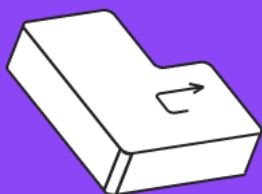




Генерация признаков. Методы отбора признаков. Подбор гиперпараметров.

Библиотеки Python для Data
Science



Оглавление

Введение	3
Термины, используемые в лекции	5
Генерация признаков	6
Отбор признаков	14
Подбор гиперпараметров	30
Что можно почитать еще?	38
Используемая литература	38

Введение

Всем привет! В этой лекции мы поговорим о важном этапе в машинном обучении — генерации признаков. Мы рассмотрим различные методы отбора признаков и подбора гиперпараметров, которые помогут нам создать эффективные модели. Эффективные модели в машинном обучении могут дать значительное улучшение в точности и производительности на практике. То есть даст нам:

1. Более точные прогнозы: эффективные модели могут давать более точные предсказания и прогнозы, что может улучшить качество.
2. Более быстрая обработка данных: эффективные модели имеют более высокую скорость обработки данных, что может быть важным, особенно при работе с большими объемами данных или в реальном времени.
3. Лучшая интерпретируемость: модели машинного обучения, такие как логистическая регрессия или деревья решений, обладают лучшей интерпретируемостью, что позволяет лучше понять, какие факторы оказывают наибольшее влияние на прогнозы.

4. Более эффективное использование ресурсов: эффективные модели могут потреблять меньше вычислительных ресурсов (например, памяти или процессорного времени), что может быть важным для экономии средств.
5. Более робастное обобщение: эффективные модели могут лучше справляться с шумом или выбросами в данных, что делает их более надежными и робастными в реальных условиях, то есть обладают способностью поддерживать свою производительность при столкновении с неопределенностями.

Отбор признаков часто используется в машинном обучении для уменьшения размерности данных и улучшения производительности модели. Он позволяет выбрать наиболее информативные и релевантные признаки, удаляя неинформативные или коррелирующие признаки. Это может привести к более простой и интерпретируемой модели, улучшению обобщающей способности, избежанию переобучения и увеличению скорости обучения.

Существует несколько методов отбора признаков, которые помогают выбрать наиболее важные признаки для построения модели. Некоторые из них включают:

- **Обратное удаление (Backward Elimination):** Начиная с модели, включающей все признаки, последовательно удаляются наименее значимые признаки на основе их влияния на модель. Этот процесс продолжается до получения оптимального набора признаков.
- **Прямой отбор (Forward Selection):** Начиная с модели без признаков, последовательно добавляются наиболее информативные признаки на основе их влияния на модель. Этот процесс продолжается до достижения оптимального набора признаков.
- **Рекурсивное исключение признаков (Recursive Feature Elimination):** Модель обучается с использованием всех признаков, и затем наименее важные признаки исключаются. Затем процесс повторяется с уменьшенным набором признаков до тех пор, пока не будет достигнуто определенное количество наиболее важных признаков.
- **Отбор на основе важности (Importance-based Selection):** Используются методы, которые оценивают важность каждого признака в модели, такие как случайный лес или градиентный бустинг. После оценки важности, можно выбрать наиболее значимые признаки для использования в модели.

- **Корреляционный отбор (Correlation-based Selection):** Оцениваются корреляции между признаками и целевой переменной, а также между самими признаками. Признаки с высокой корреляцией с целевой переменной и низкой корреляцией между собой могут быть выбраны как наиболее информативные.
- **Методы регуляризации:** Некоторые алгоритмы, такие как LASSO и Ridge регрессия, могут использоваться для отбора признаков путем добавления штрафа к коэффициентам признаков. Это приводит к уменьшению вклада незначимых признаков и способствует отбору наиболее информативных.

Это лишь несколько примеров методов отбора признаков. Выбор конкретного метода зависит от типа данных, задачи и сущностей, с которыми работает модель. Часто комбинирование нескольких методов может привести к лучшим результатам.

Подбор гиперпараметров является важным этапом в процессе разработки модели машинного обучения. Гиперпараметры — это настройки модели, которые настраиваются вручную перед обучением и не могут быть оптимизированы методом обратного распространения ошибки. Они влияют на работу модели и могут существенно влиять на ее производительность.

Вот несколько причин, по которым подбор гиперпараметров важен:

- **Оптимизация производительности:** Подбор гиперпараметров позволяет настроить модель таким образом, чтобы достичь наилучших показателей производительности. Тщательное подбор гиперпараметров может привести к улучшению точности модели, скорости обучения и снижению переобучения.
- **Адаптация к конкретным данным и задачам:** Разные наборы данных и задачи требуют разных комбинаций гиперпараметров для достижения наилучших результатов. Подбор гиперпараметров позволяет настроить модель на специфические особенности данных и требования задачи, улучшая ее способность к обобщению.
- **Управление сложностью модели:** Гиперпараметры позволяют контролировать сложность модели, влияя на ее размерность, структуру и функциональность. Например, в глубоком обучении гиперпараметры, такие, как количество скрытых слоев, количество нейронов в каждом слое и скорость обучения, определяют сложность и емкость модели.

- **Оптимизация ресурсов:** Модели с разными гиперпараметрами могут потреблять разное количество вычислительных ресурсов, таких как время обучения и требуемая память. Подбор гиперпараметров позволяет настроить модель с учетом доступных ресурсов, чтобы достичь наилучших результатов при оптимальном использовании ресурсов.

Подбор гиперпараметров является итеративным процессом, в котором проводится множество экспериментов с различными комбинациями параметров. Используя методы перекрестной проверки и алгоритмы оптимизации, можно найти оптимальные значения гиперпараметров для каждой конкретной модели и задачи.

Понимание этих процессов является ключевым для достижения высокой точности и надежности наших моделей. Давайте начнем!

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Что такое генерация признаков
- Какие есть методы отбора признаков
- Как осуществляется подбор гиперпараметров

Термины, используемые в лекции

Гиперпараметры — это настройки модели, которые настраиваются вручную перед обучением и не могут быть оптимизированы методом обратного распространения ошибки.

Признак («фича» (от англ feature)) – это переменная (столбец в таблице), которая описывает отдельную характеристику объекта.

Построение признаков (Feature Engineering) - это процесс, при котором мы создаем новые переменные для таблицы из исходных данных.

Отбор признаков (feature selection) – это оценка важности того или иного признака с помощью алгоритмов машинного обучения и отсеивание ненужных.

Генерация признаков

Генерация новых признаков, также называемая инженерией признаков, является процессом создания новых переменных на основе существующих данных.

Этот процесс имеет целью улучшить представление данных для модели и расширить информацию, которую модель может использовать для прогнозирования или классификации.

Несколько методов генерации новых признаков:

Производные от числовых переменных:

Математические операции: Новые признаки могут быть получены путем выполнения математических операций на числовых переменных, таких как сложение, вычитание, умножение и деление.

Логарифмы и экспоненты: Взятие логарифма или экспоненты от числовых переменных может помочь нормализовать распределение или выделить определенные шаблоны данных.

Полиномиальные признаки: Создание новых признаков путем возведения в степень или перемножения числовых переменных может помочь захватить нелинейные взаимодействия.

Бинарные и категориальные переменные:

Преобразование категориальных переменных: Категориальные переменные могут быть преобразованы в числовые с помощью методов, таких как кодирование One-Hot или Label Encoding.

Создание комбинированных или агрегированных признаков: Новые признаки могут быть созданы на основе комбинаций или агрегации категориальных переменных, например, среднее или сумма по группам.

Информация о времени и дате:

Извлечение компонентов времени: Извлечение компонентов времени, таких как год, месяц, день недели, может помочь раскрыть сезонные или циклические паттерны данных.

Расчет периодичности: Создание новых признаков, которые отражают периодичность данных, таких как среднее или стандартное отклонение за предыдущие периоды, может быть полезно для моделей, работающих с временными рядами.

Извлечение признаков из текстовых данных:

Мешок слов: Представление текстовых данных в виде мешка слов (Bag of Words) позволяет создать новые бинарные или числовые признаки, отражающие наличие или частоту определенных слов или фраз.

TF-IDF: Создание новых признаков на основе меры TF-IDF (term frequency-inverse document frequency) для слов или фраз может помочь в идентификации ключевых понятий или выделении важности текстовых данных.

Этапы генерации новых признаков могут включать следующие шаги:

1. Понимание данных: В первую очередь необходимо иметь хорошее понимание данных, с которыми вы работаете. Исследуйте структуру, типы переменных, распределения, особенности и зависимости между признаками. Это поможет определить потенциальные области для генерации новых признаков.
2. Идеи генерации признаков: Основываясь на знаниях о данных, сформулируйте идеи о том, какие новые признаки могут быть полезны для модели. Размышляйте о преобразованиях, комбинациях, агрегациях признаков и других способах расширения информации, содержащейся в данных.
3. Создание новых признаков: Реализуйте свои идеи, создавая новые признаки на основе существующих данных. Это может быть математические операции, преобразования категориальных переменных, извлечение признаков из текста и другие методы, упомянутые ранее. Эти операции могут выполняться с использованием программного кода или визуальных инструментов для обработки данных.
4. Оценка важности новых признаков: После создания новых признаков важно оценить их влияние на модель. Используйте методы отбора признаков, статистические тесты или модели машинного обучения для определения значимости новых признаков в отношении целевой переменной.
5. Итеративный процесс: Генерация новых признаков часто является итеративным процессом, требующим экспериментирования и оценки различных вариантов. Повторяйте шаги 3 и 4, модифицируя и добавляя новые признаки, а затем оценивайте их влияние на результат модели.

