Лабораторная работа «Ансамбль ДР»

Цель работы: получить практические навыки работы с ансамблями деревьев решений на практических примерах с использованием языка программирования python.

1. Теоретический материал

Ансамбль — это некая совокупность моделей, части которой образуют единое целое.

Например:

Теорема Кондорсе «о жюри присяжных» (1784).

Если каждый член жюри присяжных имеет независимое мнение, и, если вероятность правильного решения члена жюри больше 0.5, то тогда вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри и стремится к единице. Если же вероятность быть правым у каждого из членов жюри меньше 0.5, то вероятность принятия правильного решения присяжными в целом монотонно уменьшается и стремится к нулю с увеличением количества присяжных.

N — количество присяжных

p — вероятность правильного решения присяжного

μ — вероятность правильного решения всего жюри

$$\mu = \sum_{i=m}^{N} C_N^i p^i (1-p)^{N-i}$$

Если $p > 0.5$, то $\mu > p$

Если $N \to \infty$, то $\mu \to 1$

"Мудрость толпы".

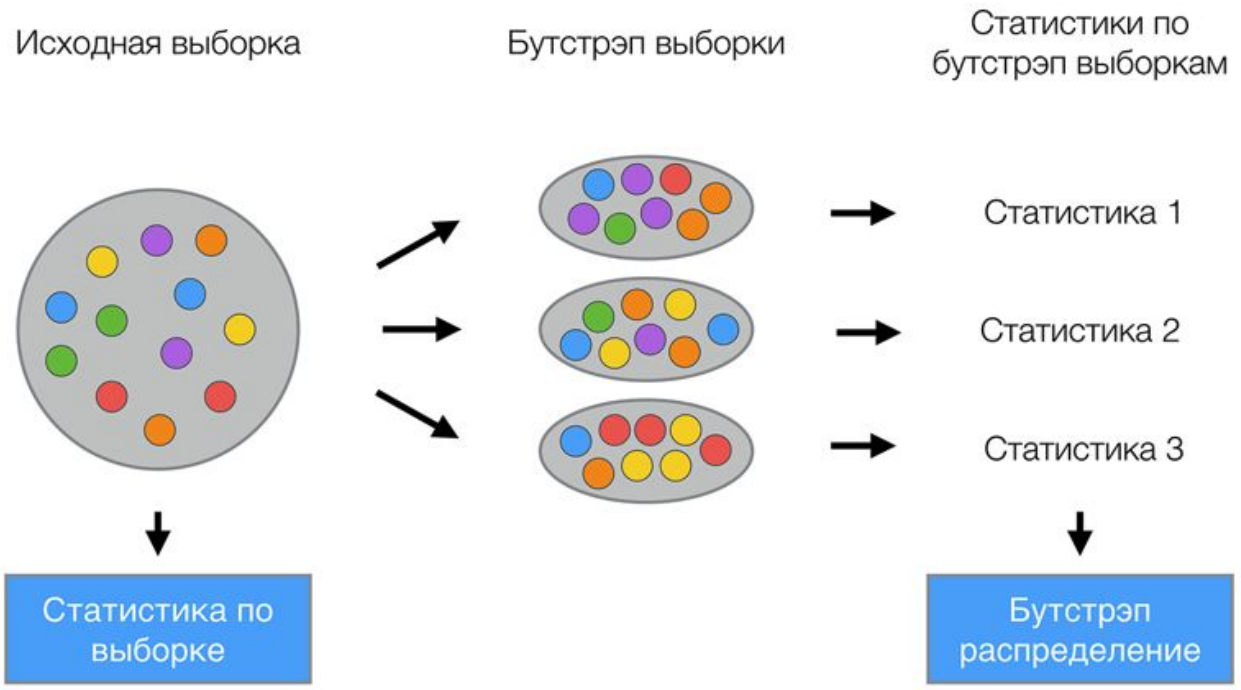
Фрэнсис Гальтон в 1906 году посетил рынок, где проводилась некая лотерея для крестьян. Их собралось около 800 человек, и они пытались угадать вес быка, который стоял перед ними. Бык весил 1198 фунтов. Ни один крестьянин не угадал точный вес быка, но если посчитать среднее от их предсказаний, то получим 1197 фунтов.

