Московский государственный университет им М. В. Ломоносова
Физический факультет

Курсовая работа

Сравнительный анализ Алгоритмов машинного обучения

Выполнил:
Студент 216 группы
Каримов В.А
Научный руководитель:
д.ф.-м.н., доц. Голубцов П. В.

Москва
2024

Оглавление

Введение

В современном мире алгоритмы машинного обучения и нейронные сети являются ключевыми инструментами для многих компаний, которые стремятся создать прогнозы, оптимизировать процессы и принимать обоснованные решения на основе данных. Эти технологии играют важную роль в различных областях, таких как рекомендательные системы, обработка естественного языка, компьютерное зрение и прогнозирование.

Для проведения исследования по прогнозированию цены на товар, в данной работе будет использоваться два основных пакета: Scikit-learn и Pytorch. Они предоставляют широкий спектр алгоритмов машинного обучения и инструменты для предобработки данных, такие как разделение датасета на обучающую и тестовую выборки, а также преобразование номинальных признаков в числовые.

Целью данного исследования является сравнение трех различных алгоритмов машинного обучения на примере датасета с информацией о характеристиках алмазов. Проведение сравнительного анализа позволит выявить эффективность и применимость каждого из алгоритмов в задаче прогнозирования цены на товар, что может быть полезно для бизнеса и научных исследований.

Теоретическая справка

2.1 Линейная регрессия

Задача линейной регрессии (*1*) - задача в m-мерном пространстве, где $m-1$ признак элемента и 1 свойство, которое требуется предсказать, основная цель этой задачи — проверка наличия линейной зависимости свойств от признаков элемента и построение гиперплоскости от $m-1$ координат, которая будет однозначно определять значение свойства.

Множество точек линейных поверхностей отвечают уравнению плоскости:

$$f_w(x) = \sum_{i=1}^{m} x_i w_i + \omega_0 = (\vec{x}, \vec{w}) + \omega_0 = 0 \tag{2.1}$$

Очевидно, что можно выразить одну из координат через $m-1$ остальных, тогда будем считать x_m - нашим свойством объекта, а остальные координаты - признаками объекта. Перепишем наше тождество в следующем виде:

$$x_m = -\frac{w_1}{w_m}x_1 - \frac{w_2}{w_m}x_2 - ... - \frac{w_{m-1}}{w_m}x_{m-1} - \frac{w_0}{w_m}$$

Переобозначим:

$$\alpha_i := -\frac{w_i}{w_m}$$

Тогда искомое предсказание будет выглядеть вот так:

$$a(\vec{x}) = \sum_{i=1}^{m-1} \alpha_i x_i + \alpha_0, \tag{2.2}$$

где $a(x)$ - свойство элемента как функция признаков, $\overline{x}$ - вектор признаков, а $\overline{\alpha}$ - вектор весов модели.

Рассмотрим искусственный пример. Пусть x - масса алмаза, y - степень его прозрачности, тогда если цена линейно зависит от этих параметров, то мы получим следующий график:

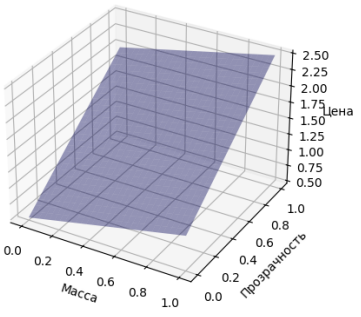


Рис. 2.1: Гиперплоскость, определяющая цену от массы и прозрачности

Так как в задаче будет использоваться не один элемент, удобно перейти к матричному виду. Пусть $X = \{x_{ij}\}$ - матрица, x_{ij} элемент которой является j-ым признаком i-ого

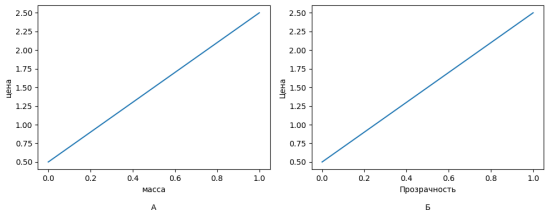


Рис. 2.2: Тот же график, но в проекциях на оси

объекта, размерности $M \times N$, в этом случае M - количество объектов, N - количество признаков объекта. Чтобы получить предсказание модели для i-ого элемента, надо умножить i строчку матрицы X на вектор весов $\overline{\alpha}$.

Теперь обсудим алгоритм обучения модели. Введем вектор $\overline{y}$ - вектор правильных ответов, ошибку модели будем оценивать с помощью метода наименьших квадратов. Тогда наша задача — минимизировать квадратичную форму Q суммарной ошибки по датасету.

$$Q(\overline{\alpha}, x, \overline{y}) = \frac{1}{M}||X\overline{\alpha} - \overline{y}||^2 \tag{2.3}$$

2.2 Решающее дерево

Следующим алгоритмом, исследуемым в данной работе, является алгоритм решающего дерева. (*2*) По своей сути алгоритм решающего дерева эквивалентен звонку в службу поддержки: оператор задает вам несколько вопросов, постоянно углубляясь в проблему и уходя от ненужных тем, например, если проблема в перегоревшей лампочке, то на поздних этапах вы не встретите вопрос: "Какого цвета вода в кране?". Если построить все возможные вопросы и варианты ответа на них в виде схемы, получится нечто, напоминающее дерево, с точки зрения формальной математики это называется граф, и выглядит следующим образом:

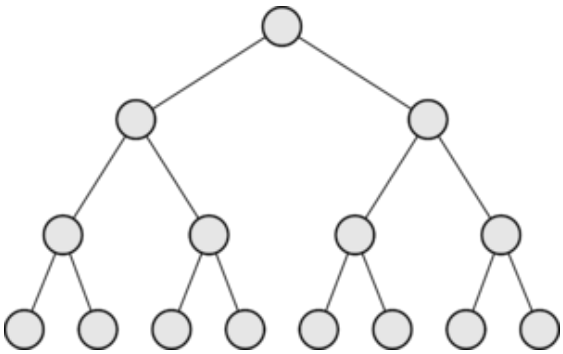


Рис. 2.3: Схема решающего дерева

Вершины графа деляется на 2 вида: внутренние и листовые — мы последовательно перемещаемся по вершинам, пока не дойдем до листовой, откуда уже нельзя перемещаться дальше. Алгоритм предсказания выглядит следующим образом:

1. Пока находимся не в листовой вершине, можно задать вопрос с ответами $True$ или $False$
   (a) Если ответ $True$, идем в левую вершину
   (b) Иначе идем в правую вершину
2. Если вершина листовая, выводим c — значение, хранящееся в этой вершине, оно же является результатом работы алгоритма

Обратимся к теории представленной в (3). Данные обучающие векторы $x_i \in R^n$, i = 1,. . . ,l и вектор-метка $y \in R^l$ дерево решений рекурсивно разбивает пространство признаков таким образом, что образцы с одинаковыми метками или аналогичными целевыми значениями группируются вместе.