6. Оптимизация и отбор признаков: Проанализируйте важности и влияние новых признаков и отфильтруйте их, оставляя только те, которые действительно улучшают производительность модели. Используйте методы отбора признаков, перекрестную проверку и другие подходы для оптимизации набора признаков.

Генерация новых признаков является творческим процессом, и конкретные методы зависят от типа данных, задачи и собственного интуитивного понимания данных. Важно проводить эксперименты, оценивать влияние новых признаков на модель и не забывать о значимости признаков для конкретной задачи. Не стесняйтесь пробовать различные идеи и методы, а также привлекать знание области, в которой работаете, для более эффективной генерации признаков.

Рассмотрим основные методы генерации признаков:

Начнем с математической операций. Вспомним, что признаки могут быть различных типов:

1. Бинарные, которые могут принимать только два значения. Например, [true, false], [0, 1], [«да», «нет»].
2. Категориальные или номинальные. Они имеют ограниченное количество уровней. Например, признак «день недели» имеет 7 уровней: понедельник, вторник и т.д. до воскресенья.
3. Упорядоченные. В некотором смысле, они похожи на категориальные признаки, но отличаются тем, что существует четкая иерархия категорий. Например, «классы в школе» от 1 до 11. Этот тип также включает в себя «время суток» с 24 уровнями и явно определенным порядком.
4. Числовые или количественные. Это значения, которые могут находиться в диапазоне от минус бесконечности до плюс бесконечности и не относятся к предыдущим типам признаков.

Важно отметить, что для задач машинного обучения необходимы только те признаки, которые реально влияют на конечный результат.

Построение признаков (Feature Engineering) - это процесс, при котором мы создаем новые переменные для таблицы из исходных данных. В реальной жизни данные редко представлены в виде готовых матриц, поэтому для решения любой задачи необходимо извлечение признаков. Например, в базе данных интернет-магазина есть таблица "Покупатели", где каждая строка соответствует посещению сайта клиентом. Также может быть таблица "Взаимодействия", где каждая строка представляет собой взаимодействие (клик или посещение страницы), совершенное клиентом на сайте. В этой таблице также содержится информация о времени взаимодействия и типе события (покупка, поиск или добавление в корзину). Две таблицы связаны между собой по столбцу "Customer ID".

Взаимодействия

Interaction ID	Customer ID	Дата покупки	Тип	Количество
0	0	16/07/2021 09:21:01	Добавить в корзину	N/A
1	0	16/07/2021 09:21:56	Покупка	60
2	0	17/07/2021 17:54:32	Покупка	400
3	1	16/08/2021 10:32:09	Добавить в корзину	N/A
4	1	16/08/2021 10:33:03	Покупка	30000

Для повышения прогностической способности нам нужно использовать данные в таблице взаимодействий. Это становится возможным благодаря отбору признаков. Мы можем вычислить статистику для каждого клиента, используя все значения в таблице "Взаимодействия" с идентификатором этого клиента. Вот несколько потенциально полезных признаков, или "фич", которые помогут нам решить задачу:

- Среднее время между предыдущими покупками.
- Средняя сумма предыдущих покупок.
- Максимальная сумма предыдущих покупок.
- Время, прошедшее с момента последней покупки.
- Общее количество покупок в прошлом.

Для создания этих признаков нам нужно найти все взаимодействия, связанные с конкретным клиентом. Затем мы отфильтруем тех, чей тип не является "Покупкой", и вычислим функцию, которая вернет одно значение, используя имеющиеся данные.



Следует обратить внимание, что данный процесс уникален для каждого случая использования и набора данных.

Основные и самые известные методы feature engineering

Построение признаков на табличных данных

- Избавление от пропущенных значений

Пропущенные значения являются одной из самых распространенных проблем, с которыми можно столкнуться при подготовке данных. Это существенно влияет на производительность моделей машинного обучения.

Простейшим решением для пропущенных значений является удаление строк или целого столбца. Оптимального порога для удаления не существует, но можно использовать значение 50% и удалить строки со столбцами, в которых количество пропущенных значений превышает этот порог.

- Заполнение пропущенных значений может быть выполнено различными способами. Один из предпочтительных вариантов — не удалять пропущенные значения, а заменить их на определенное значение. Этот подход позволяет сохранить размер данных и не потерять информацию. Важно определить, что именно считать недостающими значениями. Например, если в столбце присутствуют числа 1 и N/A, то можно предположить, что строки с N/A соответствуют значению 0.

Другой пример — если у вас есть столбец, отражающий количество посещений клиентов за последний месяц, то отсутствующие значения могут быть заменены на 0.

За исключением описанного выше, хорошим способом заполнения пропущенных значений является использование медиан столбцов. Поскольку среднее значение может быть чувствительным к выбросам, медиана является более устойчивой мерой центральной тенденции.

Одно из наиболее часто используемых математических преобразований при построении признаков — логарифмическое преобразование. Оно позволяет обрабатывать искаженные данные и придает распределению более приближенный к нормальному виду. В большинстве случаев величина данных изменяется в пределах диапазона данных. Например, разница между возрастом 15-20 лет не

равна разнице между возрастом 65-70 лет, так как в молодом возрасте разница в 5 лет имеет большее значение по всем остальным аспектам. Такой тип данных возникает в результате мультипликативного процесса, и логарифмическое преобразование нормализует подобные различия величин. Кроме того, оно снижает влияние выбросов путем нормализации разницы величин, делая модель более надежной.



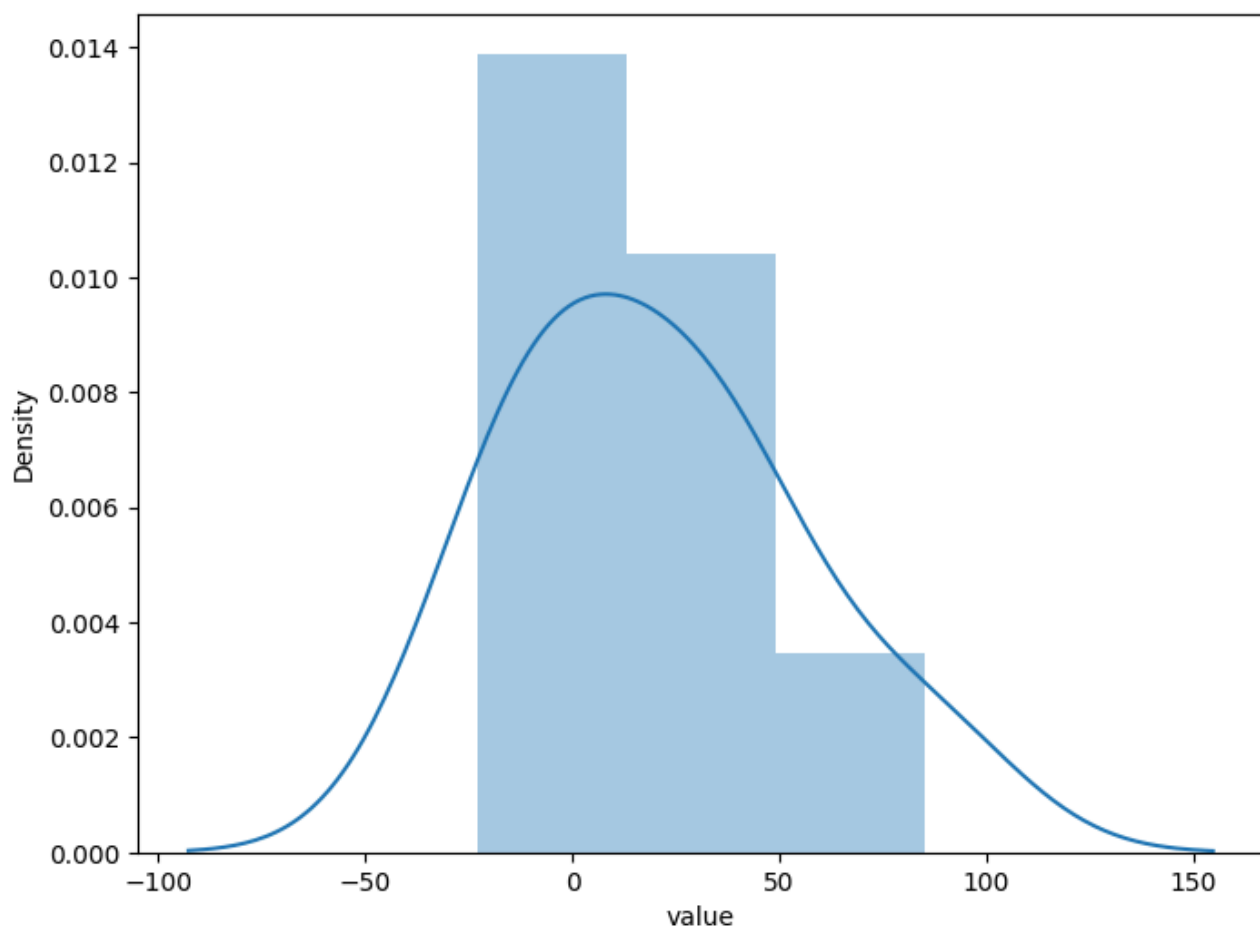
Важное примечание: данные, которые вы применяете, должны иметь только положительные значения, иначе вы получите ошибку.

```
import pandas as pd
import numpy as np

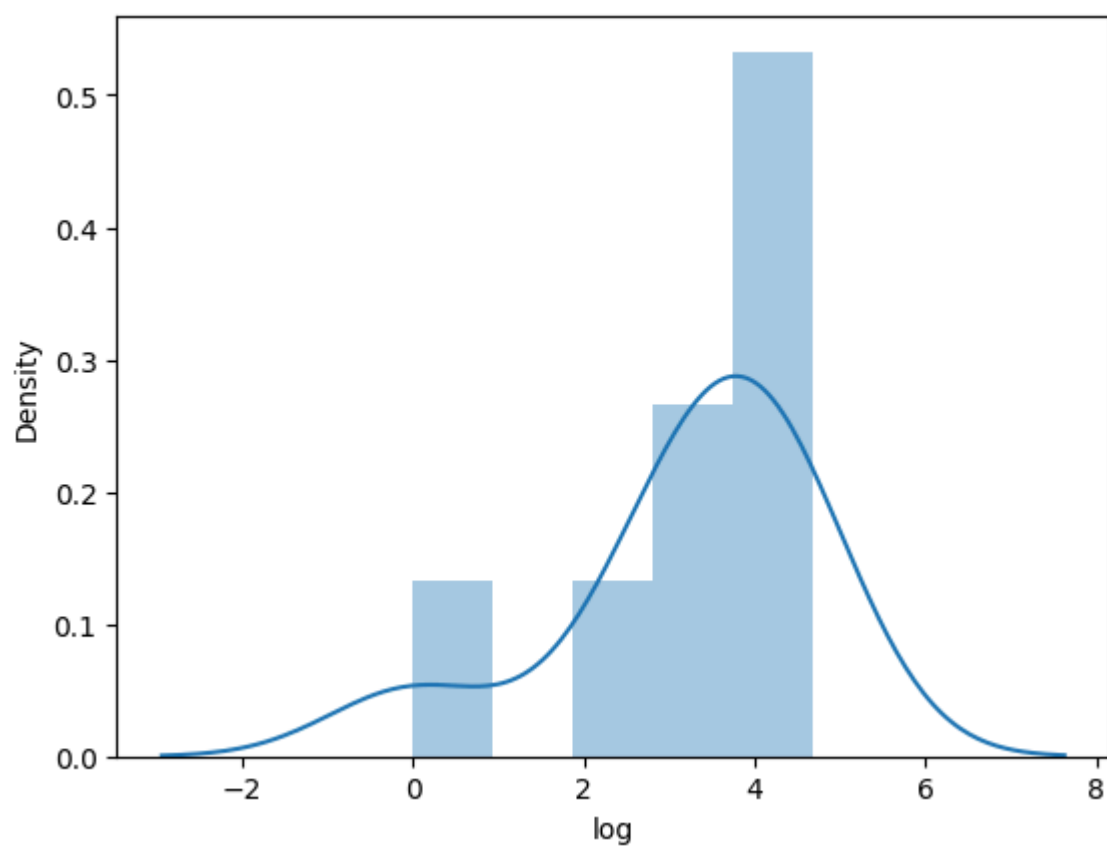
# Пример логарифмической трансформации
data = pd.DataFrame({'value': [2, 45, -23, 85, 28, 2, 35, -12]})
data['log+1'] = (data['value'] + 1).transform(np.log)

# Обработка отрицательных значений
# (Обратите внимание, что значения разные)
data['log'] = (data['value'] - data['value'].min() + 1).transform(np.log)
```

