集成学习作业说明

TA 吴韫琛
作业提交邮箱 nju_ml@163.com

集成学习

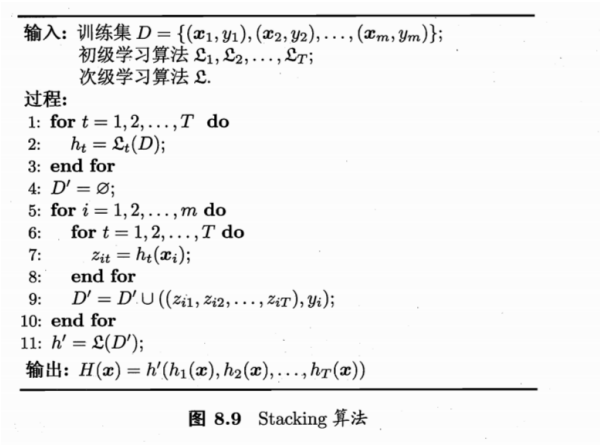
集成学习（Ensemble learning）将多个机器学习算法结合在一起，每个算法被称为**个体学习器。**总体上可以将集成学习方法分为两类：

- 个体学习器间存在强依赖关系、必须串行生成的序列化方法，典型的有 Boosting 等
- 个体学习器不存在强依赖关系、可同时生成的并行化方法，例如 Bagging 与随机森林（Random Forest）

结合个体学习器通常有许多结合策略，包括对于分类问题的投票法以及针对回归问题的平均法。此外，还可以使用另外一个机器学习算法结合个体机器学习器的结果，这一方法就是 **Stacking** 方法。

Stacking算法

在 Stacking 方法中，个体学习器被称为初级学习器，用于结合的学习器称为次级学习器或元学习器（meta-learner），次级学习器用于训练的次级训练集是在训练集上用初级学习器得到的。参考下面这段伪代码：



输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \ldots, \mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.

过程：

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathfrak{L}_t(D)$;
3: **end for**
4: $D' = \varnothing$;
5: **for** $i = 1, 2, \ldots, m$ **do**
6: **for** $t = 1, 2, \ldots, T$ **do**
7: $z_{it} = h_t(\boldsymbol{x}_i)$;
8: **end for**
9: $D' = D' \cup ((z_{i1}, z_{i2}, \ldots, z_{iT}), y_i)$;
10: **end for**
11: $h' = \mathfrak{L}(D')$;

输出： $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

图 8.9 Stacking 算法

其中：

- 过程 1~3 训练个体学习器，也就是初级学习器。
- 过程 5~9 使用训练出来的个体学习器来得到预测的结果，预测的结果当做次级学习器的训练集。
- 过程 11 使用初级学习器预测的结果训练出次级学习器，得到我们最后训练的模型。

如果想要预测一个数据的输出，只需要把这条数据用初级学习器预测，然后将预测后的结果用次级学习器预测即可。但是简单按照上述流程实现存在问题。在原始数据集D上面训练的模型，然后用这些模型再D上面再进行预测得到的次级训练集容易出现过拟合的现象。需要更换一种思路，使用交叉验证的想法实现 Stacking 模型。次级训练集的构成不是直接由模型在训练集D上面预测得到，而是使用交叉验证的方法，将训练集D分为k份。对于每一份，用剩余数据集训练模型，然后预测出这一份的结果，重复上面步骤直到每一份都预测出来。这样就可以避免过拟合的情况。并且在构造次级训练集的同时把测试集的次级数据一起构造出来。

Bagging算法

Bagging的基本思想是通过对训练数据进行重采样，生成多个不同的训练集，随后在这些训练集上训练多个模型，最后将这些模型的输出进行汇总。Bagging算法的优势是能够降低模型方差，提高模型的鲁棒性。

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $\quad h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: **end for**

输出： $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$

参考上述伪代码，算法的步骤并不复杂，可分为如下三个阶段：

- 重采样：从原始训练集中有放回地抽取多个子集（每个子集大小与原始集相同）
- 模型训练：在每个子集上训练一个独立的模型
- 结果汇总：对所有模型的预测结果进行聚合（针对回归任务使用平均法，分类任务使用投票法）

AdaBoosting算法

Adaboost是一种迭代算法，核心思想是针对同一个训练集训练不同的分类器(弱分类器)，然后将这些弱分类器组合成一个更强的最终分类器（强分类器）。

在算法中，后一个模型的训练永远是在前一个模型的基础上完成。组合的策略是通过提高前一轮分类器分类错误的样本的权值，降低分类分类正确的样本权值，对于那些没有被分类正确的样本便会得到后面分类器更多的关注。这些训练完成的多个弱分类器将通过多数加权投票的方法组合，以加大误差率小的分类器，减少误差率大的分类器，使其在表决中起到较小的作用。

