# Случайный лес и оценка значимости признаков

## Цель занятия

В результате обучения на этой неделе:

- вы разберете различные техники ансамблирования и теоретические предпосылки к их применению;
- познакомитесь с процедурой bootstrap, на основе которой строится метод ансамблирования бэггинг;
- рассмотрите принципы построения случайного леса;
- узнаете, как использовать технику блендинга, которая позволяет строить ансамбли не параллельно друг другу, а последовательно;
- разберете один из самых популярных методов ансамблирования — стекинг;
- узнаете, в чем состоит дилемма смещения-дисперсии (bias-variance tradeoff);
- вы узнаете способы оценки значимости признаков для определенной модели или класса моделей — permutation importance, специфичные для деревьев методы (Gain, Frequency, Cover), SHAP;
- рассмотрите на практике, как оценить влияние признаков на прогноз модели машинного обучения с помощью библиотеки SHAP.

## План занятия

## Конспект занятия

### 1. Техника ансамблирования

Процедура Bootstrap

Рассмотрим некоторую выборку X. Мы можем выбирать из нее объекты с повторениями. То есть берем объект, изучаем его признаковое описание, а затем возвращаем объект обратно. На втором шаге мы опять берем случайный объект. Из-за возвращения предыдущего объекта обратно есть небольшая вероятность, что мы возьмем тот же.

**Bootstrap** — техника порождений нескольких выборок из одной исходной с помощью выбора с повторениями.

Логично, что бутстрапированные выборки похожи друг на друга, потому что есть совпадающие объекты. Такие выборки обычно используются в прикладной статистике для оценки различных параметров распределения, доверительных интервалов. Мы будем использовать Bootstrap для обучения сразу нескольких моделей.

Пусть у нас есть несколько моделей, и каждая обладает некоторой ошибкой:

$$\varepsilon_j(x) = b_j(x) - y(x), \; j = 1, \ldots, N$$

Здесь:

- $b_j(x)$ — предсказание j-й модели;
- $y(x)$ — истинная зависимость, которую мы пытаемся описать;
- $N$ – количество бутстрапированных выборок.

На каждой из выборок обучено по одной модели, то есть всего обучено $N$ моделей.

Тогда средняя квадратичная ошибка — мат. ожидание по x:

$$\mathrm{E}_x\Big(b_j(x) - y(x)\Big)^2 = \mathrm{E}_x\varepsilon_j^2(x)$$

Средняя ошибка для $N$ моделей:

$$E_1 = \frac{1}{N}\sum_{j=1}^{N} \mathrm{E}_x\varepsilon_j^2(x).$$

На практике вместо мат. ожидания будем использовать выборочное среднее, поскольку нам не всегда будет доступно все распределение.

Добавим некоторые предположения:

- Ошибки не смещенные:

  $$\mathrm{E}_x \varepsilon_j(x) = 0$$

- Ошибки некоррелированы между собой:

  $$\mathrm{E}_x \varepsilon_i(x)\varepsilon_j(x) = 0, \quad i \neq j$$

То есть наши модели ошибаются случайным образом.

Мы предполагаем, что ошибка случайна.

Такие предположения приводят к тому, что итоговая оценка на предсказание модели — теперь усредненная модель из всех:

$$a(x) = \frac{1}{N}\sum_{j=1}^{N} b_j(x)$$

То есть у нас теперь есть некоторая модель, которая агрегирует данные всех моделей, по сути строит их ансамбль, и выдает усредненное предсказание.

Тогда ошибка модели $a(x)$ — ошибка ансамбля из N моделей:

$$E_N = \mathrm{E}_x\left(\frac{1}{N}\sum_{j=1}^{n} b_j(x) - y(x)\right)^2 = \mathrm{E}_x\left(\frac{1}{N}\sum_{j=1}^{N} \varepsilon_j(x)\right)^2 = \frac{1}{N^2}\mathrm{E}_x\left(\sum_{j=1}^{N} \varepsilon_j^2(x) + \sum_{i\neq j} \varepsilon_i(x)\varepsilon_j(x)\right) = \frac{1}{N}E_1 \quad (1)$$

Ошибка становится меньше в N раз.

Если ошибка так сильно уменьшается, почему бы тогда не использовать ансамблирование везде? Но мы предположили, что ошибки не зависимы друг от друга и не скоррелированы, а это на практике встречается довольно редко.

Но даже если ошибки имеют корреляцию меньше, чем 1, сумма (1) все равно будет меньше, чем ошибка одной модели. Ошибку мы все равно понизим, но, конечно, не в N раз.

**Пример**. «Мудрость толпы». В некотором городе собралась на площади толпа крестьян. Крестьянам предложили задачу: привели быка и сказали, что тот, кто сможет оценить вес быка с точностью до фунта, получит этого быка. Показания людей записывали. Все оценки усреднили и получили довольно точный вес быка.

Идея «мудрости толпы» в том, что множество крестьян не советовались друг с другом. Каждый из них делал оценку веса на глаз, не слушая других. Конечно, оценки должны иметь смысл. Крестьяне имели опыт оценки на глаз в сельском хозяйстве.

Бэггинг

На основе процедуры Bootstrap можно построить бэггинг:

Bagging = Bootstrap aggregating

То есть порождаем с помощью бутстрапирования несколько выборок. Затем обучаем на этих выборках и агрегируем предсказание, усредняя между собой. Это позволяет снизить дисперсию предсказания.

Бэггинг на практике используется часто с решающими деревьями. Решающие деревья очень хорошо подходят под бэггинг, потому что за счет переобучения они становятся сильно непохожими друг на друга. Из-за этого их ошибки не будут полностью скоррелированы. Усредняя несколько таких переобученных деревьев, мы повышаем точность предсказания.

## 2. RSM

Случайный лес

Как сделать деревья еще более непохожими друг на друга?

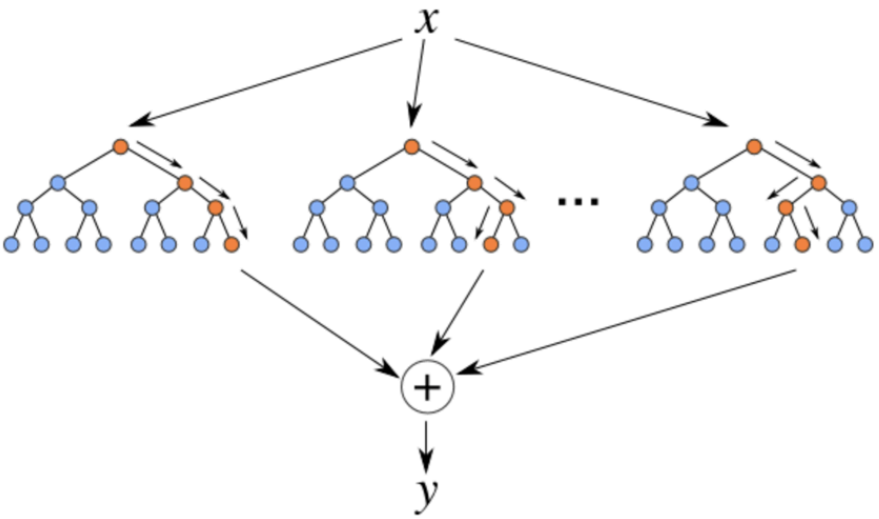
Можно воспользоваться **методом случайных подпространств** (Random Subspace Method, RSM). Мы будем использовать почти такой же подход, как в Bagging, но случайным образом выбирать не объекты, а признаки.

То есть мы взяли подвыборку, и эта выборка описывается только, допустим, пятью признаками из доступных восьми. Признаки выбраны случайным образом.

Или на каждом шаге построения дерева мы можем случайным образом выбирать признаковое подмножество. То есть мы теперь не позволяем на каждом шаге

выбирать дереву наилучший признак из всех с точки зрения разбиения. Мы позволяем выбирать наилучший только из случайно выбранных. Это делает дерево более «случайным», но при этом дерево продолжит работать.

За счет этого получаем много деревьев, которые сильно не похожи друг на друга:




Деревья рандомизированы и на уровне выборки, и на уровне признаков.

Теперь несколько непохожих обученных деревьев можно усреднить и тем самым значительно уменьшить ошибку.

Такой метод называется **случайным лесом (Random Forest)**:

Bagging + RSM = Random Forest.

