Ансамбли

Ансамбль народной музыки «Солнцеворот»

Объединение нескольких моделей – ансамбль решений

- Объединение прогнозов нескольких классификаторов
- Передаем список пар (название, классификатор)

  estimators=[('kNN', knn), ('GaussNB', gauss), ('Decision Tree', tree)]
- voting – тип голосования. hard – результаты от функции predict, soft – от predict_proba
- Дополнительно можно указать с какими весам брать ответы классификаторов weights=[2,1,2]
- Точно так же как для одного классификатора вызываем метод fit, predict

lection_5_VotingClassifier

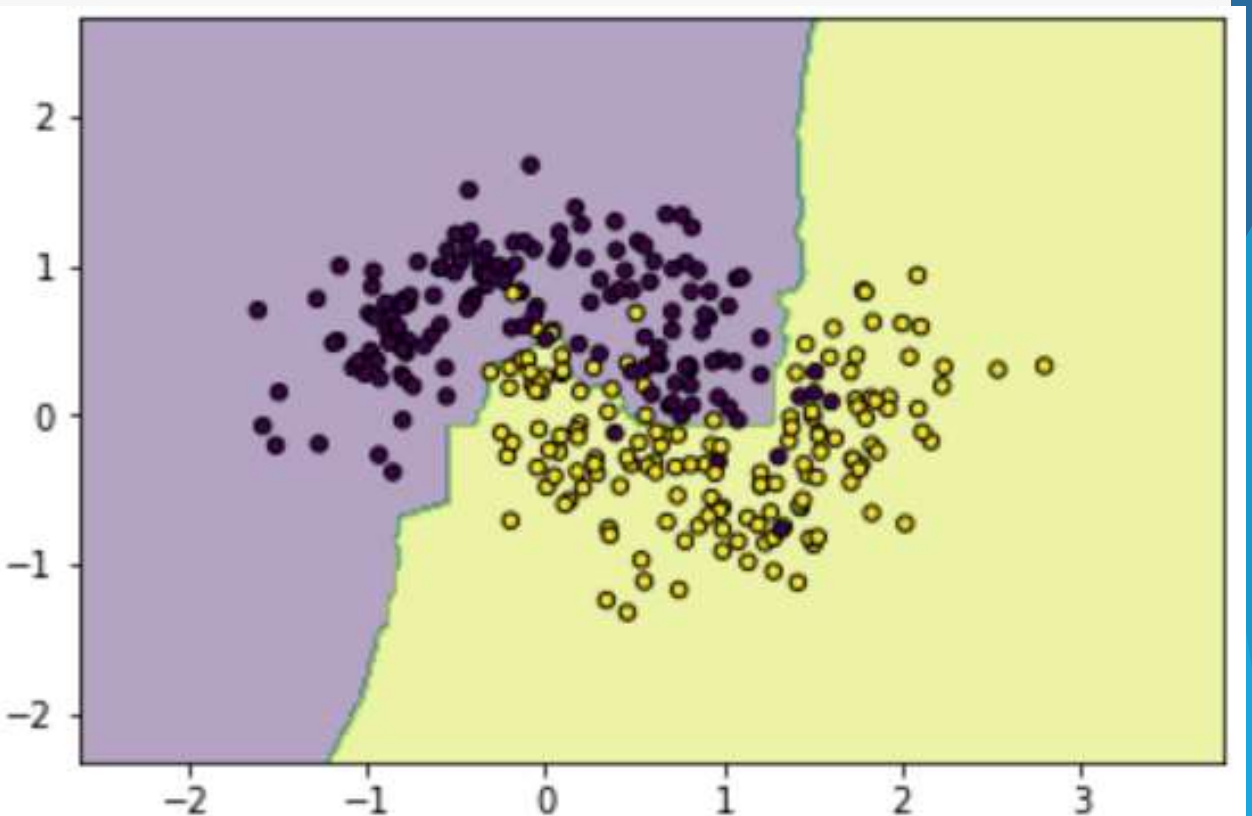
Ансамбль решений

```
1  X,y = make_moons(n_samples=300, noise=0.3, random_state=0)
```