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad h_t = \mathfrak{L}(D, \mathcal{D}_t)$;
4: $\quad \epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;
5: $\quad$ **if** $\epsilon_t > 0.5$ **then break**
6: $\quad \alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;
7: $\quad \mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
$\qquad\qquad = \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$
8: **end for**

输出： $H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

其迭代流程大致为：

- 1~3使用加权样本集（初始权重分布为均匀分布）来训练数据，得到弱分类器
- 4计算弱分类器的分类误差率

- 6计算弱分类器的系数
- 7更新样本集的权重分布

代码实现

Stacking算法实现

```
import numpy as np
from sklearn.model_selection import KFold

np.random.seed(2025)
def get_stacking(clf, x_train, y_train, x_test, n_folds=10):
    '''
    stacking的核心算法，使用交叉验证的方法得到次级训练集
    '''
    train_num, test_num = x_train.shape[0], x_test.shape[0]
    second_level_train_set = np.zeros((train_num,))
    second_level_test_set = np.zeros((test_num,))
    test_nfolds_sets = np.zeros((test_num, n_folds))
    kf = KFold(n_splits=n_folds)

    for i,(train_index, test_index) in enumerate(kf.split(x_train)):
        x_tra, y_tra = x_train[train_index], y_train[train_index]
        x_tst, y_tst =  x_train[test_index], y_train[test_index]

        clf.fit(x_tra, y_tra)

        second_level_train_set[test_index] = clf.predict(x_tst)
        test_nfolds_sets[:,i] = clf.predict(x_test)

    second_level_test_set[:] = test_nfolds_sets.mean(axis=1)
    return second_level_train_set, second_level_test_set
```

初级学习器与次级学习器构造

```
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, ExtraTreesClassifier,
StackingClassifier)
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


rf_model = RandomForestClassifier()
adb_model = AdaBoostClassifier()
gdbc_model = GradientBoostingClassifier()
et_model = ExtraTreesClassifier()
svc_model = SVC()

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
train_x, test_x, train_y, test_y = train_test_split(iris.data, iris.target,
test_size=0.2)
```

```
train_sets = []
test_sets = []
base_classifiers = [rf_model, adb_model, gdbc_model, et_model, svc_model]

for clf in base_classifiers:
    train_set, test_set = get_stacking(clf, train_x, train_y, test_x)
    train_sets.append(train_set)
    test_sets.append(test_set)

meta_train = np.concatenate([result_set.reshape(-1,1) for result_set in
train_sets], axis=1)
meta_test = np.concatenate([y_test_set.reshape(-1,1) for y_test_set in
test_sets], axis=1)

# 使用决策树作为我们的次级分类器
from sklearn.tree import DecisionTreeClassifier
meta_clf = DecisionTreeClassifier()
meta_clf.fit(meta_train, train_y)
predict_y = meta_clf.predict(meta_test)

accuracy = accuracy_score(test_y, predict_y)
print(f"Stacking Classifier Accuracy: {accuracy:.4f}")
```

封装为自定义类

在这一部分，我们将上面实现的 Stacking 算法封装为自定义类，实现 `fit` 与 `predict` 方法方便后续调用。

```
from sklearn.model_selection import KFold
from sklearn.base import BaseEstimator, RegressorMixin, TransformerMixin, clone

class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # 将原来的模型clone出来，并且进行实现fit功能
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=156)

        #对于每个模型，使用交叉验证的方法来训练初级学习器，并且得到次级训练集
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred

        # 使用次级训练集来训练次级学习器
```

```
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self

    #在上面的fit方法当中，我们已经将我们训练出来的初级学习器和次级学习器保存下来了
    #predict的时候只需要用这些学习器构造我们的次级预测数据集并且进行预测就可以了
    def predict(self, X):
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in
base_models]).mean(axis=1)
            for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)

stacking_avg = StackingAveragedModels(base_models=base_classifiers, meta_model=
meta_clf)
stacking_avg.fit(train_x, train_y)
pred_y = stacking_avg.predict(test_x)
accuracy = accuracy_score(test_y, pred_y)
print(f"Stacking Classifier Accuracy: {accuracy:.4f}")
```

Bagging算法

```
class MyBaggingClassifier:
    def __init__(self, base_learner, n_learners):
        self.learners = [clone(base_learner) for _ in range(n_learners)]

    def fit(self, X, y):
        for learner in self.learners:
            examples = np.random.choice(
                np.arange(len(X)), int(len(X)), replace=True)

            learner.fit(X[examples, :], y[examples])

    def predict(self, X):
        preds = [learner.predict(X) for learner in self.learners]
        return self._aggregate(np.array(preds))

    def _aggregate(self, predictions):
        # 分类任务投票，回归任务平均
        final_pred = np.apply_along_axis(
            lambda x: np.bincount(x).argmax(),
            axis=0,
            arr=predictions.astype(int)
        )
        return final_pred
```