Это один из наиболее сильных, эффективных методов в машинном обучении. Является классической моделью машинного обучения.

Метод случайного леса, поскольку это решающие деревья, позволяет работать и с задачей классификации, и с задачей регрессии.

Random Forest — это техника, которая использует слабость деревьев в качестве их силы.

В случайном лесе все деревья строятся независимо друг от друга. Именно за счет этого достигается результат снижения ошибки.

Свойства случайного леса

Метод случайного леса обладает несколькими свойствами:

- Он универсален. Подходит задачам регрессии, классификации и ряду других.
- Довольно быстрый, эффективно обучается.
- Есть некоторые модификации:
    - Extremely Randomized Trees. Позволяет работать с еще более зашумленной выборкой.
    - Isolation Forest. Метод для детекции аномалий.
- OOB (Out of Bag estimation) — оценка на объектах, не попавших в выборку. Каждое дерево учится на бутстрапированной выборке, которая содержит в себе некоторые объекты из исходной выборки. Для каждой бутстрапированной выборки есть множество объектов, которые в нее не попали. Соответственно, эти объекты не попали в обучающую выборку для конкретного дерева. По сути, для этого дерева данные объекты являются валидационными. Если ансамбль достаточно большой, то для каждой точки из обучающей выборки найдется несколько деревьев, которые в себя эту точку не включают для обучения. Значит, с помощью этих деревьев можно сделать предсказание на тех точках, которые не попали в обучающую выборку:

$$OOB = \sum_{i=1}^{l} L\left(y_i, \frac{1}{\sum_{n=1}^{N}\left[x_i \notin X_n\right]} \sum_{n=1}^{N}\left[x_i \notin X_n\right] b_n\left(x_i\right)\right)$$

  OOB позволяет в некотором роде избавиться от деления выборки на train и validation. Теперь в процедуре Bootstrap мы уже неявным образом это делаем.

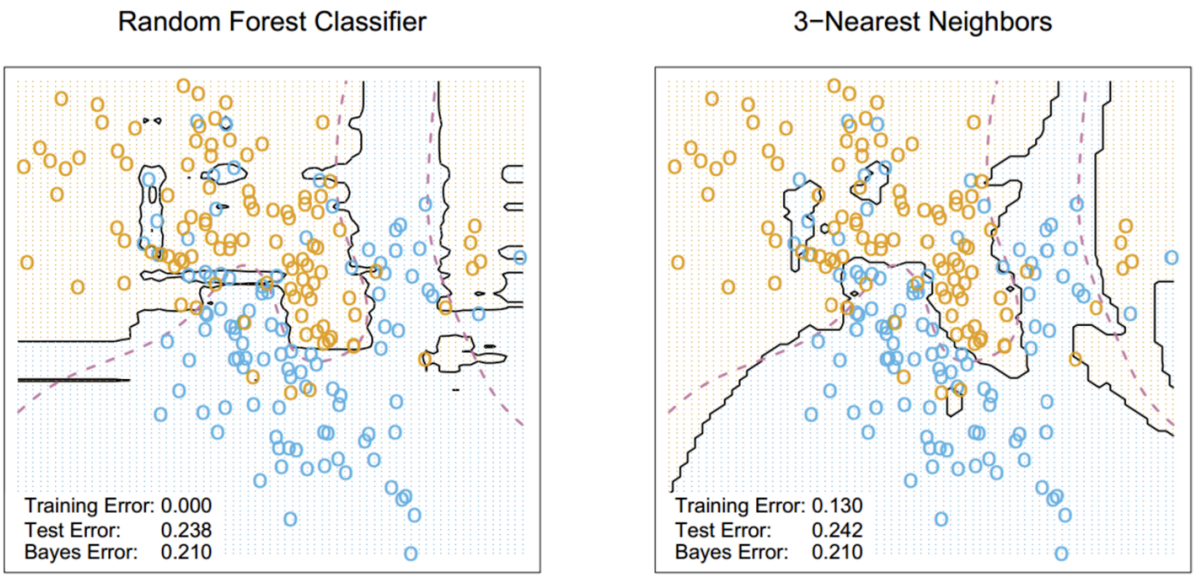
**Домашнее задание**. Посчитать, с какой вероятностью объект не попадет в бутстрапированную выборку.

Поскольку случайный лес — алгоритм ансамблирования деревьев, то он неплохо работает, если у нас сильно скоррелированы признаки.

Также случайный лес хорошо работает с пропущенными значениями (как и решающие деревья по отдельности).

Случайный лес и KNN

**Пример**.




В важных местах разделения границы похожи.

Случайный лес некоторым образом аппроксимирует метод KNN. Решающее дерево делит признаковое пространство на подобласти, каждой из которых приписывает некоторое значение. KNN для каждой точки в признаковом пространстве находит значение целевой функции, как некоторое усреднение из ближайших соседей. Если достаточно много соседей и достаточно большая глубина в дереве, картинки в примере практически совпадут.

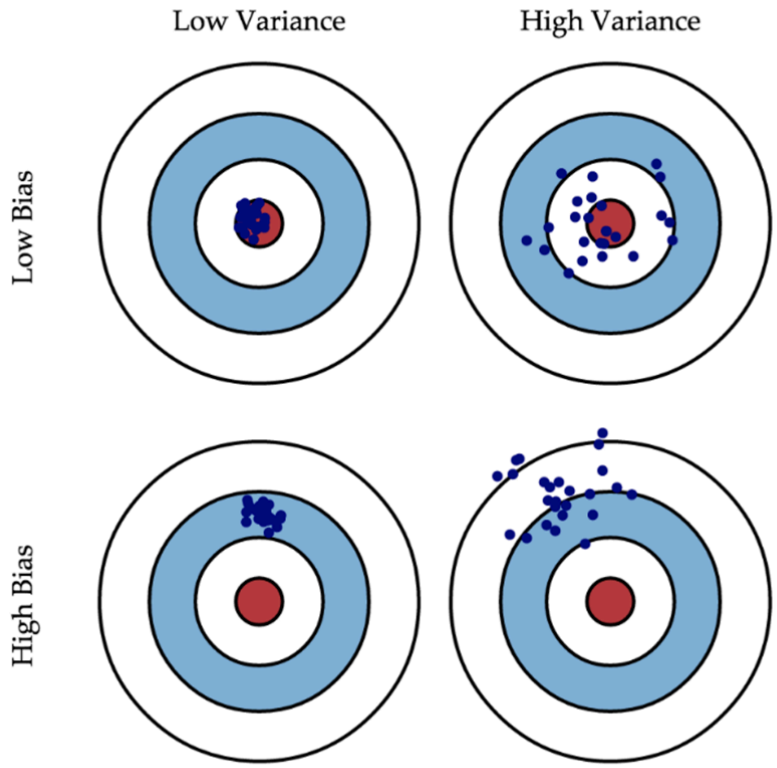
Но в отличие от KNN случайный лес не является квадратичной моделью. У нас нет необходимости считать расстояние до всех точек на каждом объекте.

Мы можем в случайном лесе не какую-нибудь энтропию снижать, а на каждом шаге пытаться отделить одну точку от всех остальных — так работает Isolation Forest. Это сделать проще для тех точек, которые находятся далеко от всех остальных. Если мы построим много деревьев и посчитаем их среднюю глубину, чтобы отделить некоторую точку, то можно увидеть, что для большинства точек глубина достаточно большая. Для аномальных точек средний путь будет гораздо короче. Это способ детекции аномалий.

### 3. Дилемма смещения

Дилемма смещения (bias-variance tradeoff) или дилемма смещения разброса — это попытка свести к минимуму источники ошибок, которые не позволяют алгоритмам обучения с учителем делать правильные предсказания за пределами своего обучающего набора.