Такая идея уменьшения ошибки применяется и в машинном обучении.

2. Бутстрэп

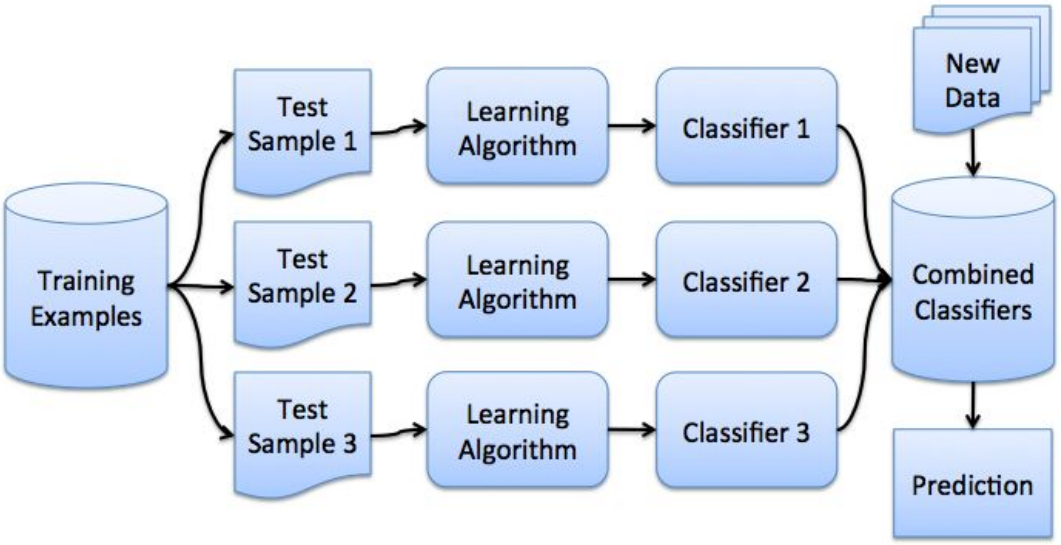
Bagging (от Bootstrap aggregation) — это один из первых и самых простых видов ансамблей. Бэггинг основан на статистическом методе ***бутстрэпа***, который позволяет оценивать многие статистики сложных распределений.

Метод бутстрэпа заключается в следующем. Пусть имеется выборка X размера N. Равномерно возьмем из выборки N объектов с возвращением. Это означает, что мы будем N раз выбирать произвольный объект выборки (считаем, что каждый объект «достается» с одинаковой вероятностью $\frac{1}{N}$), причем каждый раз мы выбираем из всех исходных N объектов. Можно представить себе мешок, из которого достают шарики: выбранный на каком-то шаге шарик возвращается обратно в мешок, и следующий выбор опять делается равновероятно из того же числа шариков. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через X_1. Повторяя процедуру M раз, сгенерируем M подвыборок $X_1, \ldots, X_M$ Теперь мы имеем достаточно большое число выборок и можем оценивать различные статистики исходного распределения.



3. Бэггинг

Пусть имеется обучающая выборка X. С помощью бутстрэпа сгенерируем из неё выборки X_1, ... , X_M. Теперь на каждой выборке обучим свой классификатор $a_i(x)$. Итоговый классификатор будет усреднять ответы всех этих алгоритмов (в случае классификации это соответствует голосованию): $a_i(x) = \frac{1}{M} \sum_{i=1}^{M} a_i(x)$. Эту схему можно представить картинкой ниже.