Пусть данные в узле m быть представлен Q_m с участием N_m образцов. Для каждого раскола кандидатов $\theta = (j, t_m)$ состоящего из функции j и порога t_m, разделите данные на $Q_m^{left}(\theta)$, а также $Q_m^{right}(\theta)$ подмножества

$$Q_m^{left}(\theta) = (x, y)|x_i <= t_m$$

$$Q_m^{right}(\theta) = Q_m \backslash Q_m^{left}(\theta)$$

Качество кандидата разделения узла m затем вычисляется с использованием функции примеси или функции потерь $Loss()$, выбор которых зависит от решаемой задачи.

Рекурсия для подмножеств $Q_m^{left}(\theta)$, а также $Q_m^{right}(\theta)$ продолжается, пока не будет достигнута максимально допустимая глубина.

Сразу в глаза бросается очевидный вопрос: "Как можно использовать дерево решений для задачи регрессии, если количество листовых вершин конечно и дискретно, а данные в общем случае непрерывны?". В таком случае идут на хитрость и в качестве значений вершины хранят не метку класса, а некоторое значение c, определяющее набор элементов, отвечающий данной вершине. Наша задача — минимизировать некоторую функцию потерь между предсказываемым значением и хранящимся в вершине значением, возьмем так же средне квадратичную ошибку:

$L = (y_i - c)^2$, где y_i - свойство элемента, c - среднее значение вершины

Тогда ошибка по всем элементам будет:

$$Loss(c, y_i) = \frac{1}{M} \sum_{i=1}^{M} (y_i - c)^2 \to \min_c \tag{2.4}$$

Несложно доказать, что минимум функции потерь будет достигаться при $c = \overline{y}$, где $\overline{y}$ - среднее значение предсказываемого признака по всем элементам набора, отвечающим данной вершине.

2.3 Метод главных компонент

2.3.1 Теория

В современном мире данные бывают очень большими, обычно это касается, например, изображений, такие данные часто являются избыточными для обучения модели, то есть можно уменьшить разрешение изображения, при этом не потеряв точность работы алгоритма, но сэкономив довольно большой промежуток времени на обучение. Поэтому прибегают к алгоритмам понижения размерности, один из таких и рассмотрим — метод главных компонент (Principal component analysis, PCA).

Вообще говоря, PCA используется не только для понижения размерности, но и для фильтрации шума, выделения главных признаков и генерации новых;в нашей работе рассмотрим только случай понижения размерности в связи с «удачностью» датасета. В методе главных компонент (PCA) мы предполагаем, что важность определенного набора признаков объектов можно оценить по тому, насколько разнообразны значения этих признаков после проекции данных на новое пространство. То есть, мы стремимся найти подпространство, на котором данные имеют наибольшую дисперсию, считая это наиболее информативным для описания их структуры.

С точки зрения математики мы пытаемся закодировать точку $x_i \in R^n$ из набора точек $\{x_1, x_2, ..., x_n\}$ так, чтобы ей однозначно отвечала точка $c_i \in R^m$, где $m<n$. Для этого нужно найти 2 функции: функцию кодирования $c_i = f(x_i)$ и функцию декодирования $x_i \approx g(c_i) = g(f(x_i))$, пусть $g(\mathbf{c}) = D\mathbf{c}$, где $D \in R^{n \times m}$. Поиск оптимального декодера может оказаться трудной задачей, поэтому на матрицу D накладываются некоторые ограничения, во-первых, столбцы матрицы ортогональны (однако это не делает матрицу ортогональной),

во-вторых, норма всех столбцов равна 1. Для работы алгоритма необходимо сначала определить способ создания оптимальной кодовой точки c* для каждой исходной точки x. Один из подходов заключается в минимизации расстояния между x и ее восстановлением g(c*). Для измерения этого расстояния используется какая-либо норма. В методе главных компонент используется норма L^2.

$$c^* = argmin_c ||\mathbf{x} - g(\mathbf{c})||$$

Оказывается, что получается эффективный алгоритм при $c = D^T\mathbf{x}$, тогда $r(\mathbf{x}) = D\mathbf{c} = DD^T\mathbf{x}$, где $r(x)$ - реконструкция, задача обучения алгоритма PCA —- это поиск оптимальной матрицы D такой, чтобы минимизировать функцию потерь, которая выглядит следующим образом (*4*):

$$loss = ||\mathbf{x} - DD^T\mathbf{x}||^2$$

На следующей странице представлен прикладной подход к рассмотрению алгоритма PCA.

2.3.2 Практический подход

Теперь рассмотрим практическое применение этого алгоритма на искусственном датасете: генерируем облако точек, лежащих на прямой с некоторым случайным отклонением, как на рисунке 2.4.

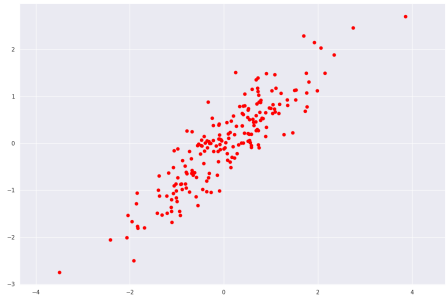


Рис. 2.4: Случайное облако точек

Далее применим к этому облаку алгоритм PCA, реализованный в sklearn, и отобразим компоненты, которые выделил PCA, по своей сути PCA нашел новый базис, в котором это облако точек обладает наибольшей дисперсией, видно, что одну из координат можно выбросить без реальной потери информативности.