Рассмотрим классическую картинку дилеммы смещения — попадание дротиком по цели:

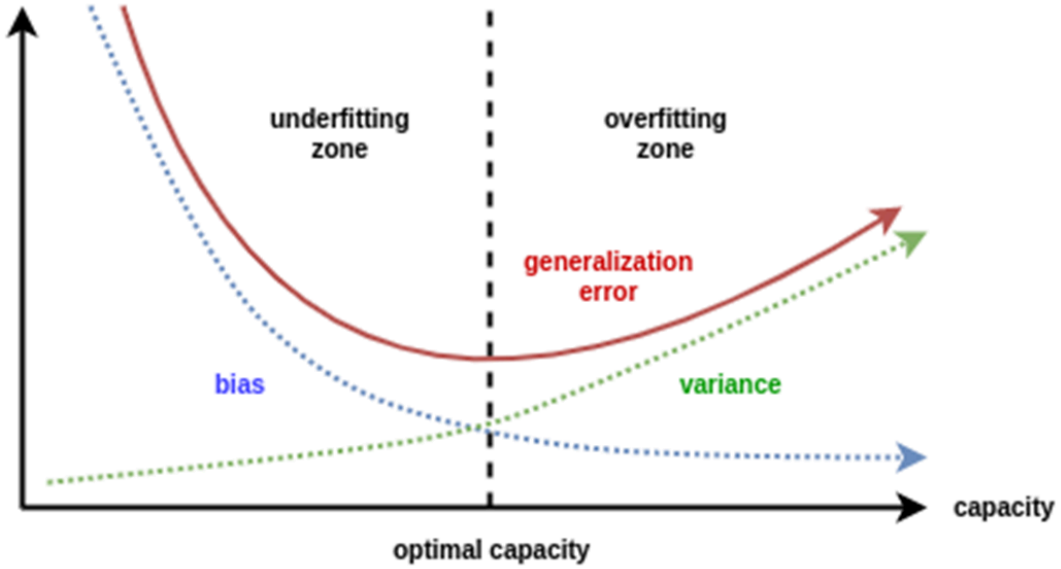



В данной задаче человек — это модель. Он как-то научился кидать дротики. Может кидать дротики достаточно кучно (low variance, предсказания достаточно стабильны) и достаточно близко к цели (low bias, небольшое смещение от центра). На практике в дилемме смещения получается минимизировать либо bias, либо variance.

Линейные модели достаточно простые и не всегда могут точно предсказать целевую переменную, то есть могут часто ошибаться на какую-то величину. Если зависимость нелинейная, то оценка линейной модели будет смещенной, то есть bias будет высокий. При этом линейные модели довольно стабильные, то есть variance низкий.

Если взять решающее дерево, особенно переобученное решающее дерево, у него bias будет маленький. Но если мы чуть-чуть поменяем наш датасет, то предсказания модели «поедут». Дерево является неустойчивой моделью, у него высокий variance.

Общая картина выглядит следующим образом:




Простые модели достаточно устойчивы, но не могут точно предсказывать целевую переменную на сложных задачах. Сложные модели — наоборот.

Суммарная ошибка — сумма bias и variance. Рассмотрим пример для средней квадратичной ошибки, получим мат. ожидания по всем переменным:

$$L(\mu) = \mathrm{E}_{x,y}\left[(y - \mathrm{E}[y|x])^2\right] + \mathrm{E}_x\left[\left(\mathrm{E}_X[\mu(X)] - \mathrm{E}[y|x]\right)^2\right] + \mathrm{E}_x\left[\mathrm{E}_X\left[\left(\mu(X) - \mathrm{E}_X[\mu(X)]\right)^2\right]\right]$$

Здесь:

- первое слагаемое — шум, который вообще есть в данных;
- второе слагаемое — bias;
- третье слагаемое — variance.

Модели могут либо точно описывать данные, но при этом быть неустойчивыми, либо быть устойчивыми, но ошибаться сильнее. Есть оптимальная точка — optimal capacity, модель оптимальной сложности, которая уже достаточно хорошо описывает данные и не сильно переобучается.

### 4. Блендинг

Блендинг или смешивание (blending) — еще одна техника ансамблирования. Достаточно широко применяется на практике и достаточно интуитивна. В ней нет явных теоретических обоснований. Скорее это уже попытка построить ансамбль моделей не параллельно друг другу (когда все модели не знали о существовании друг друга), а последовательно. Каждая следующая модель будет некоторым образом помогать предыдущей.

Глядя на случайный лес или бэггинг, почему мы усредняем модели с одинаковыми весами? Возможно, некоторые модели чуть более полезны, чем другие.

Каждая модель умеет делать предсказание. При помощи наших моделей попробуем улучшить итоговое предсказание, но не просто методом обычного усреднения.

**Пример**. Пусть у нас есть несколько моделей. Возьмем обучающую выборку и поделим на две части:

| Name | Age | Statistics (mark) | Python (mark) | Eye color | Native language | Target (mark) |
|---|---|---|---|---|---|---|
| John | 22 | 5 | 4 | Brown | English | 5 |
| Aahna | 17 | 4 | 5 | Brown | Hindi | 4 |
| Emily | 25 | 5 | 5 | Blue | Chinese | 5 |
| Michael | 27 | 3 | 4 | Green | French | 5 |
| Some student | 23 | 3 | 3 | NA | Esperanto | 2 |

На желтой части выборки обучим M базовых алгоритмов. Базовый алгоритм — несколько различных моделей (KNN, наивный Байес, SVM, логистическая регрессия, решающее дерево, блуждающий лес). После обучения эти модели могут делать предсказания.

Возьмем синюю часть обучающей выборки и для каждого объекта сделаем предсказание каждой из наших моделей. Для синей выборки получим M предсказаний для каждого объекта:

| f1 | f2 | f3 | f4 | f5 | Target |
|---|---|---|---|---|---|
| 2.2 | 8 | 17 | 5 | 44 | 5 |
| 7.2 | 21 | 33 | 4 | 9 | 2 |

По сути, у нас теперь есть новая обучающая выборка, которая находится в новом признаковом пространстве. Теперь каждый из новых признаков — это предсказание одной из M моделей, которые были обучены на желтой части выборки.

Теперь мы можем обучить какую-нибудь другую модель, например, взвешенное среднее:

$$\hat{f}(x) = \sum_{i=1}^{M} \rho_i f_i(x)$$

Здесь $f_i(x)$ — предсказания моделей, $\rho_i$ — веса.

При этом $\sum_{i=1}^{M} \rho_i = 1, \rho_i \in [0, 1] \; \forall i.$

То есть получаем выпуклую комбинацию всех наших предсказаний.

Это и есть техника Blending, и она позволяет усреднить несколько моделей между собой. На практике неплохо работает, если есть несколько хороших моделей. Например, в соревновательном машинном обучении.

В формуле для взвешенного среднего можно сделать $\rho$ зависимым от $x$ и получить смесь экспертов, но эта техника сложна в обучении.

Блендинг также работает как в задаче классификации, так и в задаче регрессии.

**Замечание**:

1. В смешивании мы усредняем достаточно сложные модели, которые не похожи друг на друга. Если модели будут простыми, то усреднение не даст преимущества.
2. Плюсы и минусы метода смешивания:

    + Каждая из моделей в смешивании достаточно сильная, и может сама по себе работать. Поэтому интуитивно понятно, почему при усреднении нескольких сильных моделей мы получаем хороший результат.

    + Усреднение приводит к тому, что если одна модель сильно ошибалась в каком-то определенном куске признакового пространства, после усреднения эта ошибка уже будет не так существенно влиять на результат.

    - Линейной композиции моделей не всегда может быть достаточно.