Бэггинг позволяет снизить дисперсию (*variance*) обучаемого классификатора, уменьшая величину, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных, или другими словами, предотвращает переобучение. Эффективность бэггинга достигается благодаря тому, что базовые алгоритмы, обученные по различным подвыборкам, получаются достаточно различными, и их ошибки взаимно компенсируются при голосовании, а также за счёт того, что объекты-выбросы могут не попадать в некоторые обучающие подвыборки.

4. Случайный лес

Случайный лес (*random forest, RF*) — это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству.

4.1. Схема построения

Все деревья строятся независимо по следующей схеме:

1. Выбирается подвыборка обучающей выборки размера *samplesize* (м.б. с возвращением) – по ней строится дерево (для каждого дерева — своя подвыборка).

2. Для построения каждого расщепления в дереве просматриваем *max_features* случайных признаков (для каждого нового расщепления — свои случайные признаки).
3. Выбираем наилучший признак и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

Такая схема построения соответствует главному принципу *ансамблирования* (построению алгоритма машинного обучения на базе нескольких, в данном случае решающих деревьев): базовые алгоритмы должны быть хорошими и разнообразными (поэтому каждое дерево строится на своей обучающей выборке и при выборе расщеплений есть элемент случайности).

4.2. Недостатки случайного леса

Случайный лес — композиция глубоких деревьев, которые строятся независимо друг от друга. Но такой подход имеет следующую проблему. Обучение глубоких деревьев требует очень много вычислительных ресурсов, особенно в случае большой выборки или большого числа признаков.

Если ограничить глубину решающих деревьев в случайном лесе, то они уже не смогут улавливать сложные закономерности в данных. Это приведет к тому, что сдвиг будет слишком большим.

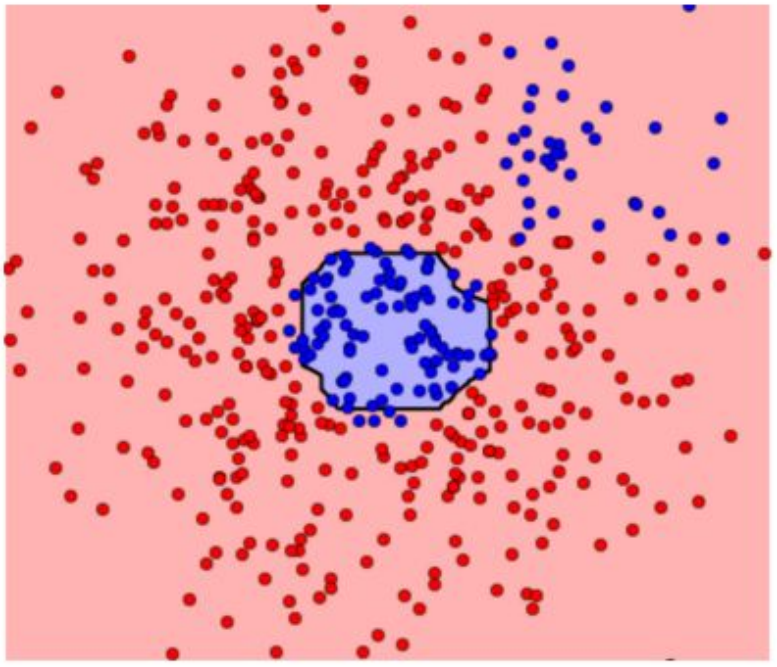


Рис. 3: Неглубокие деревья не способны улавливать все закономерности в данных. В данном случае синий класс состоит из двух групп объектов, но неглубокое дерево смогло уловить только центральную группу. На объектах из второй группы такое дерево ошибается.

Вторая проблема со случайным лесом состоит в том, что процесс построения деревьев является ненаправленным: каждое следующее дерево в композиции никак не зависит от предыдущих. Из-за этого для решения сложных задач необходимо огромное количество деревьев.

Решить данные проблемы можно с помощью так называемого бустинга.

