УНИВЕРСИТЕТ ИТМО

Практика. Ансамблирование моделей

Олег А. Евстафьев[1] Михаил А. Каканов[1]
[1] Факультет систем управления и робототехники, Университет ИТМО
{oaevstafev, makakanov}@itmo.ru

Ноябрь 2021
Курс «Прикладной искусственный интеллект»

Метода для комбинирования прогнозов

УНИВЕРСИТЕТ ИТМО

- **мешковина**, Построение нескольких моделей (обычно одного типа) из разных подвыборок обучающего набора данных.
- **стимулирование**, Построение нескольких моделей (обычно одного типа), каждая из которых учится исправлять ошибки прогнозирования предыдущей модели в цепочке.
- **голосование**, Построение нескольких моделей (обычно разных типов) и простая статистика (например, вычисление среднего значения) используются для объединения прогнозов.

модель или общая идея	описание и примеры
комитеты (голосование) / усреднение	Построение независимых алгоритмов и их усреднение / голосование по ним, в том числе с помощью **бэгинга** и предварительной деформации ответов. Здесь же и обобщения бэгинга – **случайные леса**.
кодировки / перекодировки ответов	Специальные кодировки целевых значений и сведение решения задачи к решению нескольких задач. Один из самых популярных приёмов – **ECOC** (error-correcting output coding). Другой – настройка на разные деформации целевого признака.
стекинг (stacking)	построение метапризнаков – ответов базовых алгоритмов на объектах выборки, обучение на них мета- алгоритма
бустинг (boosting)	Построение суммы нескольких алгоритмов. Каждое следующее слагаемое строится с учётом ошибок предыдущих: AdaBoost, градиентный бустинг.
«ручные методы»	Эвристические способы комбинирования ответов базовых алгоритмов (с помощью визуализаций, обучения в специальных подпространствах и т.п.)
однородные ансамбли	Пример – **нейронные сети.** Формула мета-алгоритм – базовые алгоритмы разворачивается рекурсивно, применяется общая схема оптимизации полученной конструкции.

ITMOre than a UNIVERSITY

Деревья решений

УНИВЕРСИТЕТ ИТМО

```
# Bagged Decision Trees for Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.770745044429
```

ITMOre than a UNIVERSITY

Случайный лес

УНИВЕРСИТЕТ ИТМО

```
# Random Forest Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 3
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.770727956254
```

ITMO more than a UNIVERSITY

Дополнительные деревья

УНИВЕРСИТЕТ ИТМО

```
# Extra Trees Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import ExtraTreesClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.760269993165
```

ITMOre than a UNIVERSITY

AdaBoost

УНИВЕРСИТЕТ ИТМО

```
# AdaBoost Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.76045796309
```

ITMO more than a UNIVERSITY

Стохастическое повышение градиента

УНИВЕРСИТЕТ ИТМО

```
# Stochastic Gradient Boosting Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.764285714286
```

ITMOre than a UNIVERSITY

Голосующий ансамбль

УНИВЕРСИТЕТ ИТМО

```
# Voting Ensemble for Classification
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

ITMO more than a UNIVERSITY

```
0.712166780588
```

Разработка смешанного ансамбля

УНИВЕРСИТЕТ ИТМО

```
# get a list of base models
def get_models():
models = list()
models.append(('lr', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
models.append(('cart', DecisionTreeClassifier()))
models.append(('svm', SVC(probability=True)))
models.append(('bayes', GaussianNB()))
return models
```

```
...
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict(X_val)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store predictions as input for blending
meta_X.append(yhat)
```

Разработка смешанного ансамбля

УНИВЕРСИТЕТ ИТМО

```
...
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
```

```
...
# define blending model
blender = LogisticRegression()
# fit on predictions from base models
blender.fit(meta_X, y_val)
```

ITMO more than a UNIVERSITY

Разработка смешанного ансамбля

УНИВЕРСИТЕТ ИТМО

```
# fit the blending ensemble
def fit_ensemble(models, X_train, X_val, y_train, y_val):
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict(X_val)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store predictions as input for blending
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# define blending model
blender = LogisticRegression()
# fit on predictions from base models
blender.fit(meta_X, y_val)
return blender
```

ITMOre than a UNIVERSITY

Разработка смешанного ансамбля

УНИВЕРСИТЕТ ИТМО

```
# make a prediction with the blending ensemble
def predict_ensemble(models, blender, X_test):
# make predictions with base models
meta_X = list()
for name, model in models:
# predict with base model
yhat = model.predict(X_test)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store prediction
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# predict
return blender.predict(meta_X)
```

ITMO more than a UNIVERSITY

Смешанный ансамбль для задачи классификации

УНИВЕРСИТЕТ ИТМО

```
# test classification dataset
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=10000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
# summarize the dataset
print(X.shape, y.shape)
```

```
...
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# split training set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.33, random_state=1)
# summarize data split
print('Train: %s, Val: %s, Test: %s' % (X_train.shape, X_val.shape, X_test.shape))
```

