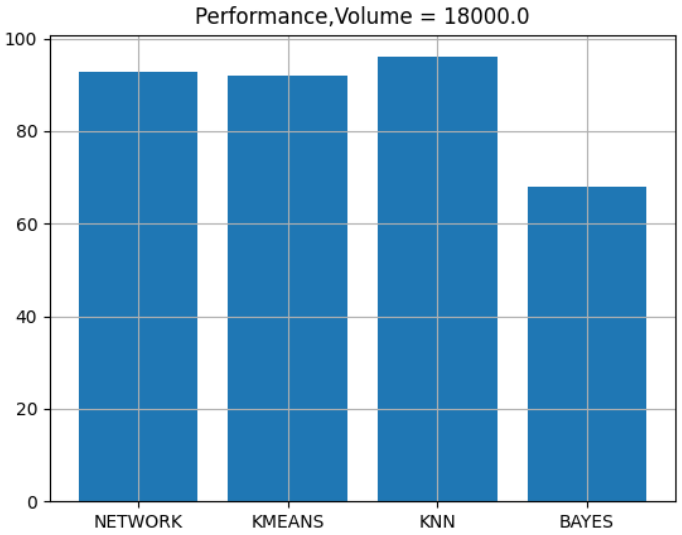
1) Если у нас совсем немного данных, то следует использовать нейросеть, потому что она показывает лучший результат и работает меньше 1 секунды.

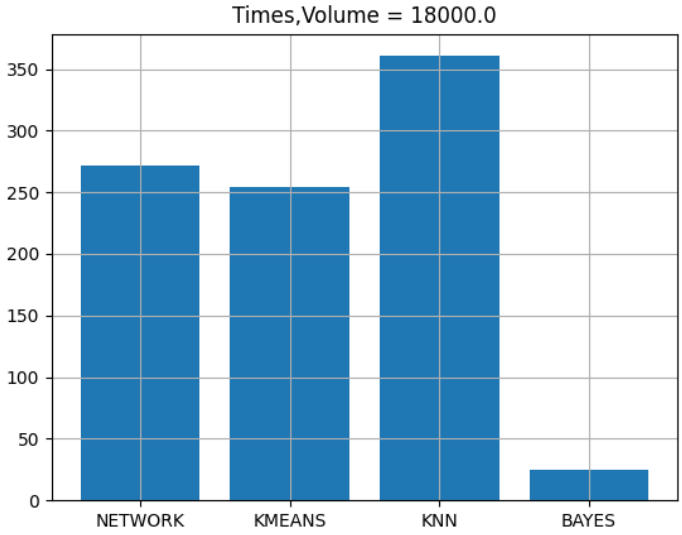
2) Во всех остальных случаях лучше воспользоваться алгоритом KNN, или K-Means, поскольку время работы сильно меньше чем у нейросети, а результат не хуже, а то и лучше.

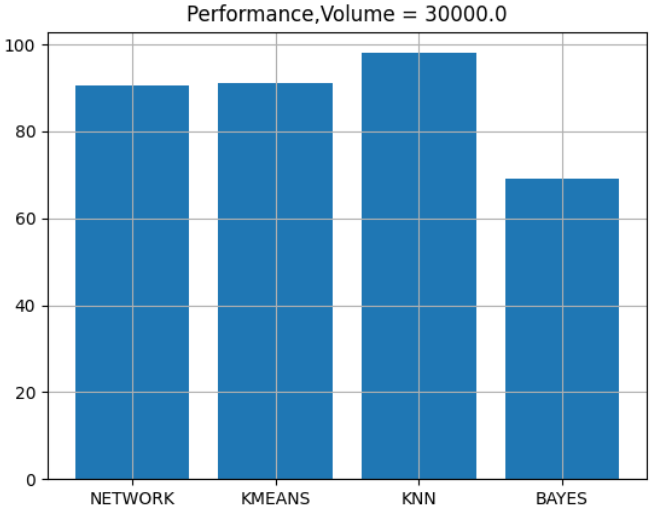
Теперь на основе MNIST:

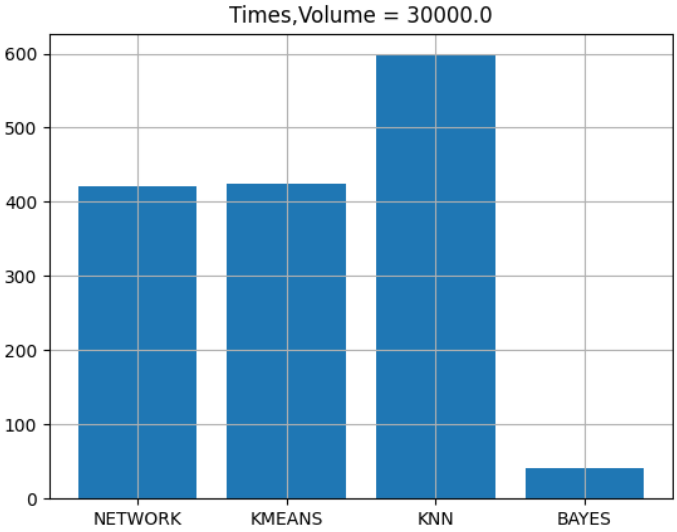


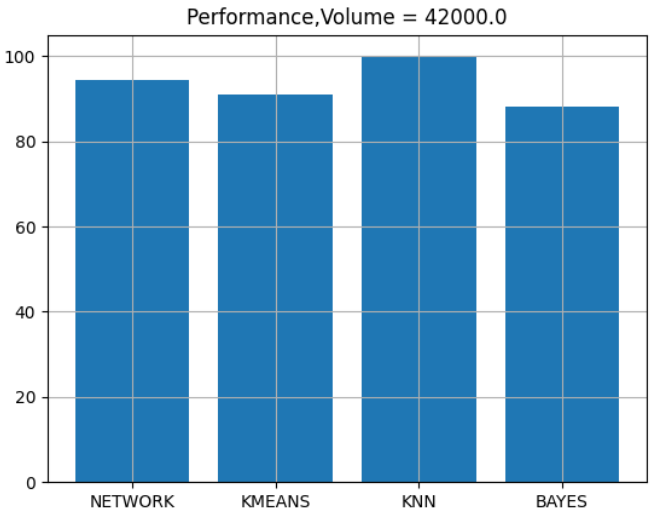


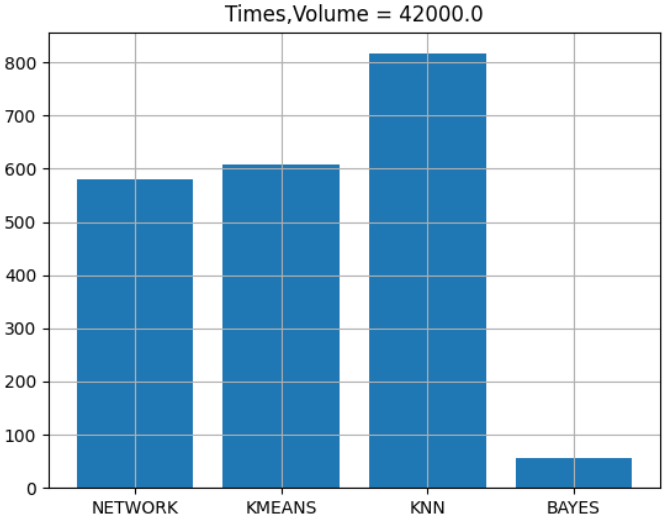

Performance,Volume = 18000.0
100
80
60
40
20
0
NETWORK
KMEANS
KNN
BAYES


Times,Volume = 18000.0
350
300
250
200
150
100
50
0
NETWORK
KMEANS
KNN
BAYES


Performance,Volume = 30000.0
100
80
60
40
20
0
NETWORK
KMEANS
KNN
BAYES


Times,Volume = 30000.0
600
500
400
300
200
100
0
NETWORK
KMEANS
KNN
BAYES


Performance,Volume = 42000.0
100
80
60
40
20
0
NETWORK
KMEANS
KNN
BAYES


Times,Volume = 42000.0
800
700
600
500
400
300
200
100
0
NETWORK
KMEANS
KNN
BAYES

Выводы:

1) в текущей реализации алгоритмов KNN всегда показывает лучший результат в плане точности предсказания, но самое худшее время работы.

2) Байесовский классификатор(сломаный для этого датасета), не показывает высоких результатов, но работает гораздо быстрее всех алгоритмов, что на самом деле большое его преищуество, т. к. в такой алгоритм можно закинуть очень много данных. Не факт, что алгоритм будет работать лучше, но по крайней мере такая возможность есть.

3) Нейронная сеть с заданными выше параметрами и метод k-means работают примерно одинаково на всех объемах входной информации.