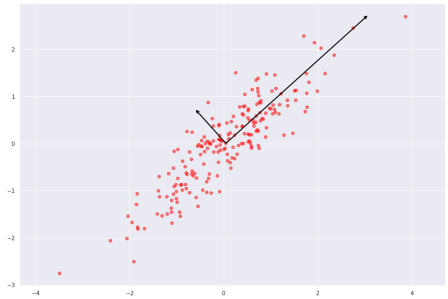


Рис. 2.5: Случайное облако точек с новым базисом

2.4 Полносвязная нейронная сеть

Простейшая сеть состоит из слоев нейронов, она является функцией переводящей $\mathbf{x} \in R^n$ в $\mathbf{y} \in R^m$, где n - количество сигналов (размерность объекта), m - размерность выходного вектора, в нашем случае $n = 9, m = 1$. (*2*) Рассмотрим модель одиночного нейрона, на вход подается некоторый вектор $\mathbf{x}$, на выходе получаем линейную комбинацию значений, поданных на вход и затем преобразованных функцией активации.

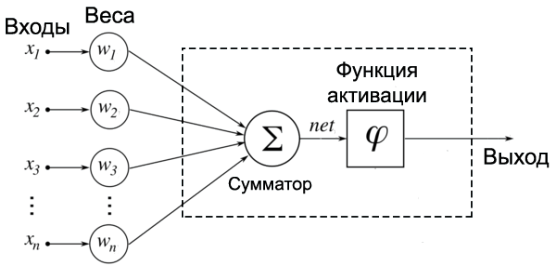


Рис. 2.6: Схема одиночного нейрона

Наиболее популярными функциями активации являются функции $tanh(x)$, $sigmoid(x)$ и $ReLU(x)$, графики которых представлены ниже. Главная особенность функций активации - они нелинейны, иначе все слои сети можно было бы объединить в одну линейную комбинацию.

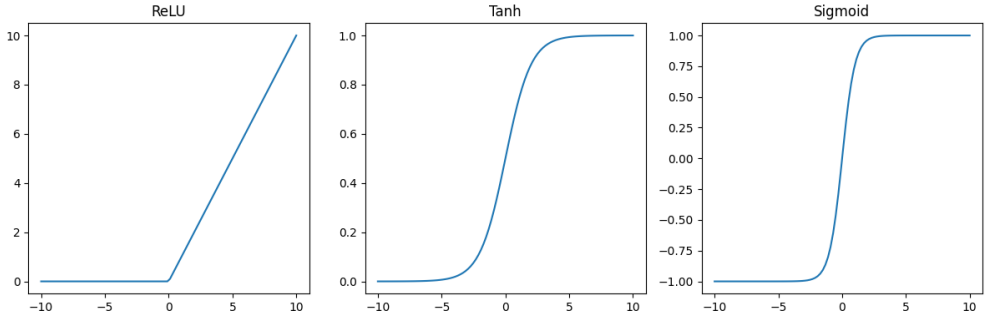


Рис. 2.7: Графики функций активации

Теперь рассмотрим полносвязную нейронную сеть. Понятие «полносвязная» содержит в себе следующую структуру:

- Существует несколько слоев
  - Входной
  - Выходной
  - Несколько скрытых слоев
- Каждый нейрон слоя связан с каждым нейроном следующего слоя
- Набор весов всех отдельно взятых нейронов образует матрицу весов, подбор оптимальной матрицы весов называется процессом обучения сети

На рисунке 2.8 представлена схема полносвязной сети с двумя скрытыми слоями, в настоящей работе использовалась сеть как раз с двумя полносвязнными слоями в связи с тем, что размерность входных данных всего 9 признаков:

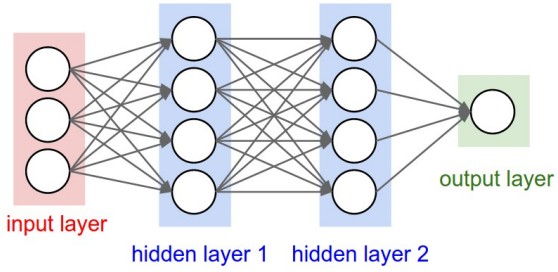


Рис. 2.8: Схема полносвязной нейросети

Результаты работы алгоритмов

3.1 Постановка задачи и предварительная обработка

3.1.1 Постановка задачи

Поставим задачу прогнозирования цены алмазов. Имея 2 набора данных (тестовую и обучающую выборки) научиться прогнозировать цену минералов из тестовой выборки с помощью алоритма, построенного на обучающей выборке. Точность предсказаний будем оценивать по известным меткам их тестовой выборки.

3.1.2 Преобработка данных

Рассмотрим датасет с характеристиками алмазов, на рисунке 3.1 представлен вид, в котором хранятся данные,сразу бросается в глаза то, что есть категориальные признаки, например цвет и огранка минерала, которые заданы словами или аббревиатурами, а также численные признаки.

К сожалению, алгоритмы неспособны работать с текстовыми данными в исходном их виде, требуется сначала закодировать текст, для этого воспользуемся классом $LabelEncoder()$, определенным в библиотеке $Sklearn$, он позволяет закодировать слова с помощью конечного набора численных признаков.

```
"","carat","cut","color","clarity","depth","table","price","x","y","z"
"1",0.23,"Ideal","E","SI2",61.5,55,326,3.95,3.98,2.43
"2",0.21,"Premium","E","SI1",59.8,61,326,3.89,3.84,2.31
"3",0.23,"Good","E","VS1",56.9,65,327,4.05,4.07,2.31
"4",0.29,"Premium","I","VS2",62.4,58,334,4.2,4.23,2.63
"5",0.31,"Good","J","SI2",63.3,58,335,4.34,4.35,2.75
"6",0.24,"Very Good","J","VVS2",62.8,57,336,3.94,3.96,2.48
"7",0.24,"Very Good","I","VVS1",62.3,57,336,3.95,3.98,2.47
"8",0.26,"Very Good","H","SI1",61.9,55,337,4.07,4.11,2.53
"9",0.22,"Fair","E","VS2",65.1,61,337,3.87,3.78,2.49
"10",0.23,"Very Good","H","VS1",59.4,61,338,4,4.05,2.39
"11",0.3,"Good","J","SI1",64,55,339,4.25,4.28,2.73
"12",0.23,"Ideal","J","VS1",62.8,56,340,3.93,3.9,2.46
"13",0.22,"Premium","F","SI1",60.4,61,342,3.88,3.84,2.33
"14",0.31,"Ideal","J","SI2",62.2,54,344,4.35,4.37,2.71
"15",0.2,"Premium","E","SI2",60.2,62,345,3.79,3.75,2.27
"16",0.32,"Premium","E","I1",60.9,58,345,4.38,4.42,2.68
"17",0.3,"Ideal","I","SI2",62,54,348,4.31,4.34,2.68
```