```
...
# create the base models
models = get_models()
# train the blending ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
# make predictions on test set
yhat = predict_ensemble(models, blender, X_test)
```

```
...
# evaluate predictions
score = accuracy_score(y_test, yhat)
print('Blending Accuracy: %.3f' % score)
```

Смешанный ансамбль для задачи классификации

УНИВЕРСИТЕТ ИТМО

```python
# blending ensemble for classification using hard voting
from numpy import hstack
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# get the dataset
def get_dataset():
X, y = make_classification(n_samples=10000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
return X, y

# get a list of base models
def get_models():
models = list()
models.append(('lr', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
models.append(('cart', DecisionTreeClassifier()))
models.append(('svm', SVC()))
models.append(('bayes', GaussianNB()))
return models

# fit the blending ensemble
def fit_ensemble(models, X_train, X_val, y_train, y_val):
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict(X_val)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store predictions as input for blending
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# define blending model
blender = LogisticRegression()
# fit on predictions from base models
blender.fit(meta_X, y_val)
return blender
```

```python
# make a prediction with the blending ensemble
def predict_ensemble(models, blender, X_test):
# make predictions with base models
meta_X = list()
for name, model in models:
# predict with base model
yhat = model.predict(X_test)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store prediction
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# predict
return blender.predict(meta_X)

# define dataset
X, y = get_dataset()
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# split training set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.33, random_state=1)
# summarize data split
print('Train: %s, Val: %s, Test: %s' % (X_train.shape, X_val.shape, X_test.shape))
# create the base models
models = get_models()
# train the blending ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
# make predictions on test set
yhat = predict_ensemble(models, blender, X_test)
# evaluate predictions
score = accuracy_score(y_test, yhat)
print('Blending Accuracy: %.3f' % (score*100))
```

```
Train: (3350, 20), Val: (1650, 20), Test: (5000, 20)
Blending Accuracy: 97.900
```

Смешанный ансамбль для задачи классификации

УНИВЕРСИТЕТ ИТМО

```
# get a list of base models
def get_models():
models = list()
models.append(('lr', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
models.append(('cart', DecisionTreeClassifier()))
models.append(('svm', SVC(probability=True)))
models.append(('bayes', GaussianNB()))
return models

...
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict_proba(X_val)
# store predictions as input for blending
meta_X.append(yhat)

...
# make predictions with base models
meta_X = list()
for name, model in models:
# predict with base model
yhat = model.predict_proba(X_test)
# store prediction
meta_X.append(yhat)
```

ITMOre than a UNIVERSITY

Смешанный ансамбль для задачи классификации

УНИВЕРСИТЕТ ИТМО

```python
# blending ensemble for classification using soft voting
from numpy import hstack
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# get the dataset
def get_dataset():
X, y = make_classification(n_samples=10000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
return X, y

# get a list of base models
def get_models():
models = list()
models.append(('lr', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
models.append(('cart', DecisionTreeClassifier()))
models.append(('svm', SVC(probability=True)))
models.append(('bayes', GaussianNB()))
return models

# fit the blending ensemble
def fit_ensemble(models, X_train, X_val, y_train, y_val):
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict_proba(X_val)
# store predictions as input for blending
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# define blending model
blender = LogisticRegression()
# fit on predictions from base models
blender.fit(meta_X, y_val)
return blender
```

```python
# make a prediction with the blending ensemble
def predict_ensemble(models, blender, X_test):
# make predictions with base models
meta_X = list()
for name, model in models:
# predict with base model
yhat = model.predict_proba(X_test)
# store prediction
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# predict
return blender.predict(meta_X)

# define dataset
X, y = get_dataset()
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# split training set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.33, random_state=1)
# summarize data split
print('Train: %s, Val: %s, Test: %s' % (X_train.shape, X_val.shape, X_test.shape))
# create the base models
models = get_models()
# train the blending ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
# make predictions on test set
yhat = predict_ensemble(models, blender, X_test)
# evaluate predictions
score = accuracy_score(y_test, yhat)
print('Blending Accuracy: %.3f' % (score*100))
```

```
Train: (3350, 20), Val: (1650, 20), Test: (5000, 20)
Blending Accuracy: 98.240
```

Смешанный ансамбль для задачи классификации

УНИВЕРСИТЕТ ИТМО

```
# evaluate base models on the entire training dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# get the dataset
def get_dataset():
X, y = make_classification(n_samples=10000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
return X, y

# get a list of base models
def get_models():
models = list()
models.append(('lr', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
models.append(('cart', DecisionTreeClassifier()))
models.append(('svm', SVC(probability=True)))
models.append(('bayes', GaussianNB()))
return models

# define dataset
X, y = get_dataset()
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# summarize data split
print('Train: %s, Test: %s' % (X_train_full.shape, X_test.shape))
# create the base models
models = get_models()
# evaluate standalone model
for name, model in models:
# fit the model on the training dataset
model.fit(X_train_full, y_train_full)
# make a prediction on the test dataset
yhat = model.predict(X_test)
# evaluate the predictions
score = accuracy_score(y_test, yhat)
# report the score
print('>%s Accuracy: %.3f' % (name, score*100))
```