5. Градиентный бустинг

5.1. Основная идея

Бустинг — это подход к построению композиций, в рамках которого:

- Базовые алгоритмы строятся последовательно, один за другим.
- Каждый следующий алгоритм строится таким образом, чтобы исправлять ошибки уже построенной композиции.

Благодаря тому, что построение композиций в бустинге является направленным, достаточно использовать простые базовые алгоритмы, например, неглубокие деревья.

5.2. Градиентный бустинг

Градиентный бустинг является одним из лучших способов направленного построения композиции на сегодняшний день. В градиентном бустинге строящаяся композиция

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

является суммой, а не их усреднением базовых алгоритмов $b_i(x)$. Это связано с тем, что алгоритмы обучаются последовательно и каждый следующий корректирует ошибки предыдущих.

Пусть задана функция потерь *L(y, z)*, где *y* — истинный ответ, *z* — прогноз алгоритма на некотором объекте. Примерами возможных функций потерь являются:

среднеквадратичная ошибка (в задаче регрессии):

$$L(y, z) = (y - z)^2$$

логистическая функция потерь (в задаче классификации):

$$L(y, z) = log(1 - exp\,(-yz))$$

5.3. Инициализация

В начале построения композиции по методу градиентного бустинга нужно ее инициализировать, то есть построить первый базовый алгоритм $b_0(x)$. Этот алгоритм не должен быть сколько-нибудь сложным и не стоит тратить на него много усилий. Например, можно использовать:

• алгоритм $b_0(x) = 0$, который всегда возвращает ноль (в задаче регрессии);

• более сложный $b_0(x) = \frac{1}{l}\sum_{i=1}^{l} y_i$, который возвращает средний по всем элементам обучающей выборки истинный ответ (в задаче регрессии);

• алгоритм $b_0(x) = argmax_{y \in Y} \frac{1}{l}\sum_{i=1}^{l} [y_i = y]$, который всегда возвращает метку самого распространенного класса в обучающей выборке (в задаче классификации).

5.4. Обучение базовых алгоритмов

Обучение базовых алгоритмов происходит последовательно. Пусть к некоторому моменту обучены N − 1 алгоритмов $b_1(x), \ldots, b_{N-1}(x)$, то есть композиция имеет вид:

$$a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x)$$

Теперь к текущей композиции добавляется еще один алгоритм $b_N(x)$. Этот алгоритм обучается так, чтобы как можно сильнее уменьшить ошибку композиции на обучающей выборке:

$$\sum_{i=1}^{l} L(y_i,\ a_{N-1}(x_i) + b(x_i)) \rightarrow min_b$$

Сначала имеет смысл решить более простую задачу: определить, какие значения $s_1, \ldots, s_l$ должен принимать алгоритм $b_N(x_i) = s_i$ на объектах обучающей выборки, чтобы ошибка на обучающей выборке была минимальной:

$$F(s) = \sum_{i=1}^{l} L(y_i,\ a_{N-1}(x_i) + s_i) \rightarrow min_s$$

где $s = (s_1, \ldots, s_l)$ — вектор сдвигов.

Другими словами, необходимо найти такой вектор сдвигов s, который будет минимизировать функцию $F(s)$. Поскольку направление наискорейшего убывания функции задается направлением антиградиента, его можно принять в качестве вектора s:

$$s = -\nabla F = (-L_z'(y_1, a_{N-1}(x_1)), \cdots -L_z'(y_l, a_{N-1}(x_l)))$$

Компоненты вектора сдвигов s, фактически, являются теми значениями, которые на объектах обучающей выборки должен принимать новый алгоритм $b_N(x)$, чтобы минимизировать ошибку строящейся композиции.