Рис. 3.1: Исследуемые данные

Далее обратим внимание на то, что выборка хранится целым объектом, а также весь датасет упорядочен по возрастанию цены, поэтому нам надо разделить датасет на тренировочную и тестовые части и проверить: влияет ли порядок датасета на точность, для этого воспользуемся функцией $Train\_Test\_Split$ определенной в той же библиотеке, нас интересуют 3 аргумента этой функции:

1. Аргумент $test\_size$ позволяет определить соотношение между размерами тестовой и тренировочной выборок

2. Аргумент $shuffle$ определяет будет ли датасет перемешан перед делением или нет, перемешивание играет большую роль в моделях машинного обучения.

3. $random\_state$ аргумент, позволяющий зафиксировать псевдослучайную последовательность для воспроизводимости измерений.

3.1.3 Метрика качества работы алгоритма

В настоящей работе для оценки качества прогнозов от алгоритмов использовалась метрика R^2, он же коэффициент детерминации, определяемая по следующей формуле:

$$R^2 = 1 - \frac{\sum\limits_{i=1}^{N}(f(x_i) - y_i)^2}{\sum\limits_{i=1}^{N}(y_i - \bar{y_i})^2},$$

где y_i - значение свойства элемента x_i, $f(x_i)$ - предсказание модели для элемента x_i, $\overline{y_i}$ - среднее значение всех меток свойств. По сути R^2 представляет нормированную квадратичную ошибку, которую мы вычитаем из 1, значение R^2 надо максимизировать: $R^2 = 1$ - алгоритм отработал идеально, $R^2 \in (0, 1)$ - какая-то доля верных ответов у алгоритма была, стоит также обратить внимание на то, что в силу особенностей реализации $r2\_score$ в $sklearn$ иногда возникает ситуация, при которой $r2\_score$ отрицателен. Эта ситуация говорит о крайне низком качестве регрессии, худшем, чем простое усреднение всех результатов.

3.2 Линейная регрессия

Начнем с линейной регрессии, как с самого простого и понятного алгоритма, у алгоритма линейной регрессии нет настраиваемых параметров, однако эти параметры есть у деления на обучающую и тестовую выборки.

3.2.1 PCA и линейная регрессия

Первым делом обратимся к проверке понижения размерности и выявим, от каких признаков алмазов цена зависит линейно, для этого проведем серию экспериментов, где постепенно будем добавлять по 1 признаку к обучающей выборке. Заметим, что приемлемая, для

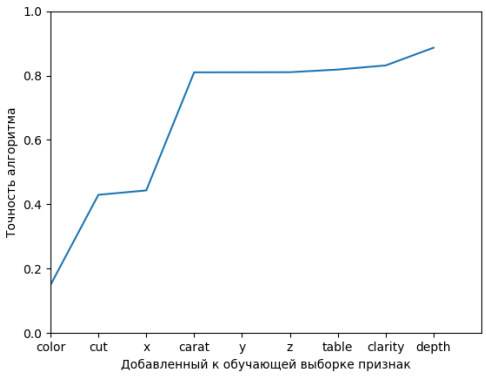


Рис. 3.2: График точности от используемых признаков

этого алгоритма, точность, а именно $R^2 \approx 0.85$, достигается уже при 4 признаках, а дальше растет лишь на пару процентов. Легко проверить, что точность увеличивается тогда, когда среди признаков появляется масса, если провести обучение алгоритма на одном признаке - массе, точность получается $R^2 \approx 0.845$. Отсюда следует, что линейной зависимости цены от категориальных признаков нет и обучать алгоритм для относительно неплохой точности достаточно на одном признаке.

3.2.2 Уменьшение обучающей выборки

Уменьшив, размерность данных, мы увеличили скорость работы алгоритма, однако есть еще способ ускорить процесс — уменьшение количества объектов в обучающей выборке. Рассмотрим два процесса, в которых будем постепенно увеличивать долю тестовой выборки и следить за точностью алгоритма.

Первый процесс будем проводить с перемешанной выборкой, то есть $shuffle = True$. Как мы видим, линейная регрессия показывает хороший ($R^2 \approx 0.8$) результат даже при раз-

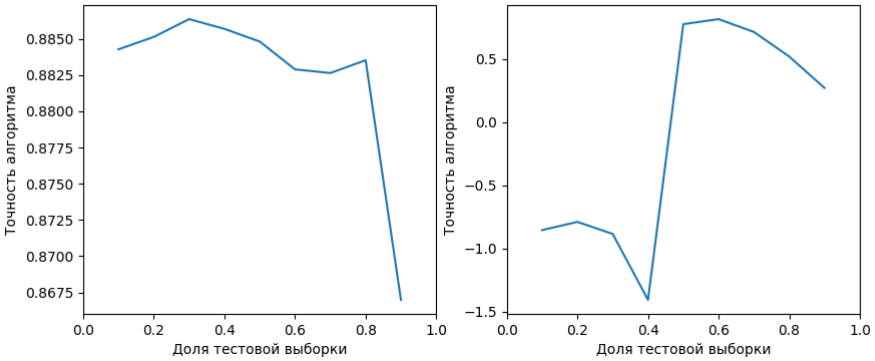


Рис. 3.3: График точности от доли тестовой выборки

мере обучающей выборки 20%, это отображает основное преимущество линейной регрессии — сильное уменьшение размеров исходных данных ведет к значительному сокращению времени работы и незначительной потери в точности.