```
Train: (5000, 20), Test: (5000, 20)
>lr Accuracy: 87.800
>knn Accuracy: 97.380
>cart Accuracy: 88.200
>svm Accuracy: 98.200
>bayes Accuracy: 87.300
```

Смешанный ансамбль для задачи регрессии

УНИВЕРСИТЕТ ИТМО

```
# test regression dataset
from sklearn.datasets import make_regression
# define dataset
X, y = make_regression(n_samples=10000, n_features=20, n_informative=10, noise=0.3, random_state=7)
# summarize the dataset
print(X.shape, y.shape)
```

```
# get a list of base models
def get_models():
models = list()
models.append(('lr', LinearRegression()))
models.append(('knn', KNeighborsRegressor()))
models.append(('cart', DecisionTreeRegressor()))
models.append(('svm', SVR()))
return models
```

```
...
# define blending model
blender = LinearRegression()

...
# evaluate predictions
score = mean_absolute_error(y_test, yhat)
print('Blending MAE: %.3f' % score)
```

ITMOre than a UNIVERSITY

Смешанный ансамбль для задачи регрессии

УНИВЕРСИТЕТ ИТМО

```python
# evaluate blending ensemble for regression
from numpy import hstack
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

# get the dataset
def get_dataset():
X, y = make_regression(n_samples=10000, n_features=20, n_informative=10, noise=0.3, random_state=7)
return X, y

# get a list of base models
def get_models():
models = list()
models.append(('lr', LinearRegression()))
models.append(('knn', KNeighborsRegressor()))
models.append(('cart', DecisionTreeRegressor()))
models.append(('svm', SVR()))
return models

# fit the blending ensemble
def fit_ensemble(models, X_train, X_val, y_train, y_val):
# fit all models on the training set and predict on hold out set
meta_X = list()
for name, model in models:
# fit in training set
model.fit(X_train, y_train)
# predict on hold out set
yhat = model.predict(X_val)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store predictions as input for blending
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# define blending model
blender = LinearRegression()
# fit on predictions from base models
blender.fit(meta_X, y_val)
return blender
```

```python
# make a prediction with the blending ensemble
def predict_ensemble(models, blender, X_test):
# make predictions with base models
meta_X = list()
for name, model in models:
# predict with base model
yhat = model.predict(X_test)
# reshape predictions into a matrix with one column
yhat = yhat.reshape(len(yhat), 1)
# store prediction
meta_X.append(yhat)
# create 2d array from predictions, each set is an input feature
meta_X = hstack(meta_X)
# predict
return blender.predict(meta_X)

# define dataset
X, y = get_dataset()
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# split training set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.33, random_state=1)
# summarize data split
print('Train: %s, Val: %s, Test: %s' % (X_train.shape, X_val.shape, X_test.shape))
# create the base models
models = get_models()
# train the blending ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
# make predictions on test set
yhat = predict_ensemble(models, blender, X_test)
# evaluate predictions
score = mean_absolute_error(y_test, yhat)
print('Blending MAE: %.3f' % score)
```

```
Train: (3350, 20), Val: (1650, 20), Test: (5000, 20)
Blending MAE: 0.237
```

Смешанный ансамбль для задачи регрессии

УНИВЕРСИТЕТ ИТМО

```python
# evaluate base models in isolation on the regression dataset
from numpy import hstack
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

# get the dataset
def get_dataset():
X, y = make_regression(n_samples=10000, n_features=20, n_informative=10, noise=0.3, random_state=7)
return X, y

# get a list of base models
def get_models():
models = list()
models.append(('lr', LinearRegression()))
models.append(('knn', KNeighborsRegressor()))
models.append(('cart', DecisionTreeRegressor()))
models.append(('svm', SVR()))
return models

# define dataset
X, y = get_dataset()
# split dataset into train and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
# summarize data split
print('Train: %s, Test: %s' % (X_train_full.shape, X_test.shape))
# create the base models
models = get_models()
# evaluate standalone model
for name, model in models:
# fit the model on the training dataset
model.fit(X_train_full, y_train_full)
# make a prediction on the test dataset
yhat = model.predict(X_test)
# evaluate the predictions
score = mean_absolute_error(y_test, yhat)
# report the score
print('>%s MAE: %.3f' % (name, score))
```

```
Train: (5000, 20), Test: (5000, 20)
>lr MAE: 0.236
>knn MAE: 100.169
>cart MAE: 133.744
>svm MAE: 138.195
```

Ссылки

1. Ensemble-methods-notebooks
   https://github.com/gkunapuli/ensemble-methods-notebooks
2. CatBoost vs. Light GBM vs. XGBoost
   https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db
3. Ensemble methods scikit-learn
   https://scikit-learn.org/stable/modules/ensemble.html
4. История развития ансамблевых методов классификации в машинном обучении
   https://www.researchgate.net/publication/278019662_Istoria_razvitia_ansamblevyh_metodov_klassifikacii_v_masinnom_obucenii