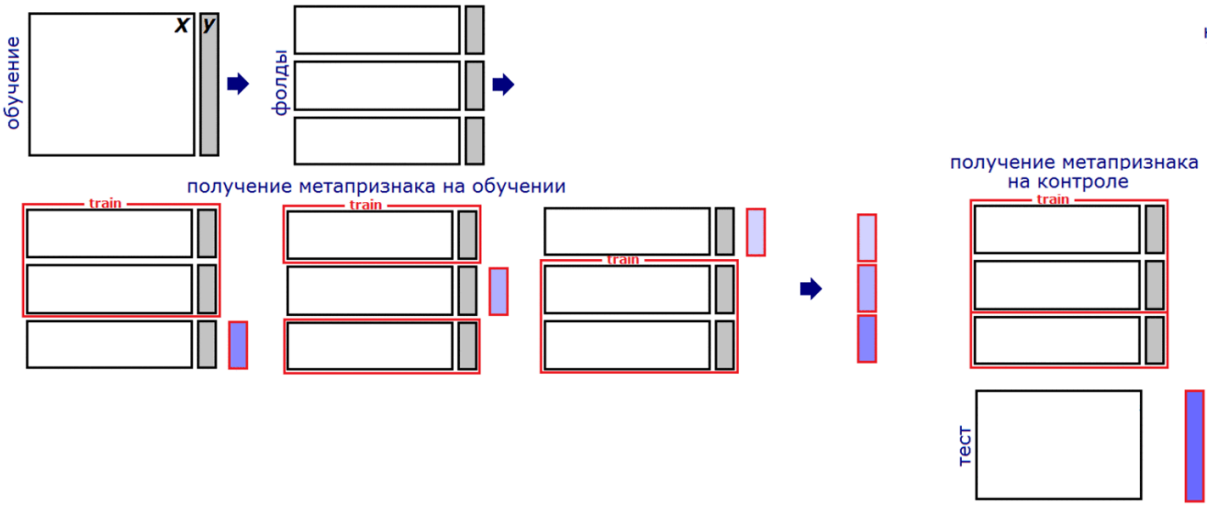
    - Приходится разбивать данные на части, а это бывает достаточно трудоемко. Кроме того, если оставить мало на обучение линейной композиции моделей, то можно переобучиться под нее. Если очень много, то мало данных для базовых алгоритмов. Чтобы это починить, нужно развить идею смешивания до стекинга.

## 5. Стекинг

Для смешивания одной линейной композиции моделей может быть недостаточно для решения сложной задачи.

Возьмем некоторую выборку. Мы бы не хотели отделять от нее кусок, потому что разделять данные довольно трудоемкий процесс. И мы бы хотели использовать не только линейный ансамбль из моделей.

Можно вспомнить, как работает кросс-валидация, взять данные и разбить их на фолды:




При этом мы можем обучать каждый из базовых алгоритмов на первых двух фолдах и сделать предсказание на оставшемся (нижнем), затем обучить на верхнем и нижнем и сделать предсказание для среднего. Затем сделать обучение на двух нижних и сделать предсказание для верхнего. Получится, что для всей обучающей выборки есть предсказание от десяти базовых алгоритмов, и при этом мы не теряем данные. У нас есть предсказания для всей обучающей выборки.

Далее мы можем взять наши модели, обучить теперь на всех данных полностью и предсказать с помощью них какие-то метапризнаки для тестовой части выборки.

Теперь каждый объект в выборке описан в терминах десяти предсказаний от десяти моделей, и у нас все еще есть истинный таргет. А после этого мы можем сделать еще одну итерацию стекинга, поскольку мы получили новую обучающую выборку для построения метамодели.

Итог: у нас метаалгоритм — любой алгоритм машинного обучения, какой был.

Преимущества и недостатки стекинга:

+ Это очень сильный метод. На практике особенно для табличных данных хорошо работает. Стекинг некоторым образом предвосхищает нейронные сети.

+ Очень популярен в различных соревнованиях по машинному обучению. С помощью стекинга можно очень сильно повысить качество предсказания модели.

+ Ничто не мешает использовать стекинг несколько раз подряд, то есть делать стекинг над стекингом. Как правило, они бывают глубиной три или четыре шага. С каждым новым шагом улучшение предсказания будет все ниже, все выше будет вероятность переобучиться.

- На каждом фолде метапризнаки предсказываются различными моделями. Их обучали на разных данных, и они будут различны. А, значит, эти метапризнаки обладают различными свойствами, приходят из различных распределений. Мы ломаем гипотезу IID. Если сильно маленькие фолды, модель может не заработать. Но регуляризация, которая ограничивает модели и не позволяет им быть сильно различными, может помочь.

- Очень сложно объяснить, что «натворила» модель. Понять ее работу трудно. В некоторых задачах, например, в кредитных рисках, понятность работы модели может быть решающим фактором.

Более подробно можно почитать про стекинг в блоге Александра Дьяконова.

**NB**. Базовые алгоритмы должны обладать хоть каким-то представлением о решаемой задаче. При ансамблировании моделей нужно помнить, что модели должны быть адекватными.

### 6. Оценка значимости признаков

Рассмотрим способы оценивания значимости признаков для определенной модели или для конкретного класса моделей.

### Permutation importance

Permutation importance — метод, который корректно оценивает значимость признаков, но неэффективен по вычислительной сложности.

Пусть есть некоторая выборка:

| Рост в возрасте 20 лет (см) | Рост в возрасте 10 лет (см) | … | Количество носков в наличии в возрасте 10 лет |
|---|---|---|---|
| 182 | 155 | … | 20 |
| 175 | 147 | … | 10 |
| … | … | … | … |
| 156 | 142 | … | 8 |

| | | | |
|---|---|---|---|
| 153 | 130 | ... | 24 |

Хочется оценить, насколько количество носков важно для целевой переменной — например, средний балл при обучении на курсе. Казалось бы, при чем тут носки? Есть модель. В ней можно увидеть, что количество носков дает какой-то вклад в предсказание. Как оценить этот вклад?

Перемешаем тот признак, который хотим оценить. Например, в таблице рост в возрасте 10 лет. И посмотрим, насколько от этого «испортится» качество модели. Мы не просто лишаем модель данного признака, мы закладываем некоторый «шум» в наши данные. То же самое можно сделать для каждого признака и увидеть, насколько каждый из них по отдельности важен.

Если выборка размера миллион объектов, метод работает крайне медленно. Поэтому на практике он используется нечасто, его приводят как некоторую теоретическую возможность оценить значимость.

### Специфичные для деревьев методы

Вспомним критерии информативности для деревьев.

**Gain** — насколько критерий изменился благодаря каждому из признаков.

**Frequency** — насколько часто признак использовался для разбиения. Если дерево глубокое, то чем чаще признак используется, тем более он важен.

**Cover** — взвешенная по количеству объектов частота использования признака для разбиения.

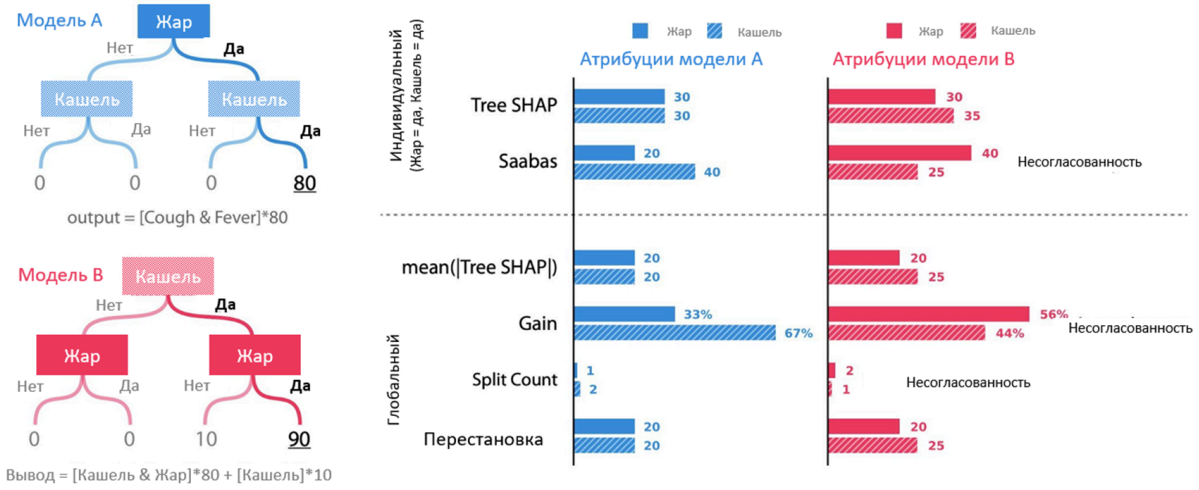
Все эти методы не очень корректны, они эвристические, поскольку в одном случае дают более интуитивно понятные и логичные выводы, а в другом случае — непонятные.

Для оценки значимости признаков могут использоваться различные методы, учитывающие специфику используемых моделей.

### Метод SHAP

Метод SHAP более корректный, он показывает явную значимость признаков. Для деревьев он позволяет сделать это естественным образом, поскольку построение деревьев позволяет использовать его полностью.

Рассмотрим две модели — A и B:




Эти модели пытаются оценить, насколько важны имеющиеся признаки, чтобы оценить, болеет человек или нет. Сначала построили дерево по тому, есть у человека жар или нет. А потом есть у него кашель или нет. Как можно понять, что признак важен? Оптимально иметь модели с этим признаком и без него. Но некоторые признаки важны только в группе друг с другом.