Обучение $b_N(x)$, таким образом, представляет собой задачу обучения на размеченных данных, в которой $\{(x_i, s_i)\}_{i=1}^{l}$ — обучающая выборка, и используется, например, квадратичная функция ошибки:

$$b_N(x) = argmin_b \frac{1}{l} \sum_{i=1}^{l} (b(x_i) - s_i)^2$$

Следует обратить особое внимание на то, что информация об исходной функции потерь $L(y, z)$, которая не обязательно является квадратичной, содержится в выражении для вектора оптимального сдвига s. Поэтому для большинства задач при обучении $b_N(x)$ можно использовать квадратичную функцию потерь.

5.5. Описание алгоритма градиентного бустинга

1. **Инициализация:** инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. Шаг итерации:
    a) **Вычисляется вектор сдвига**

$$s = -\nabla F = (-L_z'(y_1, a_{N-1}(x_1)), \cdots -L_z'(y_l, a_{N-1}(x_l)))$$

b) **Строится алгоритм**

$$b_N(x) = argmin_b \frac{1}{l} \sum_{i=1}^{l} (b(x_i) - s_i)^2$$

параметры которого подбираются таким образом, что его значения на элементах обучающей выборки были как можно ближе к вычисленному вектору оптимального сдвига s.

c) **Алгоритм** $b_n(x)$ **добавляется в композицию**

$$a_n(x) = \sum_{m=1}^{n} b_m(x)$$

Если не выполнен критерий останова (об этом будет рассказано далее), то выполнить еще один шаг итерации. Если критерий останова выполнен, остановить итерационный процесс.

5.6. Градиентный бустинг для классификации

В задаче бинарной классификации ($Y = \{-1, +1\}$) популярным выбором для функции потерь является логистическая функция потерь:

$$\sum_{i=1}^{n} log(1 - exp(-y_i a(x_i)))$$

где $a(x) \in R$ — оценка принадлежности положительному классу. Если $a(x) > 0$, классификатор относит объект x к классу +1, а при $a(x) \leq 0$ – к классу −1. Причем, чем больше $|a(x)|$, тем больше классификатор уверен в своем выборе. Функция потерь в этом случае записывается следующим образом:

$$L(y, z) = log(1 + exp(-yz))$$

$$L_z'(y, z) = -\frac{y}{1 + exp(yz)}$$

Вектор сдвигов s в этом случае будет иметь вид:

$$s = (-\frac{y_1}{1+exp(y_1 a_{N-1}(x_1))}, \cdots -\frac{y_l}{1+exp(y_l a_{N-1}(x_l))})$$

Новый базовый алгоритм будет настраиваться таким образом, чтобы вектор его ответов на объектах обучающей выборки был как можно ближе к s. После того, как вычислен алгоритм $a_N(x)$, можно оценить вероятности принадлежности объекта x к каждому из классов:

$$P(x) = \frac{1}{1+exp(-a_N(x))}$$

$$P(x) = \frac{1}{1+exp(a_N(x))}$$

6. Случайный лес и градиентный бустинг в Scikit-Learn

Метод случайного леса реализован в библиотеке машинного обучения scikit-learn двумя классами ***RandomForestClassifier***:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True,
oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
class_weight=None)
```

Для градиентного бустинга имеется такая же реализация ***GradientBoostingClassifier:***

```
class sklearn.ensemble.GradientBoostingClassifier(loss='deviance',
learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_depth=3, min_impurity_decrease=0.0, min_impurity_split=None, init=None,
random_state=None, max_features=None, verbose=0, max_leaf_nodes=None,
warm_start=False, presort='auto')
```

6.1. Важные параметры:

n_estimators — число деревьев в "лесу" (по умолчанию – 10);

criterion — функция, которая измеряет качество разбиения ветки дерева (по дефолту — "mse" , так же можно выбрать "mae");

max_features — число признаков, по которым ищется разбиение. Можно указать конкретное число или процент признаков, либо выбрать из доступных значений: "*auto*" (все признаки), "*sqrt*", "*log2*". По дефолту стоит "*auto*".

max_depth — максимальная глубина дерева (по умолчанию глубина не ограничена);

random_state — начальное значение для генерации случайных чисел (по дефолту его нет, если хотите воспроизводимые результаты, то нужно указать любое число типа *int*;

min_samples_split — минимальное количество объектов, необходимое для разделения внутреннего узла. Можно задать числом или процентом от общего числа объектов (по дефолту — 2);

min_samples_leaf — минимальное число объектов в листе. Можно задать числом или процентом от общего числа объектов (по дефолту — 1);

criterion — поскольку у нас теперь задача классификации, то по дефолту выбран критерий "*gini*" (можно выбрать "*entropy*").

6.2. Примеры

RandomForestClassifier