Распределение без логарифмирования




Распределение с логарифмированием



Другой способ - быстрое кодирование (One-Hot encoding). В данном методе значения в столбце распределяются по нескольким столбцам-флагам и им присваиваются значения 0 или 1. Бинарные значения выражают связь между группированным и закодированным столбцами. Этот метод позволяет преобразовать категориальные данные, которые сложно интерпретировать алгоритмам, в числовой формат. При этом группировка происходит без потери какой-либо информации. Например, можно преобразовать категорию "цвет" в несколько столбцов-флагов, где каждый столбец будет соответствовать определенному цвету, и присвоить им значения 0 или 1 в зависимости от наличия этого цвета в исходных данных.

User	City		
1	Roma		
2	Madrid		
1	Madrid		
3	Istanbul		
2	Istanbul		
1	Istanbul		
1	Roma		



User	Istanbul	Madrid
1	0	0
2	0	1
1	0	1
3	1	0
2	1	0
1	1	0
1	0	0

Приведенная ниже функция отражает использование метода быстрого кодирования с вашими данными.

```
encoded_columns = pd.get_dummies(data['column'])  
data = data.join(encoded_columns).drop('column', axis=1)
```

Процесс кодирования меток (label encoding) заключается в преобразовании строковых признаков в числовые значения, присваивая каждой категории свой уникальный номер.

Этот подход хорошо работает с методами, основанными на деревьях решений, так как они способны самостоятельно разделить этот непрерывный признак на сегменты. Однако он обычно не подходит для линейных моделей.

Кроме того, кодирование меток подходит для порядковых переменных, так как сохраняются отношения порядка между значениями.

```
train['MARRIAGE'].value_counts()
```

```
не женат/не замужем    4261
женат/замужен          3649
прочее                  90
Name: MARRIAGE, dtype: int64
```

```
1 < 2 < 3
```

Изменение масштаба признаков

В большинстве случаев числовые характеристики набора данных имеют различные диапазоны и отличаются друг от друга.

Например, столбцы возраста и месячной зарплаты будут иметь совершенно разные диапазоны.

Как сравнить эти два столбца, если это необходимо в нашей задаче? Изменение масштаба решает эту проблему, так как после этой операции элементы становятся идентичными по диапазону.

Существует два распространенных способа изменения масштаба:

Нормализация.

В данном случае все значения будут находиться в диапазоне от 0 до 1. Дискретные бинарные значения определяются как 0 и 1.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Стандартизация — это процесс масштабирования значений с учетом стандартного отклонения. Если функции имеют различное стандартное отклонение, то их диапазоны также будут различаться. Такой подход позволяет уменьшить влияние выбросов на элементы. В формуле стандартизации среднее значение обозначается как μ , а стандартное отклонение — как σ .

$$z = \frac{x - \mu}{\sigma}$$

Работа с текстом

Существует множество методов работы с текстом, но все они не могут быть охарактеризованы в одном тезисе статьи. Однако ниже будет описан наиболее популярный подход.

Перед тем, как приступить к работе с текстом, необходимо разбить его на токены — отдельные слова. Однако, если мы просто разделим текст на слова, мы можем упустить часть его смысла. Например, "Великие Луки" не является двумя токенами, а одним.

После того как документ превратится в последовательность слов, можно начинать преобразовывать их в векторы. Самый простой метод — мешок слов (Bag of Words). Мы создаем вектор, длина которого соответствует размеру словаря, и для каждого слова подсчитываем количество его вхождений в текст, затем это число подставляем на соответствующую позицию в векторе.

Отбор признаков

"Прогонка" всех признаков в модели, чтобы посмотреть, какие из них работают — не лучшая идея. На самом деле, когда в алгоритмы попадает слишком много "фич", они работают плохо. Как решить эту проблему? При помощи отбора признаков.

Отбор признаков (фич) - это оценка важности каждого признака с использованием алгоритмов машинного обучения и удаление ненужных. Существует множество алгоритмов, которые преобразуют набор слишком большого количества признаков в управляемое подмножество. Как и в случае с созданием признаков, для разных типов данных оптимальны разные подходы. Кроме того, при выборе алгоритма необходимо учитывать наши цели и то, что мы хотим сделать с обрабатываемым набором данных.

Выбор наиболее оптимальных признаков из множества существующих - сложная задача. Большое количество признаков значительно увеличивает время вычислений, а также существует риск переобучения.

Существует несколько общих методов для решения данной проблемы, которые можно отнести к одной из следующих категорий.

1. Методы фильтрации (filter methods)

Эти методы выбирают внутренние свойства признаков. Они являются более быстрыми и менее затратными с точки зрения вычислений, чем методы-оболочки.

Если у вас есть данные с большой размерностью, то использование методов фильтрации будет более экономичным с вычислительной точки зрения.

Методы фильтрации для отбора признаков основываются на статистических или эвристических критериях и не включают в себя обучение модели машинного обучения. Они оценивают важность каждого признака отдельно и выбирают наиболее информативные. Некоторые из распространенных методов фильтрации для отбора признаков включают:

1. Дисперсионный отбор (Variance Threshold): Удаляет признаки с низкой дисперсией. Низкая дисперсия означает, что признак имеет мало изменений в данных и поэтому не содержит много информации.
2. Корреляционный отбор: Оценивает корреляцию между каждым признаком и целевой переменной. Признаки с низкой корреляцией могут быть отброшены.

Напомню, что корреляция – это мера линейной связи двух или более переменных. С помощью этого метода мы можем предсказывать одну переменную на основе другой. Логика использования этого метода для выбора характеристик заключается в том, что "хорошие" переменные имеют сильную корреляцию с нашей целью. Важно отметить, что переменные должны быть коррелированы с целевым показателем, но не должны иметь корреляцию между собой.

3. Отбор на основе статистики: Использует статистические метрики, такие как chi-square, ANOVA F-value или mutual information, чтобы оценить взаимосвязь между признаком и целевой переменной. Выбираются признаки с самыми высокими значениями метрик.

Например, критерий Фишера, также известный как F-тест, является одним из наиболее распространенных методов выбора признаков. Наш алгоритм оценивает критерий для каждой переменной и возвращает их ранги в порядке убывания, после чего происходит отбор.


```

import pandas as pd
import numpy as np
from skfeature.function.similarity_based import fisher_score
import matplotlib.pyplot as plt

# Вычисляем критерий
# Где X, y - входные и выходные данные соответственно.
ranks = fisher_score.fisher_score(X, y)

# Делаем график наших "фич"
# Где data - ваш датасет
feature_importances = pd.Series(ranks, data.columns[0:len(data.columns)-1])
feature_importances.plot(kind='barh', color='teal')
plt.show()

```

Критерий хи-квадрат (тест Хи-квадрат)

Применяется для анализа категориальных признаков в наборе данных. Мы вычисляем значение хи-квадрат между каждым признаком и целевой переменной, а затем выбираем определенное количество "фич", у которых наиболее высокие показатели. Чтобы правильно использовать этот критерий для проверки связи между различными функциями в наборе данных и целевой переменной, следует учитывать следующие условия: выбранные категориальные переменные должны быть независимыми, а также частота значений должна быть больше 5.