Второй график отображает непредсказуемое поведение алгоритма при не перемешанной выборке, то есть $shuffle = False$, это означает, что при делении выборки на тестовую и обучающую в силу упорядоченности датасета попали только элементы с меньшей ценой, значит, мы начали решать задачу экстраполяции, для которой в нашем случае результат неприемлем.

3.3 Решающее дерево

Теперь перейдем к алгоритму решающего дерева, в отличие от линейной регрессии класс $DecisionTreeRegressor()$ имеет 3 аргумента:

- Максимальная глубина дерева, max_depth, определяет максимальное количество ветвлений дерева
- $Random\_state$, суть та же, что и в $Train\_Test\_Split$, фиксирует псевдослучайную последовательность для воспроизводимости измерений
- Критерий регрессии, определяет функцию потерь, которую будет минимизировать алгоритм:
  - $poisson$ - отклонение Пуассона, обладает недостатком — отрабатывает существенно дольше, чем последующие критерии
  - $squared\_error$ - классический МНК
  - $absolute\_error$ - модуль отклонения

3.3.1 Максимальная глубина

Для начала рассмотрим зависимость точности алгоритма от максимальной глубины дерева, построим набор деревьев с глубиной от 1 вершины до 100 вершин. Для сохранения масштабов на рисунке 3.4 слева представлен график точности при глубине от 1 до 10, как видно, точность растет по мере углубления дерева. На правый график рисунка 3.4 на нанесена точность для деревьев с глубиной только от 10 до 100, легко заметить, что точность растет только до $R^2 = 0.975$ при $max\_depth = 12$, сразу после этого точность начинает падает до $R^2 \approx 0.96$, очевидно случилось переобучение алгоритма, дальнейшее увеличение параметра max_depth не улучшает результат при этом значительно замедляет работу алгоритма.

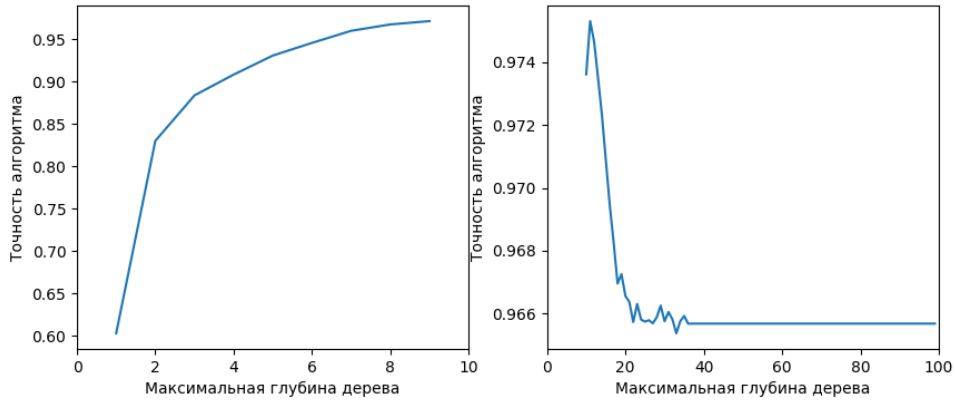


Рис. 3.4: График точности от максимальной глубины

3.3.2 PCA и решающее дерево

Теперь обратимся к PCA, как известно из предыдущей части про линейную регрессию, наиболее информативными данными оказалась масса минерала, для решающего дерева это не должно измениться, однако точность должна возрасти в силу того, что дерево способно приближать нелинейные зависимости.

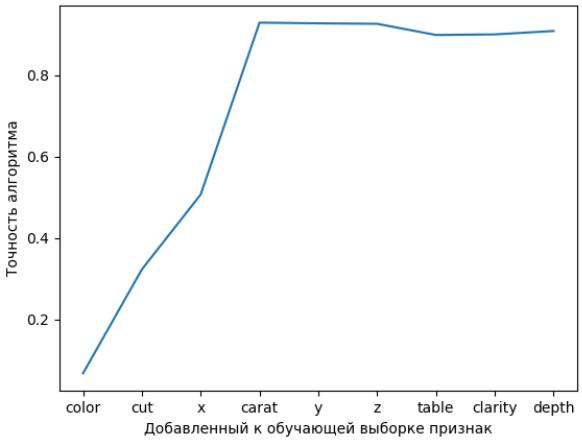


Рис. 3.5: График точности от используемых признаков

На рисунке 3.5 отчетливо видно, что предположение подтвердилось, резкий скачок точности происходит при добавлении размера x, а затем массы в набор признаков, также точность находится на уровне $R^2 = 0.91 - 0.93$. Стоит обратить внимание, что применение PCA к обучающей выборке приводит к понижению точность примерно на 5%, это связано как раз с тем, что матрица D не найдена идеально и не удалось декодировать данные без потери информации. Таким образом, оптимальное дерево решений является дерево со следующими параметрами:

- $max\_depth = 12$
- 4 признака, выделенных с помощью PCA
- $criterion = squared\_error$

Оно дает точность $R^2 = 0.928$, однако дерево с глубиной 12 вершин без применения PCA дает точность $R^2 = 0.975$, но и обучается чуть дольше. В приложении также есть итоговый вид обученного дерева, но с max_depth лишь равное 5 в силу сложности представления

изображений с большей глубиной, это дерево дает значение метрики $R^2 > 0.9$, на него можно ориентироваться, как на хорошо обученное дерево.

3.4 Полносвязная нейронная сеть

Наконец рассмотрим последний подход к решению текущей задачи — полносвязную нейронную сеть. Несмотря на большое количество теоретических преимуществ у алгоритмов глубокого обучения перед алгоритмами машинного обучения, особенно в сфере решения многомерных задач и задач, связанных с обработкой текстов и изображений, нейронные сети не всегда являются лучшим вариантом, у них есть несколько существенных недостатков:

1. Даже самая простая нейронная сеть требует достаточно много вычислительной мощности, глубокие и сложные сети, например, требуют отдельного ядра в вычислительной машине для своих расчетов, тогда как алгоритмы, рассмотренные выше, справляются с явно меньшими затратами.

2. Как следствие предыдущего пункта - время обучения значительно выше времени обучения алгоритмов машинного обучения.