SHAP предлагает посмотреть на все возможные подвыборки признаков и построить на них модель. Для деревьев это реализуемо, ведь деревья могут работать с пропущенными значениями.

Рассмотрим $i$-тый признак. Согласно SHAP, его значимость будет равна:

$$\phi_i(p) = \sum_{S \subseteq N/\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(p(S \cup \{i\}) - p(S)),$$

где $p(S \cup \{i\})$ — это предсказание модели на выборке $S$ с использованием $i$-го признака.

Теперь мы можем оценить значимость абсолютно корректно. Для деревьев не требуется перестроения модели огромное количество раз.

Для сложных моделей типа нейронных сетей это работать не будет, а для линейных моделей SHAP вырождается в выражение для оценки значимости признаков — в коэффициенты в линейной функции, которая и является моделью.

**Важно!** Нельзя забывать про нормировку данных, поскольку в оценке значимости признаков коэффициенты зависят от шкалы.

Деревья к нормировке признаков нечувствительны, они каждый признак рассматривают по отдельности. Тем не менее нормировка данных полезна, она повышает вычислительную устойчивость модели.

При оценке значимости признаков их значимость оценивается для конкретной модели.

### 7. Feature importances

Рассмотрим вопрос интерпретируемости машинного обучения. В машинном обучении и глубоком обучении остро стоит проблема отсутствия информации о модели (например, нейронной сети), через которую пропускают данные и получают на выходе информацию. Совершенно непонятно, как работают нейроны и слои. Модель машинного обучения представляется черным ящиком.

Возьмем данные из медицинской базы данных:

```
# If running on Colab, uncomment this cell
# ! pip install pdpbox
# ! pip install eli5
# ! pip install SHAP
# ! wget
https://raw.githubusercontent.com/neychev/fall19_madmo_adv/master/week02_Boosting_
and_importances/data/medical.csv
# ! mkdir data
# ! mv medical.csv data
```

```
import pandas as pd
data = pd.read_csv('data/medical.csv')
data.columns
```

```
>>> Index(['time_in_hospital', 'num_lab_procedures', 'num_procedures',
       'num_medications', 'number_outpatient', 'number_emergency',
       'number_inpatient', 'number_diagnoses', 'race_Caucasian',
```

'race_AfricanAmerican', 'gender_Female', 'age_[70-80)', 'age_[60-70)',
'age_[50-60)', 'age_[80-90)', 'age_[40-50)', 'payer_code_?',
'payer_code_MC', 'payer_code_HM', 'payer_code_SP', 'payer_code_BC',
'medical_specialty_?', 'medical_specialty_InternalMedicine',
'medical_specialty_Emergency/Trauma',
'medical_specialty_Family/GeneralPractice',
'medical_specialty_Cardiology', 'diag_1_428', 'diag_1_414',
'diag_1_786', 'diag_2_276', 'diag_2_428', 'diag_2_250', 'diag_2_427',
'diag_3_250', 'diag_3_401', 'diag_3_276', 'diag_3_428',
'max_glu_serum_None', 'A1Cresult_None', 'metformin_No',
'repaglinide_No', 'nateglinide_No', 'chlorpropamide_No',
'glimepiride_No', 'acetohexamide_No', 'glipizide_No', 'glyburide_No',
'tolbutamide_No', 'pioglitazone_No', 'rosiglitazone_No', 'acarbose_No',
'miglitol_No', 'troglitazone_No', 'tolazamide_No', 'examide_No',
'citoglipton_No', 'insulin_No', 'glyburide-metformin_No',
'glipizide-metformin_No', 'glimepiride-pioglitazone_No',
'metformin-rosiglitazone_No', 'metformin-pioglitazone_No', 'change_No',
'diabetesMed_Yes', 'readmitted'],
dtype='object')

```
data.describe()
```

>>>

| | time_in_hospital | num_lab_procedures | num_procedures | num_medications | number_outpatient | number_emergency | number_inpatient | number_diagnoses | rea |
|---|---|---|---|---|---|---|---|---|---|
| count | 25000.000000 | 25000.00000 | 25000.000000 | 25000.000000 | 25000.000000 | 25000.000000 | 25000.00000 | 25000.000000 | 25000 |
| mean | 4.395640 | 42.96012 | 1.341080 | 15.988440 | 0.365920 | 0.203280 | 0.64300 | 7.420160 | 0 |
| std | 2.991165 | 19.76881 | 1.705398 | 8.107743 | 1.224419 | 0.982973 | 1.26286 | 1.940932 | 0 |
| min | 1.000000 | 1.00000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.00000 | 1.000000 | 0 |
| 25% | 2.000000 | 31.00000 | 0.000000 | 10.000000 | 0.000000 | 0.000000 | 0.00000 | 6.000000 | 0 |
| 50% | 4.000000 | 44.00000 | 1.000000 | 15.000000 | 0.000000 | 0.000000 | 0.00000 | 8.000000 | 0 |
| 75% | 6.000000 | 57.00000 | 2.000000 | 20.000000 | 0.000000 | 0.000000 | 1.00000 | 9.000000 | 1 |
| max | 14.000000 | 126.00000 | 6.000000 | 81.000000 | 36.000000 | 64.000000 | 21.00000 | 16.000000 | 1 |

В данной таблице записаны истории болезни людей. Наш таргет — это колонка readmitted. Она означает, что пациент с подобной историей болезни уже был.

Построим модель машинного обучения, зададим ей признаки, таргетную колонку.
Начнем с модели случайного леса:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

y = data.readmitted

base_features = [c for c in data.columns if c != "readmitted"]

X = data[base_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
my_model = RandomForestClassifier(n_estimators=30, random_state=1).fit(train_X,
train_y)
```

В sklearn можно посмотреть важность всех признаков:

```
my_model.feature_importances_
```

По этому атрибуту мы получаем набор чисел, это веса каждого признака.

Построим следующий датасет:

```
import numpy as np
pd.DataFrame(my_model.featyre_importances_, index=X.columns)
```

>>>

| | 0 |
|---|---|
| time_in_hospital | 0.076277 |
| num_lab_procedures | 0.121372 |
| num_procedures | 0.052653 |
| num_medications | 0.107680 |
| number_outpatient | 0.023855 |
| ... | ... |
| glimepiride-pioglitazone_No | 0.000009 |
| metformin-rosiglitazone_No | 0.000000 |
| metformin-pioglitazone_No | 0.000000 |
| change_No | 0.017159 |
| diabetesMed_Yes | 0.011410 |

Левая колонка — признаки, правая колонка 0 — вес признака.

Чтобы создать массив признаков, нужно отсортировать индексы от максимального к минимальному:

```
pd.DataFrame(my_model.featyre_importances_,
index=X.columns).index[np.argsort(-my_model.feature_importances)]
```

Рассмотрим пакет eli5, это обертка над sklearn. Он помогает объяснить, почему в классификаторов именно такие результаты:

```
# Use permutation importance as a succinct model summary
# A measure of model performance on validation data would be useful here too