- kNN = 0.907, GaussianNB = 0.853, DecisionTreeClassifier(max_depth=2) = 0.88

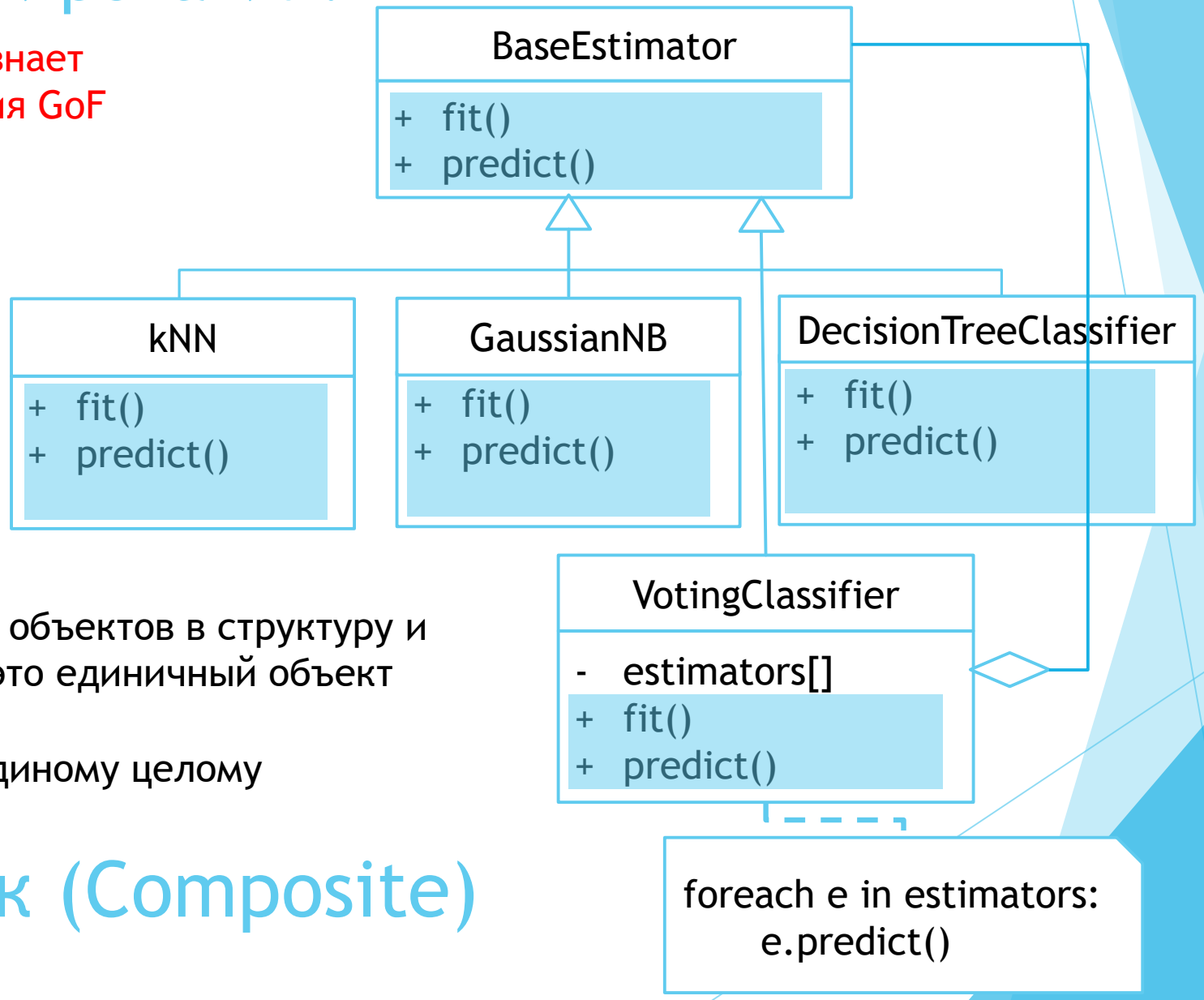
```
1  from sklearn.ensemble import VotingClassifier
2  ensemble = VotingClassifier(
3      estimators=[('kNN', knn), ('GaussNB', gauss), ('Decision Tree', tree)],
4      voting='soft', weights=[2,1,2])
5  ensemble.fit(X_train,y_train)
6  yperc = ensemble.predict(X_test)
7  accuracy_score(yperc, y_test)
```

- soft – по суммам вероятностей
- hard – по меткам классов
- Точность 0.927



Ансамбль – аналог какого структурного шаблона проектирования?