3. Большое количество гиперпараметров: пока у линейной регрессии было 2 гиперпараметра: размер обучающей выборки и размерность данных, у нейронной сети добавляется возможность выбора количества эпох, функции активации, оптимизатора, а также шаг градиента.

Рассмотрим поведение простой двухслойной сети при решении задачи регрессии на минералах, все варианты обучения были проведены при доле обучающей выборки $R^2 = 0.7$ и шаге градиента $1e - 2$. Первый эксперимент представлен на рисунке 3.6, функция активации $ReLU(x)$, количество эпох 6, на рисунке находятся 2 пары графиков: точность от количества эпох и значение функции потерь от количества эпох. Видно, что точность на

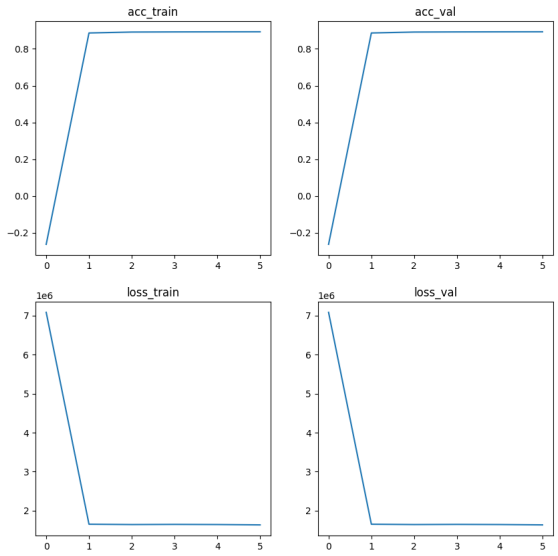


Рис. 3.6: Графики точности и функции потерь от количества эпох

обучающей выборке резко возросла уже после 2 эпохи обучения, что является очень хорошим результатом, однако после 6 эпох обучения результат на тестовой выборке оказался $R^2 = 0.896$, что с одной стороны хорошее значение, но с другой стороны линейная регрессия показала результат чуть хуже за явно меньшее время работы.

Попробуем увеличить количество эпох до 15, чтобы повысить точность модели, при этом сохранив функцию активации $ReLU(x)$.

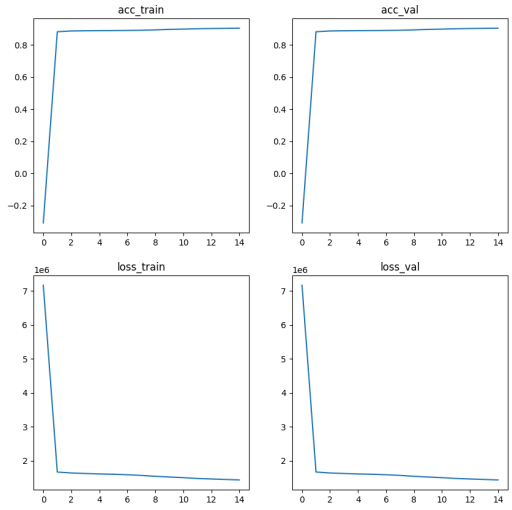


Рис. 3.7: Графики точности и функции потерь от количества эпох

Точность на обучающей выборке возросла, ровно как и на тестовой, правда, всего на 2.5% до $R^2 \approx 0.91$, однако опять видно явное превосходство алгоритмов машинного обучения, например, решающее дерево дает точность $R^2 = 0.97$ при сильно меньших затратах мощности и времени.

При замене функции активации на $sigmoid(x)$ точность упала на пару процентов до $R^2 = 0.86$ при 6 эпохах обучения, график представлен на рисунке 3.8.

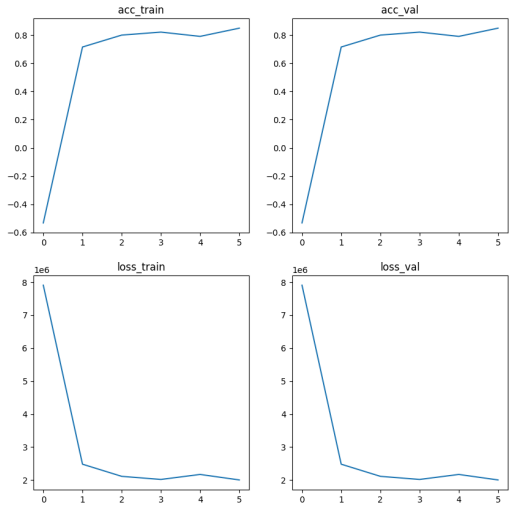


Рис. 3.8: Графики точности и функции потерь от количества эпох

При попытке заменить функцию активации на $tanh(x)$ модель переставала давать какие-либо адекватные результаты, что является крайне неожиданным поведением, ведь $tanh(x)$ должен справляться с задачей регрессии, тогда как $sigmoid(x)$ используется в основном для задачи бинарной классификации.

Последним экспериментом с нейронной сетью был запуск обучения на 150 эпох с функцией активации $sigmoid(x)$. Видно, что произошло переобучение сети еще примерно на 25 эпохе, поведение стало непредсказуемо и ожидать каких-либо вменяемых решений от такой модели не стоит.

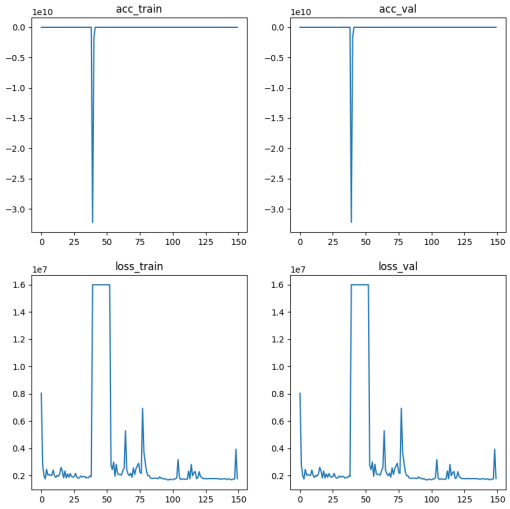


Рис. 3.9: Графики точности и функции потерь от количества эпох

Выводы

4.1 Линейная регрессия

Подводя некоторый итог о линейной регрессии, хочется сказать, что алгоритм прекрасно справляется с несложными задачами и позволяет сэкономить время и вычислительную мощность, однако данные не всегда линейно зависимы, а также ошибка в 20% может оказаться катастрофической, если, например, решается задача о диагнозе пациента.