import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(my_model, random_state=1).fit(val_X, val_y)
eli5.show_weights(perm, feature_names = val_X.columns.tolist())
```

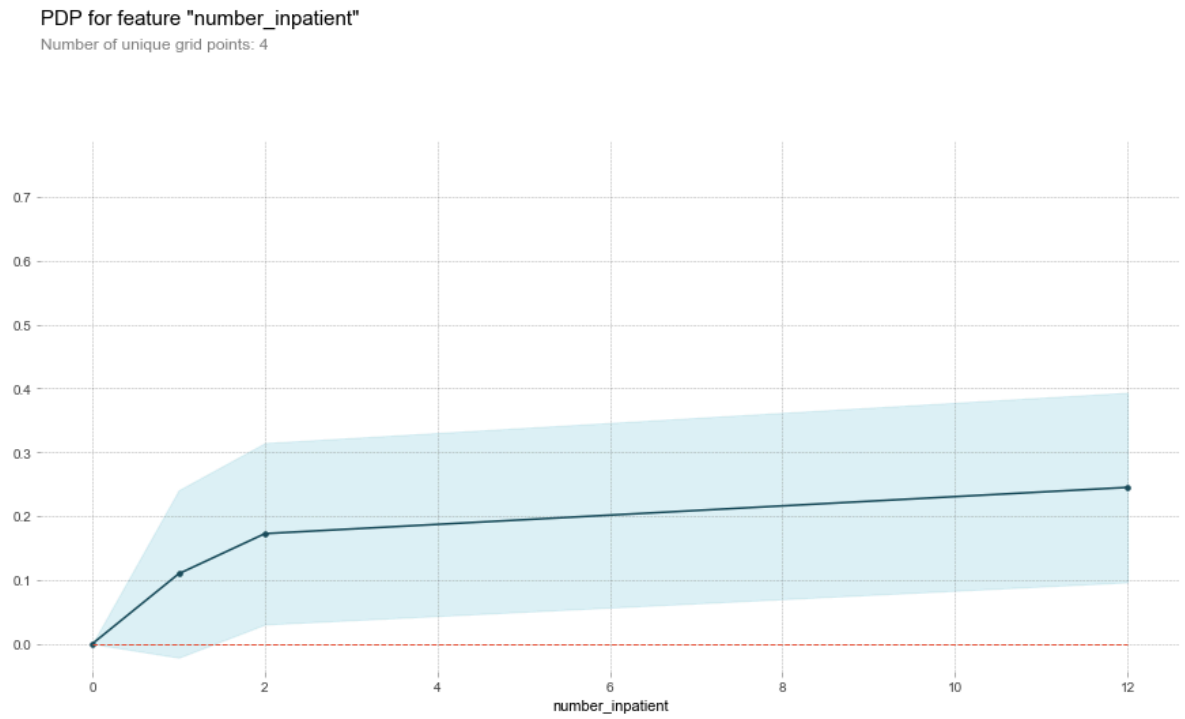
| Weight | Feature |
|---|---|
| 0.0451 ± 0.0068 | number_inpatient |
| 0.0087 ± 0.0046 | number_emergency |
| 0.0062 ± 0.0053 | number_outpatient |
| 0.0033 ± 0.0016 | payer_code_MC |
| 0.0020 ± 0.0016 | diag_3_401 |
| 0.0016 ± 0.0031 | medical_specialty_Emergency/Trauma |
| 0.0014 ± 0.0024 | A1Cresult_None |
| 0.0014 ± 0.0021 | medical_specialty_Family/GeneralPractice |
| 0.0013 ± 0.0010 | diag_2_427 |
| 0.0013 ± 0.0011 | diag_2_276 |
| 0.0011 ± 0.0022 | age_[50-60) |
| 0.0010 ± 0.0022 | age_[80-90) |
| 0.0007 ± 0.0006 | repaglinide_No |
| 0.0006 ± 0.0010 | diag_1_428 |
| 0.0006 ± 0.0022 | payer_code_SP |
| 0.0005 ± 0.0030 | insulin_No |
| 0.0004 ± 0.0028 | diabetesMed_Yes |
| 0.0004 ± 0.0021 | diag_3_250 |
| 0.0003 ± 0.0018 | diag_2_250 |
| 0.0003 ± 0.0015 | glipizide_No |
| … 44 more … | |

Здесь представлены признаки, которые сильнее всего влияют на результат.

Рассмотрим признак number_inpatient более подробно:

```
# PDP for number_inpatient feature

from matplotlib import pyplot as plt
from pdpbox import pdp, get_dataset, info_plots
```

```
feature_name = 'number_inpatient'
# Create the data that we will plot
my_pdp = pdp.pdp_isolate(model=my_model,
                         dataset=val_X,model_features=val_X.columns,feature=feature_name)

# plot it
pdp.pdp_plot(my_pdp, feature_name)
plt.show()
```




Этот график показывает влияние признака number_inpatient на результат.

pdpbox – еще один пакет в sklearn.

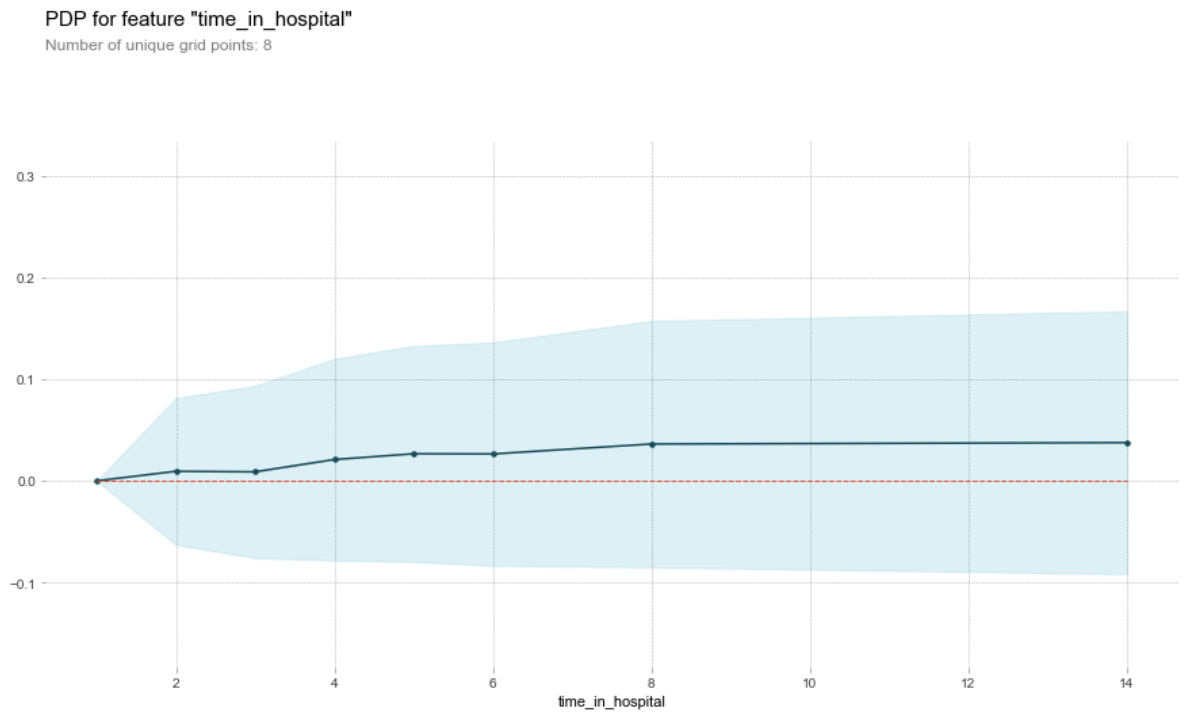
pdp = partial dependance plot.

Этот пакет показывает частичную зависимость таргета от входных данных.

На графике по оси *y* – таргетные переменные, вероятности определенного класса. Видим, что в масштабах [0, 1] признак number_inpatient может существенно повлиять на результат.

Для сравнения рассмотрим другой признак time_in_hospital – время, проведенное в больнице:

```
# The results are very different. Specifically time in hospital has a much smaller effect. Code below:

from matplotlib import pyplot as plt
from pdpbox import pdp, get_dataset, info_plots

feature_name = 'time_in_hospital'
# Create the data that we will plot
my_pdp = pdp.pdp_isolate(model=my_model,
                         dataset=val_X,
                         model_features=val_X.columns,
                         feature=feature_name)

# plot it
pdp.pdp_plot(my_pdp, feature_name)
plt.show()
```



Видим, что time_in_hospital практически не влияет на результат:

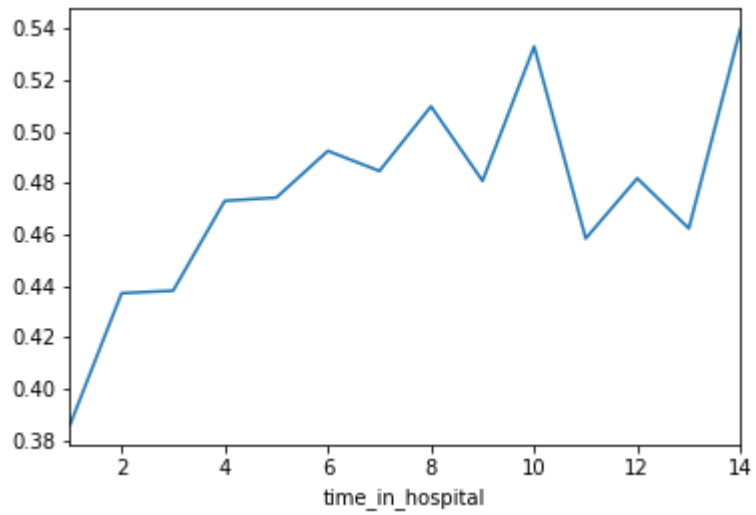
```
all_train.groupby([ 'time_in_hospital']).mean().readmitted
```

```
time_in_hospital
1.0     0.385870
2.0     0.437074
3.0     0.438037
4.0     0.472978
5.0     0.474232
6.0     0.492397
7.0     0.484594
8.0     0.509728
9.0     0.480734
10.0    0.533030
11.0    0.458333
12.0    0.481752
13.0    0.462222
14.0    0.539906
Name: readmitted, dtype: float64
```

В левой колонке представлено время в часах. Получаем, что за 1 час, проведенный в больнице, вероятность вернуться повторно составляет 38%.

Построим график зависимости признака time_in_hospital от таргета readmitted:

```
# A simple pandas groupby showing the average readmission rate for each time_in_hospital.