Основные шаги его работы:

Рассчитывается взаимосвязь (корреляция) каждого признака с целевой переменной. Для этого используются различные статистические критерии в зависимости от типа признаков и задачи (хи-квадрат, коэффициент корреляции Пирсона, взаимная информация и др.).

Все признаки сортируются по убыванию их взаимосвязи с целевым признаком.

Отбираются k признаков с наибольшей взаимосвязью. Параметр k задается заранее и определяет количество отбираемых признаков.

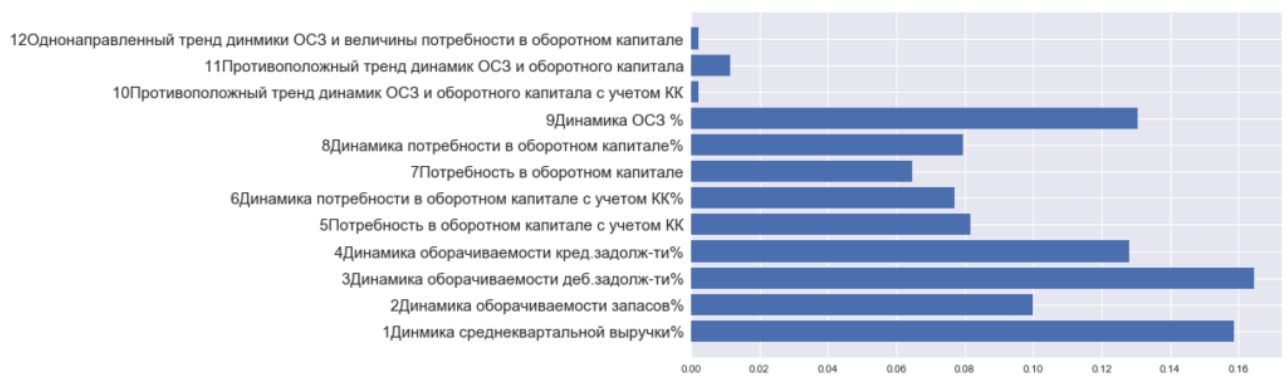
Отобранные k признаков с наибольшей значимостью и используются далее для обучения модели.

Таким образом SelectKBest позволяет выбрать наиболее важные признаки для задачи моделирования, что улучшает качество модели и снижает переобучение за счет удаления неинформативных шумовых признаков.

```
# Преобразование в категориальные данные путем преобразования в целые числа.  
# Где X, y - входные и выходные данные соответственно.  
X_categorical = X.astype(int)  
  
# Выбираем 3 признака с наивысшим "хи-квадрат".  
chi2_features = SelectKBest(chi2, k = 3)  
X_kbest_features = chi2_features.fit_transform(X_categorical, y)  
  
# Вывод "до и после"  
print("Количество признаков до преобразования:", X_categorical.shape[1])  
print("Количество признаков после преобразования:", X_kbest_features.shape[1])
```

4. Отбор с использованием важности признаков: Методы, основанные на моделях машинного обучения, могут предоставить важность каждого признака в контексте использованной модели. Например, случайные леса (Random Forest) и классификатор на основе градиентного бустинга (Gradient Boosting Classifier) предоставляют информацию о важности признаков.

```
#В список labels запишем названия признаков  
#Так как названия признаков довольно длинные, построим горизонтальную гистограмму  
position = np.arange(12)  
fig, ax = plt.subplots()  
ax.barh(position, model.feature_importances_)  
#Устанавливаем позиции тиков:  
ax.set_yticks(position)  
#Устанавливаем подписи тиков  
ax.set_yticklabels(labels,  
                    fontsize = 15)  
fig.set_figwidth(10)  
fig.set_figheight(6)  
plt.show()
```



5. Устойчивость отбора признаков (Stability Selection): Применяет метод под-выборки и случайного шума для оценки стабильности важности признаков. Случайный шум (это случайная неструктурированная информация, добавляемая к данным для внесения некоторой вариативности или случайности) в данном контексте относится к добавлению случайных значений к признакам в датасете. Такое добавление шума позволяет оценить, насколько стабильная и важная является информация, содержащаяся в каждом из признаков.

Рассмотрим пример. Предположим, у нас есть датасет, содержащий информацию о ценах на недвижимость, и мы хотим определить, какие признаки (например, площадь, количество комнат, удаленность от центра и т.д.) влияют на ценообразование. Мы можем добавить случайный шум к каждому признаку в датасете, что делает данные менее структурированными и учитывает случайные факторы, несвязанные с ценой. Затем мы можем построить модель машинного обучения, например, случайный лес или градиентный бустинг, для предсказания цены на недвижимость на основе этих признаков.

Основная идея метода заключается в создании нескольких подвыборок (subsamples) и добавлении случайного шума в данные. Затем модель машинного обучения обучается на каждой подвыборке и оценивается важность каждого признака.

Процесс стабильности отбора признаков включает несколько этапов:

Создание подвыборок: Создаются несколько подвыборок (subsamples) из исходных данных путем случайной выборки с заменой. Каждая подвыборка может содержать случайное подмножество наблюдений или признаков.

Добавление случайного шума: Случайный шум добавляется в данные путем замены или перестановки значений признаков. Это помогает в оценке устойчивости и значимости признаков.

Обучение модели: На каждой подвыборке с добавленным шумом модель машинного обучения обучается для оценки важности признаков. Это может быть любая модель, такая как случайный лес (Random Forest), метод опорных векторов (Support Vector Machines) и другие.

Оценка значимости признаков: После обучения модели на каждой подвыборке, оценивается значимость каждого признака на основе результатов из разных подвыборок. Чем чаще признак выбирается как важный, тем более стабильное значение оценивается.

Важно отметить, что метод устойчивости отбора признаков позволяет учесть случайность и вариативность в данных. Он может помочь в отборе наиболее значимых и стабильных признаков, которые могут быть более надежными и устойчивыми при использовании модели на новых данных.

6. Отбор на основе ближайших соседей (Nearest Neighbors-based Feature Selection): Основывается на оценке схожести признаков с их ближайшими соседями в пространстве признаков.

Основная идея этого метода заключается в том, что если признаки схожи между собой, то они могут нести схожую информацию о целевой переменной и быть менее информативными для моделирования.

Процесс отбора на основе ближайших соседей включает несколько шагов:

Расчет матрицы ближайших соседей: Используется алгоритм ближайших соседей, например, алгоритм k-ближайших соседей (k-Nearest Neighbors), чтобы рассчитать матрицу ближайших соседей для всех признаков.

Оценка схожести признаков: На основе матрицы ближайших соседей вычисляется метрика схожести между каждым признаком и его ближайшими соседями. Примеры метрик схожести включают коэффициент корреляции, расстояние между признаками, информационные метрики и т.д.

Ранжирование признаков: После оценки схожести признаков они могут быть ранжированы по убыванию их значимости. Признаки, более схожие с другими признаками, могут быть отмечены как менее информативные и исключены из модели.

Отбор на основе ближайших соседей может помочь в устранении избыточности и коррелированности между признаками, что позволяет улучшить производительность и интерпретируемость модели. Однако, важно иметь в виду, что эффективность этого метода может сильно зависеть от датасета и применяемых метрик схожести.

Эти методы фильтрации помогают снизить размерность признакового пространства путем отбора наиболее информативных признаков. Они могут быть быстрее, но не всегда учитывают сложные взаимодействия между признаками. Рекомендуется проводить эксперименты с различными методами фильтрации и тестировать их эффективность на конкретной задаче.

2. **Методы обертки (wrapper methods)** отличаются тем, что они ищут все возможные подмножества признаков и оценивают их качество, пропуская их через модель. Выбор функции основан на конкретном алгоритме машинного обучения, который мы используем. Эти методы следуют принципу жадного поиска, оценивая все возможные комбинации функций по определенному критерию. Методы обертки обычно обеспечивают более точные прогнозы, чем методы фильтрации.

Прямой отбор признаков

В работе используется "нулевая модель" без переменных. На первом этапе последовательно добавляются переменные в "пустую модель" и выбирается та, которая дает лучший результат. Затем в модель, содержащую только одну переменную, последовательно добавляются остальные переменные и выбирается та, которая приводит к наибольшему улучшению качества модели.

Начальная модель без переменных



Добавление наиболее значимой переменной



Добавление переменных продолжается пока не достигнуто правило остановки, или переменные не закончатся



Принцип прямого отбора признаков можно описать следующим образом:

1. Начать с пустого набора признаков: В начале процесса отбора признаков не используется ни один признак, то есть набор признаков пустой.

2. Оценка каждого признака по отдельности: Каждый признак из исходного набора оценивается по отдельности, используя какую-либо метрику или алгоритм для измерения его информативности или релевантности по отношению к целевой переменной.

Ниже приведены некоторые распространенные оценки, которые могут быть использованы при прямом отборе признаков:

Корреляция: Оценка корреляционной связи между каждым признаком и целевой переменной. Более высокая корреляция указывает на более сильное влияние признака на целевую переменную.

p-значение: Вычисление статистической значимости каждого признака с использованием теста гипотезы, например, t-тест или анализ дисперсии (ANOVA). Более низкое p-значение указывает на более значимый признак.