AdaBoost算法

```
class MyAdaBoostClassifier:
    def __init__(self, base_learner, n_learners):
        self.learners = [clone(base_learner) for _ in range(n_learners)]
        self.learning_rate = 1.0
        self.weak_classifiers = []
```

```
    def fit(self, X, y):
        '''
        算法主体部分
        '''
        sample_weights = np.ones_like(y) / len(y)
        for clf in self.learners:
            clf = DecisionTreeClassifier(max_depth=1)
            clf.fit(X, y, sample_weight=sample_weights)

            y_pred = clf.predict(X)

            incorrect = np.sum(sample_weights * (y != y_pred))
            error_rate = incorrect / np.sum(sample_weights)

            if error_rate > 0.5:
                continue

            alpha = np.log((1.0 - error_rate) / error_rate) / 2.0
            self.weak_classifiers.append((clf, alpha))
            sample_weights *= np.exp(-alpha * y * y_pred)
            sample_weights /= np.sum(sample_weights)  # 权重归一化

    def predict(self, X):
        votes = np.zeros((X.shape[0],))
        for clf, alpha in self.weak_classifiers:
            votes += alpha * clf.predict(X)
        return np.sign(votes)
```

与sklearn实现对比

在这一部分，我们系统地将本次作业介绍实现的三种集成学习算法与sklearn库中已有的方法进行性能对比：

```
stacking_avg = MyStackingClassifier(base_models=base_classifiers, meta_model=
meta_clf)
stacking_avg.fit(train_x, train_y)
pred_y = stacking_avg.predict(test_x)
accuracy = accuracy_score(test_y, pred_y)
print(f"My Stacking Classifier Accuracy: {accuracy:.4f}")

# base_classifier的格式与上面我们自己实现的稍有不同
base_classifiers = [
    ('rf', rf_model),
    ('adb', adb_model),
    ('gdbc', gdbc_model),
    ('et', et_model),
    ('svc', svc_model)
]

stacking_clf = StackingClassifier(estimators=base_classifiers,
final_estimator=meta_clf, cv=5)
stacking_clf.fit(train_x, train_y)
pred_y = stacking_clf.predict(test_x)
accuracy = accuracy_score(test_y, pred_y)
print(f"Sklearn Stacking Classifier Accuracy: {accuracy:.4f}")
```

```
base_classifier = DecisionTreeClassifier()
my_bagging_clf = MyBaggingClassifier(base_learner=base_classifier,
n_learners=100)

my_bagging_clf.fit(train_x, train_y)
pred_y = my_bagging_clf.predict(test_x)
accuracy = accuracy_score(test_y, pred_y)
print(f"My Bagging Classifier Accuracy: {accuracy:.4f}")


bagging_clf = BaggingClassifier(base_estimator=base_classifier,
                                n_estimators=100,
                                max_samples=0.5,
                                max_features=0.5,
                                random_state=42)

bagging_clf.fit(train_x, train_y)
pred_y = bagging_clf.predict(test_x)
accuracy = accuracy_score(test_y, pred_y)
print(f"Sklearn Bagging Classifier Accuracy: {accuracy:.4f}")


'''
数据准备阶段：
标准AdaBoost算法仅适用于二分类任务，上面使用的iris三分类数据集不再适用，需要我们重新生成
'''
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2,
n_redundant=0, random_state=42)
y = np.where(y == 0, -1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

dt_clf = DecisionTreeClassifier(max_depth=1)
my_adaboost_clf = MyAdaBoostClassifier(dt_clf, n_learners=50)
my_adaboost_clf.fit(X_train, y_train)
pred_y = my_adaboost_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred_y)
print(f"My AdaBoost Classifier Accuracy: {accuracy:.4f}")


adaboost_clf = AdaBoostClassifier(n_estimators=50, random_state=42)
adaboost_clf.fit(X_train, y_train)
pred_y = adaboost_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred_y)
print(f"Sklearn AdaBoost Classifier Accuracy: {accuracy:.4f}")
```

课后习题

- 标准的 AdaBoosting 算法只支持二分类，请查阅相关文档资料，实现一个支持对 iris 数据集进行三分类的 AdaBoosting 变体算法。

- 在 `sklearn.ensemble` 库中还整合了其他集成学习方法的实现，例如 `VotingClassifier` 与 `GradientBoostingClassifier` 等。请查阅相关文档资料，调用至少两种方法对 iris 数据集进行分类。同时还需要在不调库的情况下自己实现所选的两种方法，推荐封装成自定义类实现 `fit` 与 `predict` 方法，并与 `sklearn` 的实现对比。
- 在示例代码中我们主要给出了针对iris三分类数据集各种集成学习方法的实现。请替换为一个回归问题的数据集，实现示例代码中给出的三种算法 Stacking, Bagging 以及 AdaBoost 针对回归任务的实现 Regressor 版本，并与 `sklearn` 的实现进行对比。