# Do concat to keep validation data separate, rather than using all original data
all_train = pd.concat([train_X, train_y], axis=1)

all_train.groupby(['time_in_hospital']).mean().readmitted.plot()
plt.show()
```



Попытаемся понять, что происходит в этой функции, и какие факторы будут влиять на вероятность того, что пациент вернется:

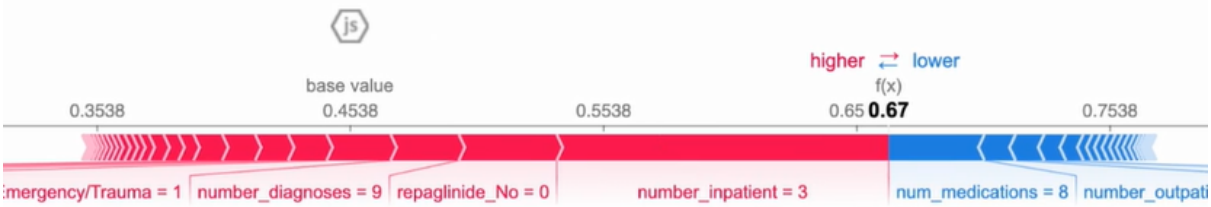
```
# Use SHAP values to show the effect of each feature of a given patient

import SHAP  # package used to calculate SHAP values

sample_data_for_prediction = val_X.iloc[0].astype(float)  # to test function

def patient_risk_factors(model, patient_data):
    # Create object that can calculate SHAP values
    explainer = SHAP.TreeExplainer(model)
    SHAP_values = explainer.SHAP_values(patient_data)
    SHAP.initjs()
    return SHAP.force_plot(explainer.expected_value[1], SHAP_values[1], patient_data)
```

Здесь с помощью библиотеки SHAP мы создаем еще одну обертку для случайного леса:

```
patient_risk_factors(my_model, val_X.iloc[4].astype(float))
```

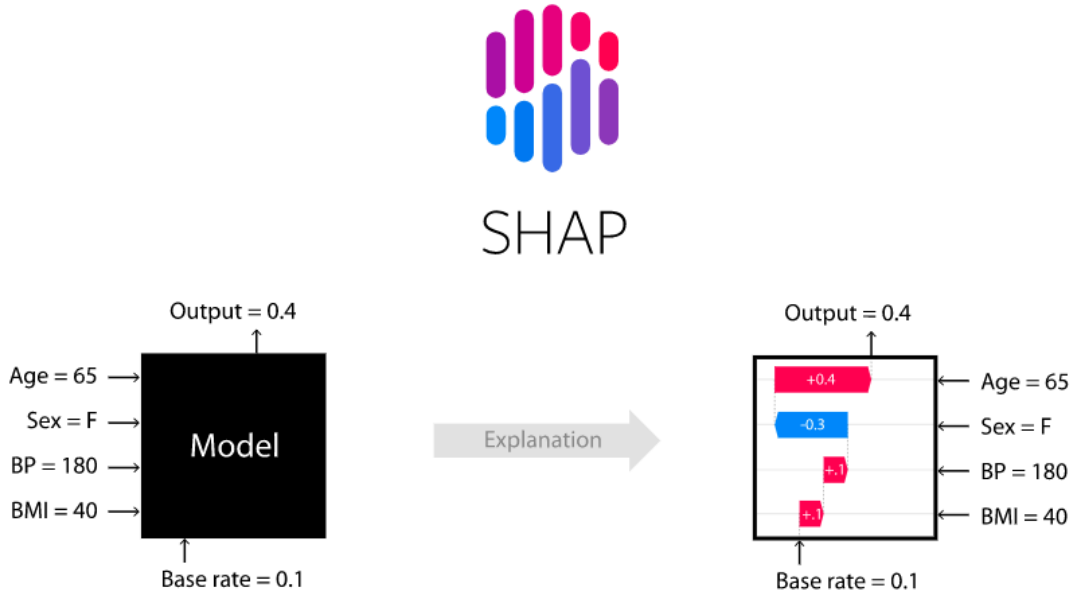
>>>


Признаки в красной области для 23 пациента увеличивают риск того, что пациент вернется в больницу.




В данном случае для 23 пациента больше факторов влияют на то, что он вернется в больницу, чем не вернется.

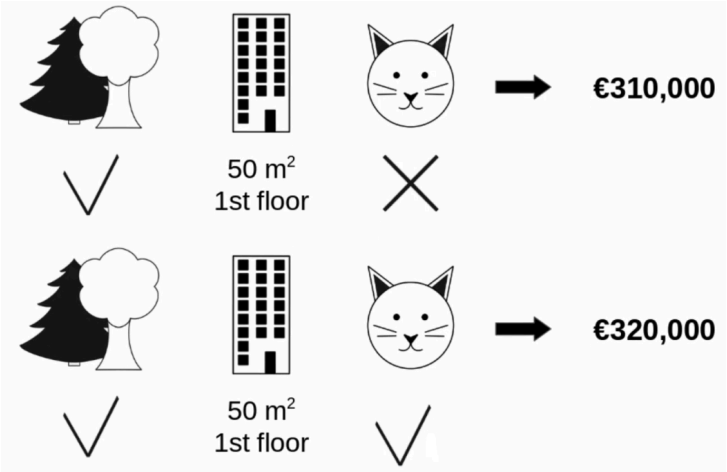
Библиотека SHAP помогает нам заглянуть внутрь модели машинного обучения.




SHAP работает с параметрами SHAPley values.

Рассмотрим пример продажи недвижимости. Есть датасет с перечнем квартир и признаками: park-nearby (близость парка), cat-banned (нельзя содержать домашних животных), area-50 (площадь), floor-2nd (этаж). Также представлены цены на квартиры. Нужно понять, какие признаки сильнее влияют на стоимость.

Построим различные коалиции из представленных признаков. Будем включать признак cat-banned и исключать из коалиций, чтобы посмотреть влияние этого признака на результат.



Можно составлять разные коалиции:

- `No feature values`
- `park-nearby`
- `area-50`
- `floor-2nd`
- `park-nearby + area-50`
- `park-nearby + floor-2nd`
- `area-50 + floor-2nd`
- `park-nearby + area-50 + floor-2nd` .

Для каждой такой коалиции можно усреднять значения цены квартиры с разрешенными кошками и запрещенными. Это среднее значение и есть число SHAPley.

Вернемся к нашему датасету медицинских данных.

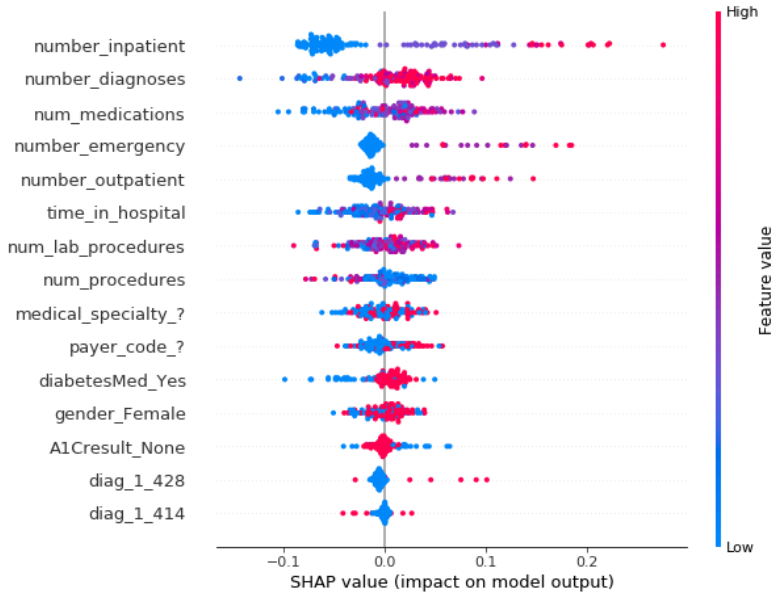
Решим задачу на число SHAPley. Выберем подмножество признаков из датасета, подмножество пациентов. Будем использовать модель случайного леса:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import SHAP

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

data = pd.read_csv('data/medical.csv')
y = data.readmitted
base_features = ['number_inpatient', 'num_medications', 'number_diagnoses',
'num_lab_procedures',  'num_procedures', 'time_in_hospital', 'number_outpatient',
'number_emergency', 'gender_Female', 'payer_code_?', 'medical_specialty_?', 'diag_1_428',
'diag_1_414', 'diabetesMed_Yes', 'A1Cresult_None']

# Some versions of SHAP package error when mixing bools and numerics
X = data[base_features].astype(float)

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# For speed, we will calculate SHAP values on smaller subset of the validation data
```

```
small_val_X = val_X.iloc[:150]
my_model = RandomForestClassifier(n_estimators=30, random_state=1).fit(train_X,
train_y)
```