Взаимная информация: Оценка количества информации, разделяемой между признаком и целевой переменной. Более высокая взаимная информация указывает на более сильную связь между признаком и целевой переменной.

Коэффициент Джини: Измерение нормированной неопределенности (или неравенства) при разделении данных на основе значения признака. Более высокий коэффициент Джини указывает на более информативный признак.

Важности признаков в модели: Оценка важности признаков, основанная на построении модели машинного обучения, такой как случайный лес (Random Forest), градиентный бустинг (Gradient Boosting) и др. Модель может предоставить оценку значимости каждого признака на основе их вклада в построение модели.

3. Добавление наиболее информативного признака: Измерения информативности каждого признака сравниваются, и наиболее информативный признак добавляется в текущий набор признаков.
4. Оценка модели с добавленным признаком: Модель обучается и оценивается с использованием текущего набора признаков, включая добавленный признак.
5. Повторение шагов 2-4: Процесс повторяется, оценивая и добавляя следующий наиболее информативный признак до достижения желаемого количества признаков или достижения требуемой производительности модели.

Прямой отбор признаков следует простому принципу пошаговой добавки признаков в модель, начиная с наиболее информативных признаков. Этот метод позволяет сохранить наиболее информативные признаки в модели и упростить сложность признакового пространства.

```

import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

# Загрузим данные
df = pd.read_csv('data.csv')
X = df.drop('SalePrice', axis=1)
y = df['SalePrice']

# Применим прямой отбор признаков
selector = SelectKBest(score_func=f_regression, k=5) # Используем корреляцию и выбираем 5 признаков
X_new = selector.fit_transform(X, y)

# Получим индексы выбранных признаков
selected_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_indices]

# Выведем выбранные признаки
print(selected_features)

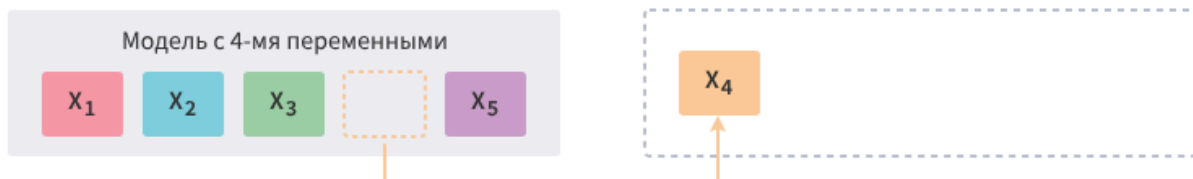
```

Метод последовательного отбора. Алгоритм отбора начинает работу с "полной" модели, содержащей все переменные. Затем пошагово удаляет наименее значимые переменные до достижения предварительно заданного правила остановки или пока в модели не останется ни одной переменной. Как и в случае прямого отбора, необходимо определить наименее значимую переменную на каждом шаге и правило остановки.

Начальная модель включает все переменные



Исключить наименее значимую переменную



Продолжить исключение пока не будет выполнено правило остановки



Метод последовательного отбора признаков (Sequential Feature Selection) — это процесс последовательного выбора подмножества признаков из исходного множества признаков для построения модели или алгоритма машинного обучения. Цель состоит в том, чтобы найти оптимальный набор признаков, который дает наилучшую производительность модели.

Процесс последовательного отбора признаков обычно включает следующие шаги:

Определение критерия или метрики, по которой будет оцениваться производительность модели (например, точность классификации или среднеквадратическая ошибка регрессии).

Инициализация полного набора признаков, в зависимости от выбранного метода.

Применение алгоритма для оценки значимости каждого отдельного признака или комбинации признаков, используя выбранный критерий оценки производительности модели. Такие алгоритмы могут включать методы обертывания (wrapper methods), встроенные методы (embedded methods), методы фильтрации (filter methods) и комбинированные методы.

Выбор наиболее значимого признака или комбинации признаков, основываясь на оценке значимости.

Повторение шагов 3 и 4, пока не будет достигнуто определенное условие остановки, такое как достижение заданного числа признаков или наилучшей производительности модели.

Валидация и тестирование модели на выборке, не участвовавшей в процессе отбора признаков, для оценки ее обобщающей способности.

Метод последовательного отбора признаков является часто используемым подходом для извлечения наиболее информативных признаков в данных или для устранения шума и избыточности в наборе признаков. В результате этого процесса модель может стать более интерпретируемой, эффективной и обладать лучшей способностью обобщения на новых наборах данных.

Визуальный пример может быть полезен для лучшего понимания. Представим, что у нас есть набор данных о домах, и мы хотим предсказать их стоимость. У нас есть следующие признаки: площадь дома, количество спален, количество ванных комнат, возраст дома и наличие гаража.

Метод исчерпывающего выбора признаков начинается с создания моделей для всех возможных комбинаций признаков. Для этого модель обучается и оценивается на каждой комбинации. Например, мы строим модель с признаками "площадь дома" и "количество спален", затем модель с признаками "площадь дома" и "количество ванных комнат" и так далее.

После того как все модели были построены и оценены, выбирается комбинация признаков, предоставляющая наилучшие результаты. В данном примере это может быть комбинация признаков "площадь дома" и "количество спален". Далее, данная комбинация признаков может быть использована для обучения модели на всем наборе данных.

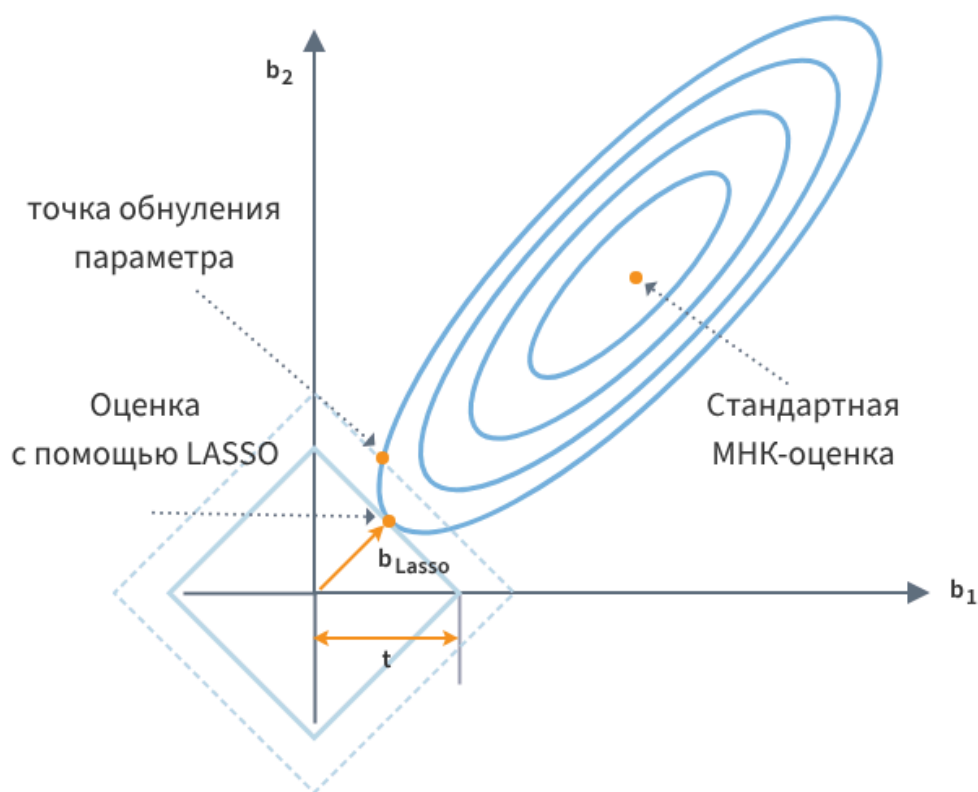
Метод исчерпывающего выбора признаков является простым в реализации, но может быть вычислительно затратным, особенно при большом количестве признаков. В таких случаях могут быть применены более эффективные методы выбора признаков, такие как методы основанные на оценке значимости признаков или алгоритмы отбора признаков на основе рекурсивного итерационного исключения.

3. Методы встроенные (embedded methods) объединяют преимущества первых двух подходов и снижают вычислительные затраты. Важной особенностью этих методов является извлечение "фич" на каждой итерации.

L1-регуляризация (регуляризация LASSO)

Регуляризация включает добавление штрафа (penalty) к различным параметрам модели, чтобы избежать чрезмерной подгонки. При использовании

L1-регуляризации в линейной модели штраф применяется к коэффициентам, умножающим каждый предиктор. Lasso-регуляризация имеет свойство, которое позволяет уменьшить некоторые коэффициенты до нуля. Следовательно, такие "фичи" можно просто удалить из модели.



Метод отбора признаков с использованием L1-регуляризации, также известный как LASSO (Least Absolute Shrinkage and Selection Operator), является эффективным инструментом для выбора наиболее значимых признаков в модели машинного обучения. Он применяется в линейной регрессии и других моделях, основанных на линейной комбинации признаков.

Метод LASSO (оператор наименьшего абсолютного сокращения и выбора) является методом оценки параметров модели линейной регрессии с использованием регуляризации. В отличие от других методов, LASSO может устанавливать нулевые значения для некоторых параметров модели.

Основная идея метода состоит в том, чтобы ограничить сумму абсолютных значений параметров модели. Вместо минимизации суммы квадратов остатков, как в случае обычной линейной регрессии, мы минимизируем сумму модулей параметров.

Параметры модели выбираются таким образом, чтобы минимизировать критерий оценки (сумму модулей параметров) при условии, что эта сумма не превышает заданного значения.