Вопрос только для тех, кто знает
про шаблоны проектирования GoF



Группировка множества объектов в структуру и работа с ней как будто это единичный объект

Задача – доступ как к единому целому

Компоновщик (Composite)

Bagging = Bootstrap aggregating
Случайный лес

- Bootstrap – выбор подмножеств исходных данных (с возвратом)
- Выращивание дерева решений на каждом таком подмножестве
- Ансамблевый метод – объединяет деревья решений, метка на основе большинства голосов
- Пример реализации – класс RandomForestClassifier
- Плюсы
  - Нет необходимости предварительно масштабировать данные
  - Можно увидеть, на какие признаки обращается больше внимания **feature_importances_**
  - Высокая степень распараллеливания

Bagging на любом алгоритме. Базовый класс – BaggingClassifier, estimator – выбранный алгоритм (параметр конструктора)

Feature_importances_

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=10)
rf.fit(X_train, y_train)
pred_test = rf.predict(X_test)
```

```
nn = p.DataFrame({'features': names, 'importances': rf.feature_importances_})
nn.sort_values(by=['importances'], ascending=False)
```

features	importances
Интенсивность цвета	0.220065
Пролин	0.208846
Оттенок	0.101640
Алкоголь	0.099604
Флаваноиды	0.099276
OD280 / OD315 разбавленных (разведенных) вин	0.057870
Яблочная кислота	0.056285
Щелочность золы	0.049187

Random forest

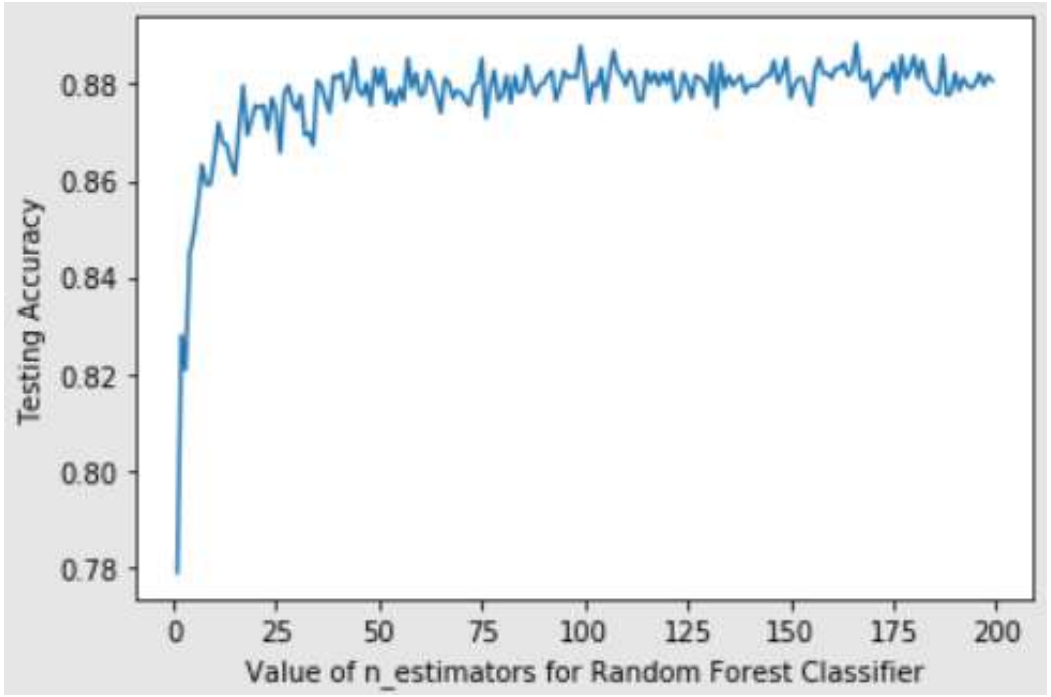
- С ростом количества деревьев в композиции склонность к переобучению уменьшается – композиция более устойчива к шуму
- Одинаково хорошо работает как с непрерывными так и с дискретными признаками

n_jobs=-1

Улучшение качества, настройка гиперпараметров:

- **n_estimators** – для поиска баланса между количеством деревьев и временем работы (найти минимальное количество, для которого качество вышло на асимптоту)
- **max_features** – количество признаков для разбиения - так называемый feature bagging. Либо явно int (обычно n/3 для задач регрессии), либо sqrt (обычно для задач классификации), либо None – все признаки
- **max_depth** – максимальная глубина дерева – уменьшение переробучения отдельно взятого дерева
- **min_samples_split** – минимальное число экземпляров, при котором будет происходить расщепление узла
- **min_samples_leaf** – минимальное число экземпляров, которые могут находиться в листовом узле

https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf



Зависимость точности от количества деревьев

Boosting – усиление слабых учеников

- Состоит из очень простых базовых классификаторов, например, одноуровневое дерево. Как бы вы думали оно назывется? ☺ Решающий пенёк
- Извлекается случайное подмножество данных (без возврата), на которых учится первый классификатор C1
- Извлекается второе случайное подмножество и к нему добавляется 50% ранее ошибочно классифицированных образцов (с увеличением весов). Обучается классификатор C2
- Извлекается третье случайное подмножество и к нему добавляются образцы, по которым C1 и C2 расходятся (с увеличением весов)