Рассмотрим, какие признаки из нашего подмножества влияют на таргет:

```
explainer = SHAP.TreeExplainer(my_model)
SHAP_values = explainer.SHAP_values(small_val_X)

SHAP.summary_plot(SHAP_values[1], small_val_X)
```




Попробуем понять, какой из двух признаков diag_1_428 или payer_code_? влияет больше на разницу между максимальным и минимальным числами SHAPley.

Для diag_1_428:

```
SHAP_values[0][:, -2].max() - SHAP_values[0][:, -2].min()
```

>>> 0.14825

Для каждого значения признака (нулевой класс или первый класс) строятся две матрицы SHAP_values, которые имеют размер:

```
SHAP_values[0].SHAPe
```

>>> (150, 15)

Точно так же мы можем посмотреть разницу для payer_code_?:

```
SHAP_values[0][:, -6].max() - SHAP_values[0][:, -6].min()
```

>>>0.103954

Если мы не помним, что принимается на вход функции и что возвращается, мы можем написать:

```
explainer.SHAP_values??
```

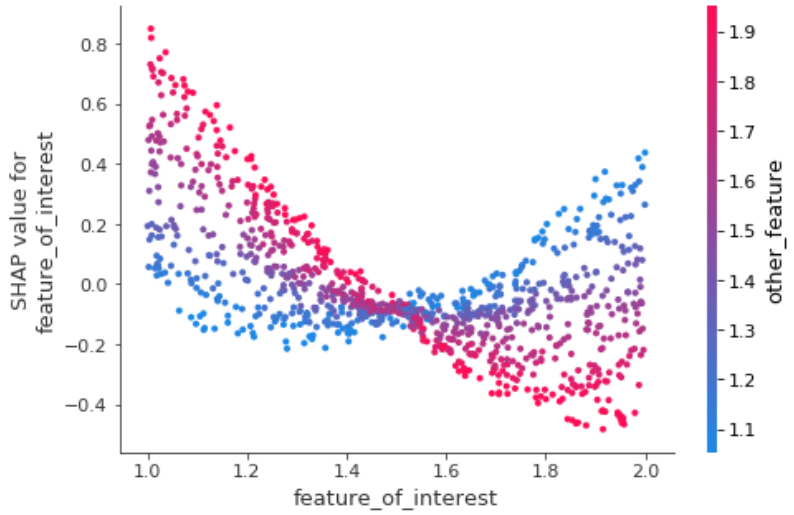
Получим тело функции со всеми подробными комментариями.

Насколько легитимна полученная метрика? Нужно сравнить признаки range of effect size и permutation importance. Range of effect size's очень чувствителен к выбросам, permutation importance более стабилен.

Оба признака diag_1_428 и payer_code_? бинарные. Что будет, если для этих признаков поменять местами 0 и 1? У признака diag_1_428 есть выраженное разделение, для payer_code_? точки перемешаны. При замене 0 на 1 для diag_1_428 получится, что здоровые люди будут с большей вероятностью ходить в больницу, чем больные. Для payer_code_? при замене 0 на 1 ничего существенно не изменится, поскольку два класса для этого признака практически неотделимы друг от друга.

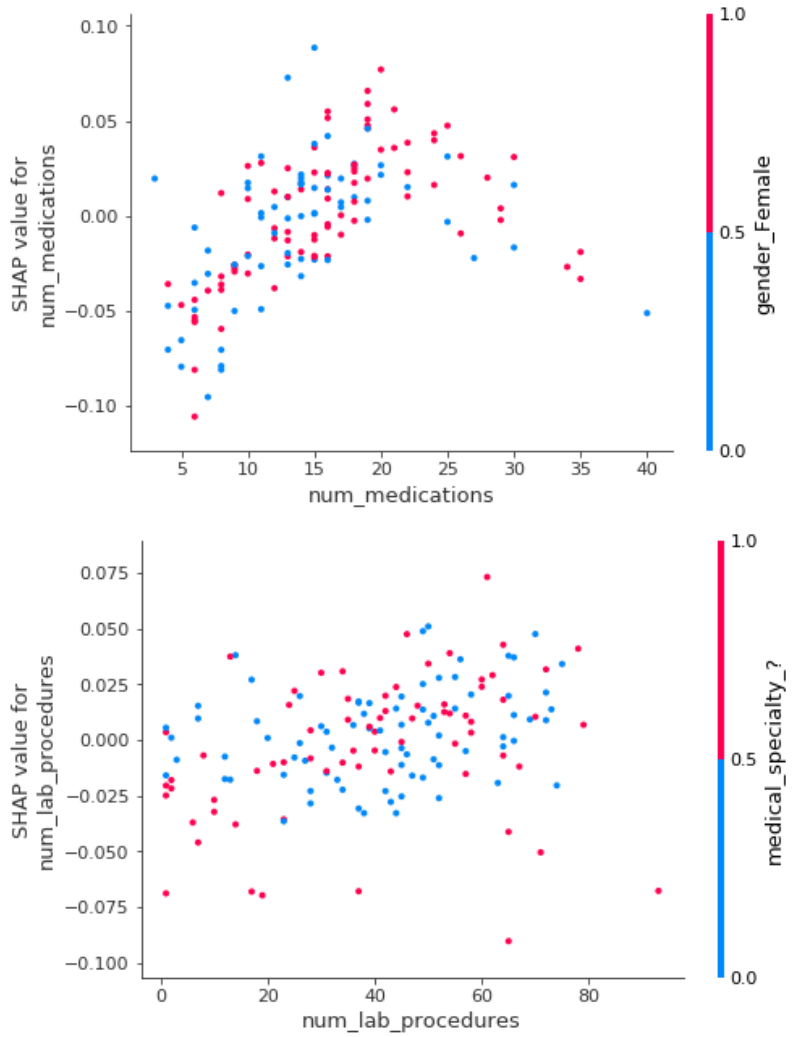
Теперь обратим внимание на признак num_lab_procedures на графике. В этом признаке также все точки перемешаны и неразделимы. Скорее всего, этот признак скоррелирован с каким-то другим, либо от него зависит.

Посмотрим на то, как взаимодействуют друг с другом признаки.



Здесь цветом обозначена степень влияния других признаков на выбранный.
Построим графики зависимостей признаков друг от друга:

```
SHAP.dependence_plot('num_medications' ,SHAP_values[1], small_val_X)
SHAP.dependence_plot('num_lab_procedures' ,SHAP_values[1], small_val_X)
```




Из графиков видим, что для num_lab_procedures разделить точки невозможно.

## Дополнительные материалы для самостоятельного изучения

1. Анализ метаданных. КвазиНаучный блог Александра Дьяконова