При большом значении параметра регуляризации, решение метода LASSO будет совпадать с решением обычной линейной регрессии. Однако, при уменьшении значения параметра регуляризации, некоторые коэффициенты регрессии будут обнуляться, что позволяет нам как повысить точность модели, так и выбрать наиболее значимые переменные.

Интуитивно, можно представить, что при ограничении суммы модулей параметров модели, мы ограничиваем размер "эллипсоида" суммы квадратов остатков. Параметр регуляризации задает форму этого эллипсоида, и при его уменьшении некоторые коэффициенты обнуляются в точках касания эллипсоида и "октаэдра" (многогранника в форме восьмигранника).

Простыми словами, конечная цель метода L1-регуляризации (или LASSO) состоит в том, чтобы выбрать самые важные признаки для нашей модели и убрать незначимые.

Мы строим модель, которая предсказывает результат на основе набора признаков. L1-регуляризация помогает нам выбирать только те признаки, которые имеют наибольший вклад в предсказание и исключать остальные, которые не вносят существенного влияния на результат.

Идея заключается в том, чтобы добавить дополнительную часть к нашей функции потерь, которая штрафует модель за использование незначимых признаков. Этот штраф основан на абсолютном значении параметров модели. Коэффициент регуляризации контролирует то, насколько сильно мы штрафуем модель за использование незначимых признаков.

При оптимизации функции потерь с помощью L1-регуляризации, некоторые параметры модели автоматически становятся равными нулю. Это означает, что соответствующие признаки исключаются из модели, так как они не оказывают существенного влияния на предсказание.

Это особенно полезно, когда у нас есть большое количество признаков и мы хотим упростить модель, выбрав только самое важное.

Рекурсивное устранение объектов (RFE) - это алгоритм выбора подмножества наиболее релевантных объектов из набора данных. В начале процесса RFE применяется ко всем объектам в наборе данных, а затем итеративно удаляются наименее значимые объекты до достижения желаемого количества объектов.

Основная идея RFE заключается в том, что наиболее релевантные функции оказывают наибольшее влияние на целевую переменную и, следовательно, являются более полезными для прогнозирования цели. Для оценки важности каждого признака RFE использует модель, такую как линейная регрессия или метод

опорных векторов. На каждой итерации удаляются признаки с наименьшей важностью.

На каждой итерации алгоритм удаляет наименее важные функции, а затем дополняет модель оставшимися функциями. Этот процесс повторяется до тех пор, пока не будет достигнуто желаемое количество функций или производительность модели больше не перестанет улучшаться.

В `scikit-learn` RFE с перекрестной проверкой может быть выполнен с использованием класса `RFECV`. Этот класс представляет собой метаоценщик, который оборачивает оценщик и выполняет RFE с перекрестной проверкой, чтобы найти оптимальное количество объектов.

Вот пример того, как использовать класс `RFECV` в `scikit-learn` научиться выполнять RFE с перекрестной проверкой для модели дерева решений:

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import RFECV
from sklearn.tree import DecisionTreeClassifier

# Load the iris dataset
X, y = load_iris(return_X_y=True)

# Create a decision tree classifier
estimator = DecisionTreeClassifier()

# Use RFE with cross-validation to
# find the optimal number of features
selector = RFECV(estimator, cv=5)
selector = selector.fit(X, y)

# Print the optimal number of features
print("Optimal number of features: %d" % selector.n_features_)

# Print the selected features
print("Selected features: %s" % selector.support_)
```

```
Optimal number of features: 3
Selected features: [False True True True]
```

В этом примере мы сначала загружаем набор данных `iris` с помощью функции `load_iris` из `scikit-learn`. Затем мы создаем классификатор дерева решений, используя класс `DecisionTreeClassifier`.

Далее мы создаем экземпляр класса `RFECV` и передаем его в классификатор дерева решений в качестве оценщика. Мы также указываем количество сгибов для перекрестной проверки, используя параметр `cv`. Затем мы вызываем метод `fit` в экземпляре `'RFECV'`, чтобы выполнить RFE с перекрестной проверкой в наборе данных `iris`. Это позволит найти оптимальное количество функций и выбрать

наиболее релевантные функции. Наконец, мы печатаем оптимальное количество функций и выбранные функции.

После выполнения RFE с перекрестной проверкой вы можете использовать выбранные функции для обучения вашей окончательной модели. В приведенном выше примере вы можете использовать атрибут `support_` экземпляра `RFECV`, чтобы получить логический массив выбранных функций. Затем вы можете использовать этот массив для выбора только соответствующих объектов из набора данных, а затем обучить свою конечную модель с использованием этих объектов.

Подбор гиперпараметров

Сначала давайте разберемся в разнице между параметрами модели и гиперпараметрами:

Параметры настраиваются в процессе обучения модели на данных. Например, это могут быть веса в линейной регрессии, нейронных сетях или структура решающего дерева.

Гиперпараметры — это характеристики модели, которые задаются до начала обучения: глубина решающего дерева, значение силы регуляризации в линейной модели, `learning rate` для градиентного спуска.

Примеры гиперпараметров включают коэффициент регуляризации, скорость обучения, количество скрытых слоев в нейронной сети и т.д.

Подбор гиперпараметров имеет цель найти наилучшие значения для гиперпараметров, которые позволят модели достичь наилучшей производительности и обеспечить хорошую обобщающую способность.

Для эффективного подбора гиперпараметров можно использовать различные методы, включая:

Grid Search

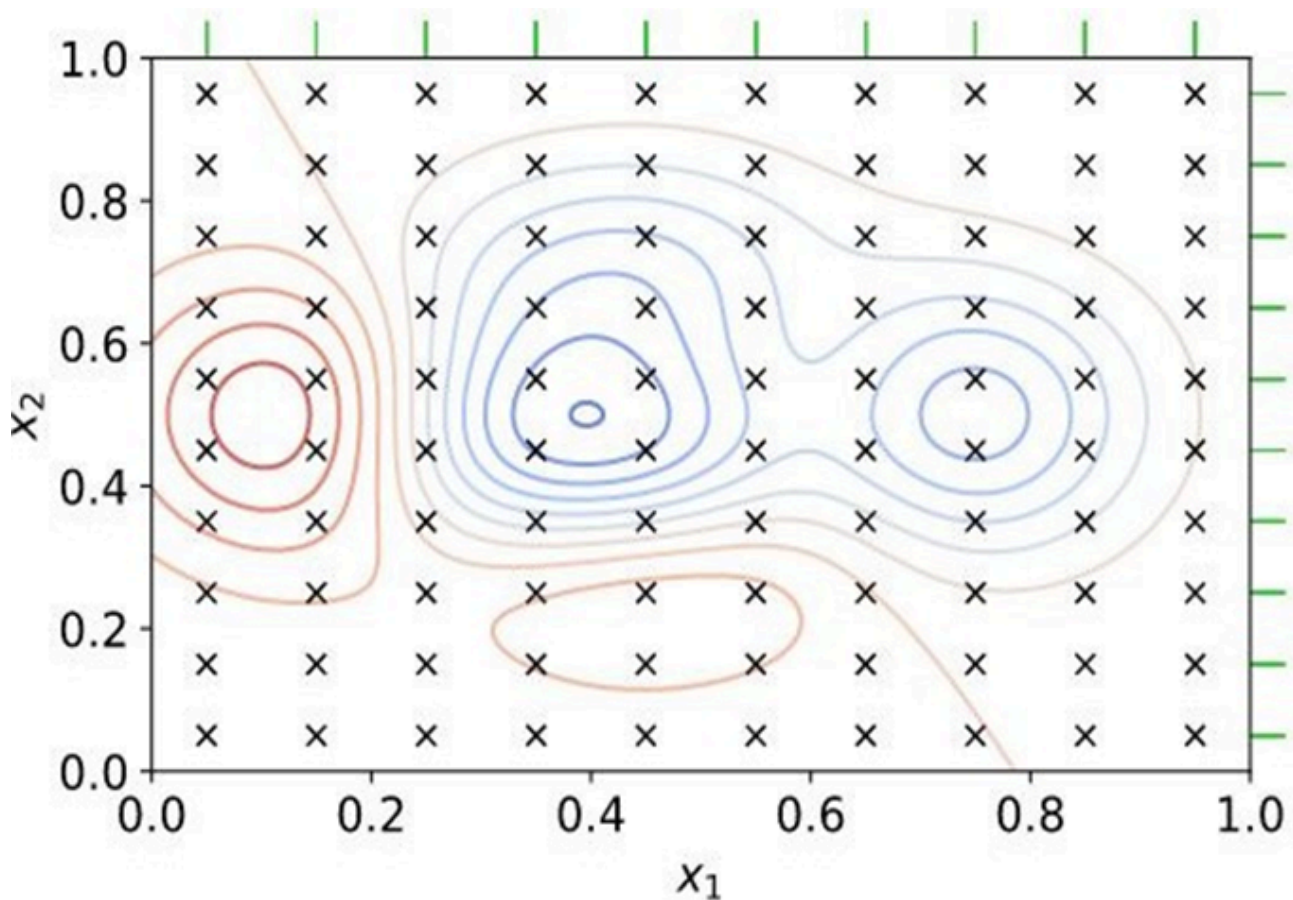
Один из самых естественных способов организации перебора наборов гиперпараметров — это сеточный перебор (Grid Search):

1. Фиксируются несколько значений для каждого гиперпараметра.
2. Перебираются все возможные комбинации значений различных гиперпараметров.
3. На каждой из этих комбинаций модель обучается и тестируется.
4. Выбирается комбинация, на которой модель показывает лучшее качество.

Примеры:

- Для метода ближайших соседей можно перебирать по сетке число соседей (например, от 1 до 20) и метрику, по которой будет измеряться расстояние между объектами выборки (евклидова, манхэттенская и т. д.).
- Для решающих деревьев можно перебирать по сетке различные комбинации значений максимальной глубины дерева и критериев ветвления (например, критерий Джини, энтропийный критерий).

Перебор некоторых значений гиперпараметров можно проводить на логарифмической шкале, так как это позволяет быстрее определить правильный порядок параметра и одновременно сократить время поиска. Такой подход можно использовать, например, для определения значения learning rate при градиентном спуске, значения регуляризации для линейной регрессии или метода SVM. Однако сразу становится очевидным естественное ограничение данного метода: если комбинаций параметров слишком много или каждое обучение/тестирование занимает много времени, алгоритм не сможет завершиться в разумные сроки.