```
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
clf = RandomForestClassifier(max_depth=2, random_state=0)
scores = cross_val_score(grd, X, y, cv=5, scoring="accuracy")
print(clf.feature_importances_)
print(clf.predict([[0, 0, 0, 0]]))
```

GradientBoostingClassifier

```
n_estimator = 10
X, y = make_classification(n_samples=80000)
grd = GradientBoostingClassifier(n_estimators=n_estimator)
scores = cross_val_score(grd, X, y, cv=5, scoring="accuracy")
print(clf.feature_importances_)
```

feature_importances_

```
X, y = make_classification(n_samples=1000, n_features=10, n_informative=3,
                           n_redundant=0, n_repeated=0, n_classes=2,
                           random_state=0, shuffle=False)

# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=250, random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
axis=0)
indices = np.argsort(importances)[::-1]

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices], color="r",
yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

7. Ход работы:

1. Прочитать теоретическую часть об ансамблях деревьев решений.
2. Изучить ***RandomForestClassifier*** из библиотеки scikit-learn.

3. Изучить ***GradientBoostingClassifier*** из библиотеки scikit-learn.
4. В качестве исходного массива данных используем предобработанные данные из 1-ой лабораторной работы. Для кросс-валидации используем разбиение на 5 фолдов (функция *cross_val_score*) с метрикой качества "*accuracy*".
5. Для своего варианта провести серию экспериментов по настройке параметров случайного леса:
    a) Зафиксировать размер леса (*n_estimators* = 50) и *random_state*;
    b) Варьируем параметр *max_features* от 2 до максимального количества признаков в выборке;
    c) Варьируем *min_sample_leaf* и *min_sample_split* (*min_sample_leaf* < min_*sample_split*);
    d) Варьируем *max_depth* (2-20);
    e) Варьируем n_estimators.
6. Для своего варианта провести серию экспериментов по настройке параметров градиентного бустинга:
    a) Зафиксировать размер леса (*n_estimators* = 50) и *random_state*;
    b) Варьируем параметр *max_features* от 2 до максимального количества признаков в выборке;
    c) Варьируем *min_sample_leaf* и *min_sample_split* (*min_sample_leaf* < min_*sample_split*);
    d) Варьируем *max_depth* (2-20);
    e) Варьируем n_estimators.
7. Для построенных моделей получить важность признаков (*feature_importances*), построить диаграммы и проанализировать.

8. Сравнить результаты работы трех алгоритмов: деревья решений, случайный лес и градиентный бустинг и сделать выводы.

Вопросы на защиту:

1. Что такое бэгинг?
2. Идея бутстрап?
3. Что такое случайный лес?
4. Постановка задачи градиентного бустинга.
5. Алгоритм градиентного бустинга.

Список литературы:

1. Случайный лес (Random Forest)

https://alexanderdyakonov.wordpress.com/2016/11/14/случайный-лес-random-forest/

2. Градиентный бустинг

https://alexanderdyakonov.files.wordpress.com/2017/06/book_boosting_pdf.pdf

3. Композиции: бэггинг, случайный лес

https://habrahabr.ru/company/ods/blog/324402/#ansambli

4. Градиентный бустинг (en)

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/