4.2 Решающее дерево

Этот алгоритм явно лучше подходит для нелинейных зависимостей, чем линейная регрессия, главным недостатком алгоритма является неустойчивость: алгоритм крайне чувствителен к входным данным, применение PCA действительно ускорило процесс немного и позволило хранить данные меньшей размерности, однако это привело к понижению точности. Выбор: меньшая размерность или более высокая точность — не имеет правильного ответа, потому что все будет зависеть от конкретной задачи, в настоящей работе выгоднее обойтись без PCA, потому что данные не слишком большие и выигрыш во времени не ощущается, однако при обучении на изображениях в большом разрешении, думаю, применение PCA будет необходимо для обучения алгоритма за разумное время.

4.3 Полносвязная нейронная сеть

Нейронные сети это сильные алгоритмы глубокого обучения, и они справляются с многими крайне сложными задачами, которые касаются обработки изображений и текстов, причем как обработка изображений делится на массу подзадач (сегментация, детекция, классификация и тд), так и задачи языкового обучения имеют массу вариантов, однако существует класс задач, с которыми более простые алгоритмы машинного обучения справляются явно лучше, одна из таких задач была расмотрена в текущей работе.

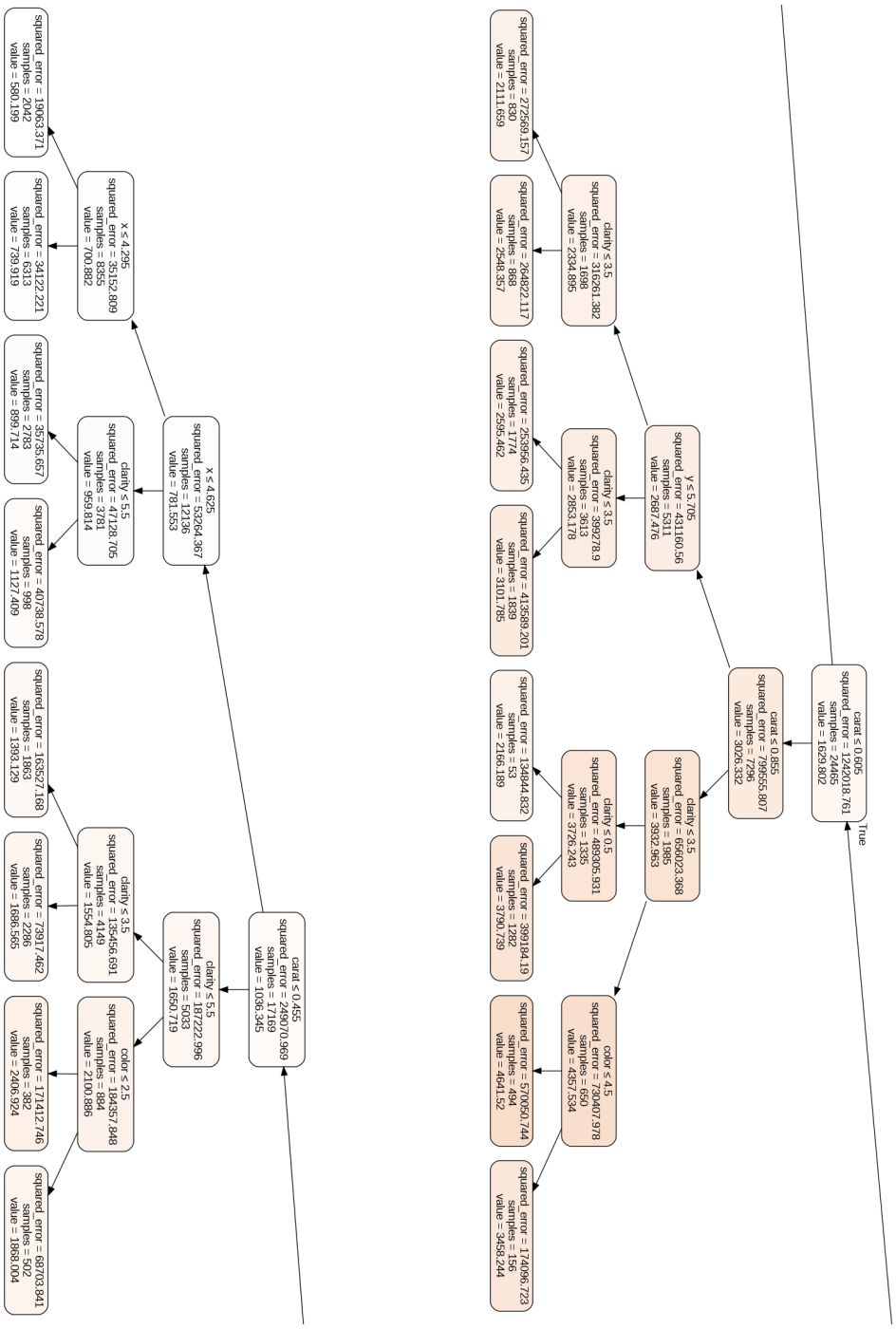
4.4 Итог

С задачей прогнозирования цены на алмазы лучше всего справился алгоритм решающего дерева, однако это не значит, что это лучший алгоритм для решения любой задачи: каждой задаче требуется свой подход и свои алгоритмы.

Список литературы

1. C. M. Bishop, *PATTERN RECORNITION and MACHINE LEARNING*, (http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf).
2. Msucourses, *Data-Analysis-with-Python*, май 2024, (https://github.com/MSUcourses/Data-Analysis-with-Python/blob/main/Machine%20Learning/lectures_spring_2023.md).
3. *1.10. Decision Trees - scikit-learn*, окт. 2021, (https://scikit-learn.ru/1-10-decision-trees).
4. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, http://www.deeplearningbook.org (MIT Press, 2016).