```
# Определение параметров и их значений для перебора
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_leaf': [1, 2, 4]
}

# Создание модели и настройка с использованием решетчатого поиска
rf_model = RandomForestClassifier()
grid_search = GridSearchCV(rf_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Random Search

Если у вас возникает большое количество комбинаций параметров, то вы можете решить эту проблему различными способами:

- Уменьшить количество значений каждого гиперпараметра, однако это может привести к пропуску наилучшей комбинации.
- Уменьшить количество фолдов в кросс-валидации, но в этом случае оценка параметров будет более шумной.
- Оптимизировать параметры последовательно, а не перебирать их комбинации. Однако снова есть риск получить неоптимальное решение.
- Перебирать не все комбинации гиперпараметров, а только случайное подмножество.

Последний метод - Random Search. Для каждого гиперпараметра задается распределение, из которого выбирается его значение. Комбинация гиперпараметров формируется путем выбора значений из этих распределений (хорошие советы по выбору распределений можно найти в документации sklearn).

```

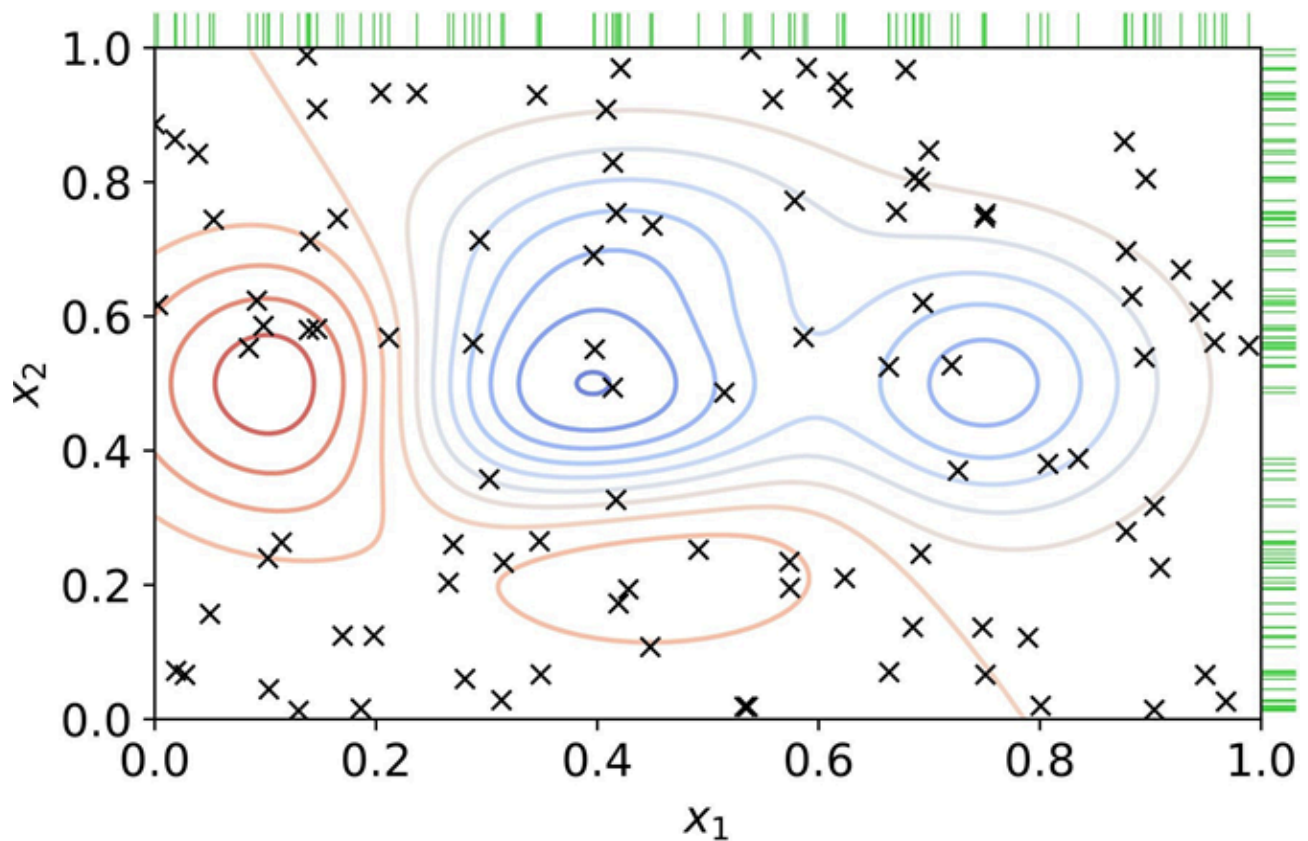
# Определение диапазонов значений для случайного поиска
param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 4]
}

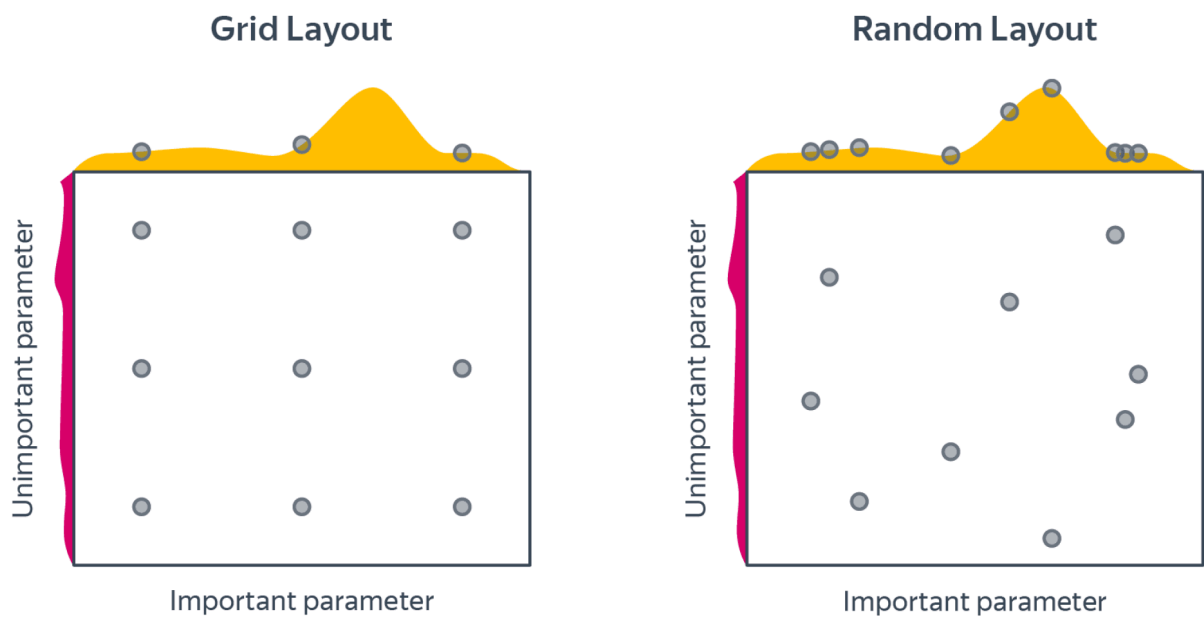
# Создание модели и настройка с использованием случайного поиска
rf_model = RandomForestClassifier()
random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist, n_iter=100, cv=5)
random_search.fit(X_train, y_train)

# Вывод наилучших гиперпараметров и оценки
print("Best Hyperparameters:", random_search.best_params_)
print("Best Cross-Validation Score:", random_search.best_score_)

```

Таким образом, случайный выбор следующей комбинации гиперпараметров позволяет найти оптимальную комбинацию за меньшее количество итераций.