- Объединяются решения классификаторов. Метка на основе большинства голосов.
- Плюсы
  - Хорошая обучающая способность
  - Возможность идентифицировать шумовые выбросы (но не сильные)
- Минусы – тяжело распараллелить, так как последовательное обучение

AdaBoostClassifier (Adaptive Boosting) – использует полную выборку (без подмножеств), estimator – выбранный алгоритм (параметр конструктора), по умолчанию – дерево решений

Gradient Boosting (Machine) GBM

- Идея аналогичная – каждый последующий классификатор пытается улучшить прогнозы
- Вводится функция потерь (например, logloss), а новый оценщик пытается подогнаться к остаточным ошибкам предыдущего

GradientBoostingClassifier – на основе DecisionTreeRegresor, деревья подгоняются к отрицательному градиенту функции потерь

XGBoost (Extreme Gradient Boosting)– одна из самых популярных и эффективных реализаций

CatBoost (Category Boost) – библиотека на основе деревьев решений от компании Яндекс, показывающая хорошие результаты даже без настройки гиперпараметров

https://xgboost.ai/

https://neerc.ifmo.ru/wiki/index.php?title=XGBoost

https://catboost.ai/

Gradient Boosting (Machine) GBM

- В правиле обучения персептрона Видроу-Хоффа вводили функцию издержек

$$S(\mathbf{w}) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)^2$$

- Градиентным спуском делали шаги в сторону минимума функции потерь

$$w := w + \Delta w \qquad \Delta \mathbf{w} = -\eta \nabla S(\mathbf{w})$$

- Движемся итеративно в сторону вектора правильных меток y, на каждом шаге добавляя с каким-то весом новый классификатор, который улучшает прогноз

$$F_m(X) = F_{m-1}(X) + \eta\Delta_m(X) \qquad \hat{y}_m = F_m(X) \qquad S(y,\hat{y}) = (y-\hat{y})^2$$

$$\hat{y}_m = \hat{y}_{m-1} + \eta\Delta_m(X) \qquad S(y,\hat{y}) = \log\left(1+\exp(-y\hat{y})\right)$$

- Хотим минимизировать функцию потерь

$$S(y,\hat{y}_m) \to \min \qquad S(y,\hat{y}_{m-1} + \eta\Delta_m(X)) \to \min$$

- Новый классификатор тренировался на разнице истиной метки и метки на шаге m-1 (обучающая выборка). Хотим минимизировать S, будем шагать по антиградиенту.

$$\hat{y}_m = \hat{y}_{m-1} + \eta(-\nabla S(y,\hat{y}_{m-1}))$$

https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

https://explained.ai/gradient-boosting/descent.html

GradientBoostingClassifier

- Сильно подгоняется под обучающую выборку. С увеличением числа деревьев будет больше переобучаться
- Одинаково хорошо работает как с непрерывными так и с дискретными признаками

Улучшение качества, настройка гиперпараметров:

- **n_estimators**, **max_features**, **max_depth**, **min_samples_split**, **min_samples_leaf** – как у Random Forest
- **learning_rate** – сокращает вклад каждого дерева (по умолчанию 1)
- **subsample -** доля образцов, которые будут использоваться для тренировки очередного оценщика (по умолчанию 1). Если меньше единицы может понизить дисперсию, но увеличить смещение.

https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regularization.html

n_jobs – нет такого параметра

Ансамбли из деревьев решений