Приложение


carat ≤ 0.605
squared_error = 1242018.761
samples = 24465
value = 1629.802
True
carat ≤ 0.855
squared_error = 799555.807
samples = 17736
value = 3026.332
carat ≤ 0.455
squared_error = 2490170.969
samples = 6729
value = 1036.345
y ≤ 5.705
squared_error = 431160.56
samples = 5311
value = 2687.476
clarity ≤ 3.5
squared_error = 656023.368
samples = 1985
value = 3932.963
x ≤ 4.625
squared_error = 532264.367
samples = 12136
value = 781.553
clarity ≤ 5.5
squared_error = 135456.691
samples = 4149
value = 1554.805
clarity ≤ 3.5
squared_error = 399278.9
samples = 3613
value = 2853.178
clarity ≤ 0.5
squared_error = 489305.931
samples = 1335
value = 3726.243
clarity ≤ 5.5
squared_error = 187222.996
samples = 5033
value = 1650.719
color ≤ 2.5
squared_error = 184257.848
samples = 884
value = 2100.886
clarity ≤ 3.5
squared_error = 316261.382
samples = 1698
value = 2334.895
color ≤ 4.5
squared_error = 730407.978
samples = 650
value = 4357.534
x ≤ 4.295
squared_error = 351452.809
samples = 8355
value = 700.882
clarity ≤ 5.5
squared_error = 471128.705
samples = 3781
value = 959.814
squared_error = 2725693.157
samples = 1920
value = 2111.659
squared_error = 2640822.117
samples = 968
value = 2548.357
squared_error = 2589566.435
samples = 1774
value = 2595.462
squared_error = 413589.201
samples = 1839
value = 3101.785
squared_error = 1349844.832
samples = 53
value = 2166.189
squared_error = 3991184.19
samples = 1282
value = 3790.739
squared_error = 5700650.744
samples = 494
value = 4641.52
squared_error = 174096.723
samples = 156
value = 3458.244
squared_error = 19063.371
samples = 2042
value = 580.199
squared_error = 34122.221
samples = 6313
value = 739.919
squared_error = 35735.657
samples = 2783
value = 899.714
squared_error = 40738.578
samples = 998
value = 1127.409
squared_error = 163527.168
samples = 1863
value = 1393.129
squared_error = 73917.462
samples = 2286
value = 1686.565
squared_error = 171412.746
samples = 382
value = 2406.924
squared_error = 68703.841
samples = 502
value = 1868.004

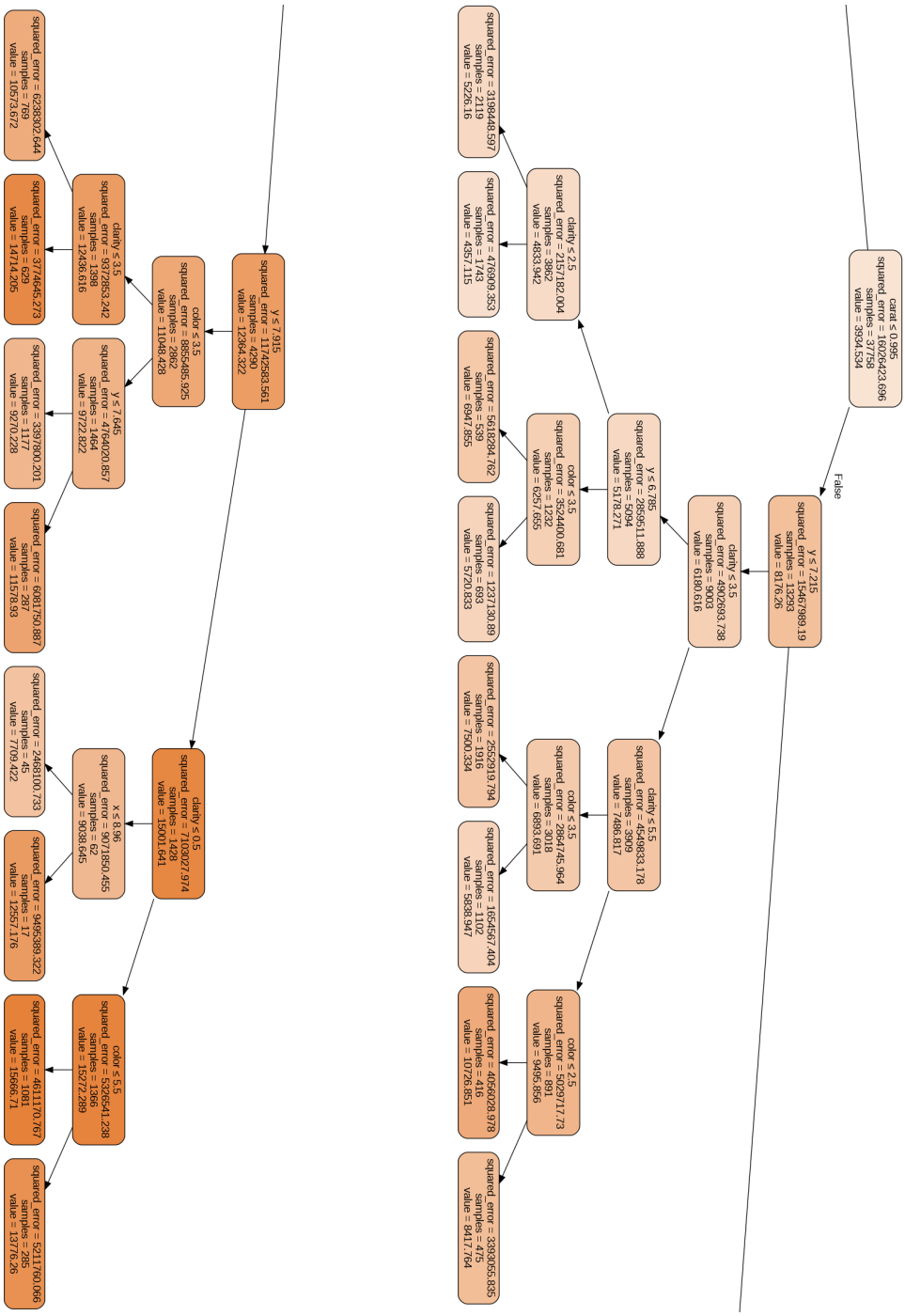


Рис. 5.1: Схема графа глубиной 5 вершин