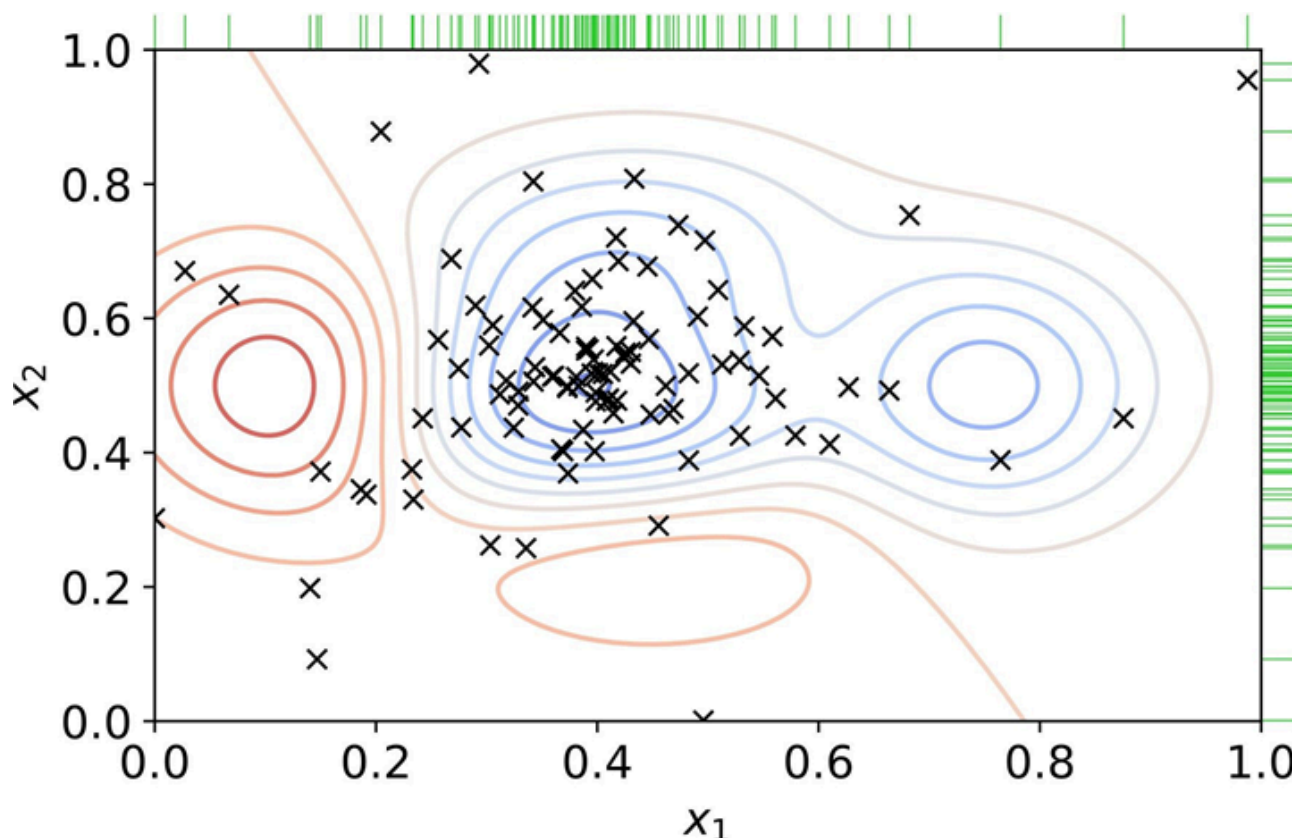




Качество нашей модели, зависящее от гиперпараметров, может быть представлено как функция многих переменных с нетривиальной поверхностью. Однако одна из переменных может оказывать гораздо меньшее влияние на эту поверхность, чем другая. Если бы мы знали, какой гиперпараметр является наиболее важным для производительности модели, мы бы рассмотрели больше возможных значений этого параметра. Однако зачастую у нас нет такой информации, поэтому мы ограничиваемся некоторым заранее заданным количеством значений для каждого гиперпараметра. Random Search может рассмотреть более разнообразные значения гиперпараметров за то же количество итераций, что и Grid Search. Таким образом, с большей вероятностью он найдет значения, которые оказывают наибольшее влияние на качество модели, и, следовательно, наилучшую комбинацию значений гиперпараметров.

Байесовская оптимизация

Байесовская оптимизация представляет собой метод, который объединяет вероятностные модели с методами оптимизации с целью эффективного поиска оптимальных гиперпараметров. Основанная на идее моделирования функции оценки производительности модели, эта методика использует эту модель для выбора следующей точки для оценки. Байесовская оптимизация обычно требует меньшего числа итераций, чем решетчатый или случайный поиск, чтобы достичь лучших результатов.



Модель байесовской оптимизации — это метод оптимизации, который помогает находить оптимальные значения для функций, которые нелегко изменять и требуют большого числа вычислений.

Представьте, что у вас есть функция, которую вы хотите максимизировать или минимизировать. Вместо того чтобы просто пробовать случайные значения и проверять, как изменяется функция, модель байесовской оптимизации позволяет вам принимать лучшие решения на основе информации, полученной из предыдущих итераций.

Модель состоит из двух важных компонентов: гауссового процесса и алгоритма выбора следующей точки для проверки.

Гауссовый процесс — это математическая модель, которая описывает распределение возможных значений функции и включает основные характеристики, такие как среднее и дисперсия. Он может предсказывать, как функция будет вести себя вне известных точек.

Алгоритм выбора следующей точки использует предсказания гауссовского процесса, чтобы найти наилучшую точку для дальнейшей проверки. Он учитывает баланс между исследованием новых областей и использованием уже полученных данных для уточнения предсказаний.

Модель байесовской оптимизации итеративно проверяет новые точки и обновляет гауссовский процесс на основе полученных результатов. Она стремится находить оптимальные значения функции с помощью наименьшего числа проверок, экономя время и вычислительные ресурсы.

Таким образом, модель байесовской оптимизации помогает исследовать и оптимизировать функции, основываясь на статистическом анализе и информации из предыдущих итераций.

Пример кода на Python с использованием библиотеки scikit-optimize:

```
from skopt import BayesSearchCV
from skopt.space import Integer, Real

# Определение пространства поиска гиперпараметров
param_space = {
    'n_estimators': Integer(50, 200),
    'max_depth': Integer(10, 50),
    'min_samples_leaf': Integer(1, 4),
    'max_features': Real(0.1, 1.0, prior='uniform')
}

# Создание модели и настройка с использованием байесовской оптимизации
rf_model = RandomForestClassifier()
bayes_search = BayesSearchCV(rf_model, param_space, n_iter=50, cv=5)
bayes_search.fit(X_train, y_train)

# Вывод наилучших гиперпараметров и оценки
print("Best Hyperparameters:", bayes_search.best_params_)
print("Best Cross-Validation Score:", bayes_search.best_score_)
```

Сравним эти алгоритмы:

Grid Search. Эффективен, когда у вас есть только несколько гиперпараметров или вы смогли распараллелить его работу.

Преимущества:

- Самый простой в понимании и реализации.
- Легко распараллеливается.

Недостатки:

- Не использует результаты других итераций.
- Ограничен выбором заданной сетки.
- Длительно работает, если выполняется последовательный перебор по сетке. Нет гарантии на необходимое количество итераций.

В пользу этого метода можно сказать, что часто на практике приходится выполнять перебор гиперпараметров вручную (если один экземпляр вашей модели обучается две недели и требует много ресурсов) или по очень маленькой сетке. Так что этот метод все еще актуален :)

Random Search. Этот метод является некоторым усложнением по сравнению с методом сеточного поиска, но при этом он оказывается намного более эффективным.

Преимущества:

- Случайный перебор по сетке позволяет находить оптимальные гиперпараметры более эффективно, чем сеточный поиск. Это связано, в частности, с возможностью задавать непрерывные параметры в виде распределения, а не перечислять значения заранее.
- Легко параллелизуется.
- Можно усилить эффективность, используя квазислучайные распределения при семплировании.

Недостатки:

- Не использует результаты других итераций.
- Ограничен выбором заданной сетки, хотя в некоторых случаях менее жестко, чем сеточный поиск.

Преимущества метода **Байесовской оптимизации:**

- Использует результаты предыдущих итераций.
- Может моделировать внутренние зависимости между гиперпараметрами, работая с ними в едином подмножестве, где число гиперпараметров.
- Может расширять изначально заданные границы множества поиска гиперпараметров.
- При достаточном количестве итераций достигает более высокого качества по сравнению с методом случайного поиска.

Недостатки метода Байесовской оптимизации:

- Параллелизация требует дополнительных усилий.
- В случае нераспараллеленной работы требуется много времени, так как для каждой итерации необходимо строить вероятностную модель. Если используются гауссовские процессы, сложность составляет порядка n^3 , где n - число гиперпараметров.
- Для работы с категориальными гиперпараметрами требуются нетривиальные приемы.

Итоги:

1. Генерация признаков: Эффективная генерация признаков является одним из ключевых моментов в анализе данных. Хорошо подобранные признаки могут

значительно улучшить качество модели. Для генерации признаков можно использовать различные методы, такие как полиномиальные и тригонометрические преобразования, производные и логарифмы, агрегацию данных и многое другое.

2. Методы отбора признаков: Методы отбора признаков помогают выявить наиболее информативные и значимые признаки для построения модели. Отбор признаков может осуществляться на основе статистических метрик, таких как корреляция, mutual information, chi-square и др., а также на основе моделей машинного обучения, таких как случайный лес или логистическая регрессия. Это помогает улучшить интерпретируемость модели и уменьшить размерность данных.

3. Подбор гиперпараметров: Гиперпараметры это параметры модели, которые должны быть настроены до обучения и влияют на ее производительность. Подбор гиперпараметров может быть осуществлен с использованием методов перебора, таких как сеточный поиск и случайный поиск, а также с использованием более сложных алгоритмов оптимизации, таких как градиентный спуск или байесовская оптимизация. Правильно подобранные гиперпараметры позволяют достичь более высокой точности и улучшить обобщающую способность модели.

4. Практическая и теоретическая важность подходов: Генерация признаков, методы отбора признаков и подбор гиперпараметров являются важными этапами в процессе построения модели машинного обучения. Правильный выбор и оптимизация признаков и гиперпараметров позволяют достичь лучших результатов и повысить качество модели. Теоретические основы этих подходов также важны для понимания принципов работы моделей и их способности улавливать информацию и делать предсказания.

В целом, генерация признаков, методы отбора признаков и подбор гиперпараметров являются важными инструментами для повышения качества моделей машинного обучения и обеспечения их интерпретируемости и универсальности. Эти подходы имеют как практическую, так и теоретическую важность, помогая разработчикам и исследователям построить более эффективные модели.

Что можно почитать еще?

1. [Оптимизация](#)
2. [Поиск по сетке](#)

3. [Практический подход к оптимизации](#)

Используемая литература

1. <https://habr.com/ru/companies/otus/articles/754402/>
2. https://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-search
3. <https://questu.ru/articles/53435/>