实验六

基于集成学习的Amazon用户评价质量预测

2022年8月9日

1. 集成模型实现

Bagging

```python
import multiprocessing as mp
import copy
import numpy as np
from sklearn.datasets import dump_svmlight_file
class Bagging():
    '''
    base_models 使用的模型
    n_models 模型数量
    features 使用特征比例
    '''
    def __init__(self , base_models = None, n_models = 10, features = 1):
        if type(base_models) == list:
            self.models = list(base_models)
        else:
            self.models = []
            for i in range(n_models):
                self.models.append(copy.deepcopy(base_models))
        self.n_models = n_models
        self.features = features


    def fit(self, x, y):
        n, m = np.shape(x)
        if self.features < 1: # 选择特征
            self.feature_select = [np.random.choice(m, int(m*self.features) , replace=False) for i in range(self.n_mod
        else:
            self.feature_select = [list(range(m))] * self.n_models
        for i in range(self.n_models):
            print('.', end='')
            rnd = np.random.choice(n, n , replace=True) #  有放回随机抽取训练样本
            x_train = x[rnd, :][:, self.feature_select[i]]
            y_train = y[rnd]
            self.models[i].fit(x_train, y_train)   # 训练模型

    def predict(self, x):
        result = []
        for i in range(self.n_models):  # 每个模型预测
            result.append(self.models[i].predict(x[:, self.feature_select[i]]))

        return np.array([np.argmax(np.bincount(x)) for x in np.array(result).T])  # 返回预测数最多类别

    def predict_proba(self, x):
        result = []
        for i in range(self.n_models):
            result.append(self.models[i].predict_proba(x[:, self.feature_select[i]])[:,1])
        return np.array([sum(x) / len(x) for x in np.array(result).T])    # 返回预测为1的概率
```

Bagging 采用有放回抽样的方式，抽取多组训练样本，训练多个模型，再用这些模型的预测结果取多数投票。

AdaBoost

```python
class AdaBoost():
    '''
    base_models 使用的模型
    n_models 模型数量
    features 使用特征比例
    '''
    def __init__(self , base_models = None, n_models = 10, features = 1):
        self.n_models = n_models
        self.base_models = base_models
        self.features = features


    def fit(self, x, y):
        n, m = np.shape(x)
        model_weights = []
        models = []
        sample_weight = np.ones(n) / n # 初始化样本权重为1/n
        if self.features < 1:
            self.feature_select = [np.random.choice(m, int(m*self.features) , replace=False) for i in range(self.n_mode
        else:
            self.feature_select = [list(range(m))] * self.n_models
        for i in range(self.n_models):
            print('.', end='')
            train_x = x[:, self.feature_select[i]]
            model = copy.deepcopy(self.base_models)
            model.fit(train_x, y, sample_weight = sample_weight)
            pred = model.predict(train_x)
            error_rate = sum((y != pred) * sample_weight)  # 计算错误率
            if error_rate > 0.5:
                print("Error: need to use a best model")
            m_w = 1.0 * error_rate / (1 - error_rate)      # 计算模型权重
            models.append(model)
            model_weights.append(m_w)
            sample_weight *= ((y==pred) * m_w + (y!=pred))   #更新样本权重
            sample_weight = sample_weight / sum(sample_weight)   # 样本权重归一化

        self.models = models
        self.model_weights = np.array([np.log2(1 /  w) for w in model_weights])    #计算模型权重
        self.model_weights /= sum(self.model_weights)


    def predict(self, x):
        result = np.array([self.models[i].predict(x[:, self.feature_select[i]]) for i in range(self.n_models)])
        result = result.T
        pred = []
        for y in result:
            dt = {}
            for k in set(y):
                dt[k] = 0
            for i in range(len(y)):
                dt[y[i]] += self.model_weights[i]   # 计算各类的概率
            pred.append(max(dt, key=dt.get))  # 选择概率最高的作为预测结果
        return np.array(pred)


    def predict_proba(self, x):
        result = np.array([self.models[i].predict(x[:, self.feature_select[i]]) for i in range(self.n_models)])
        result = result.T
        return (result*self.model_weights).mean(axis=1)   # 各模型概率取平均
```

AdaBoost使用一个训练集，每次训练一个模型，并用训练模型的预测结果更新样本权重并计算模型权重。根据各模型的预测结果和模型权重加权作为预测的结果。

2. 读取数据

```
In [1]: import pandas as pd
        train_df = pd.read_csv('./data/train.csv', sep='\t')

In [2]: train_df
```

Out[2]:

	reviewerID	asin	reviewText	overall	votes_up	votes_all	label
0	7885	3901	First off, allow me to correct a common mistak...	5.0	6	7	0
1	52087	47978	I am really troubled by this Story and Enterta...	3.0	99	134	0
2	5701	3667	A near-perfect film version of a downright glo...	4.0	14	14	1
3	47191	40892	Keep your expectations low. Really really low...	1.0	4	7	0
4	40957	15367	"they dont make em like this no more..."well.....	5.0	3	6	0
...	...	...	...	...	...	...	...
57034	58315	29374	If you like beautifully shot, well acted films...	2.0	12	21	0
57035	23328	45548	This is a great set of films Wayne did Fox and...	5.0	15	18	0
57036	27203	42453	It's what's known as a comedy of manners. It's...	3.0	4	5	0
57037	33992	44891	Ellen can do no wrong as far a creating wonder...	5.0	4	5	0
57038	27478	19198	I agree with everyone else that this is a grea...	2.0	5	5	1

主要的数据为reviewText，即顾客的评论内容。拆分训练集和验证集，并用TFIDF提取文本特征。

```
[11]: from sklearn.model_selection import train_test_split
      train_x, test_x, train_y, test_y = train_test_split(train_df['reviewText'], train_df['label'], random_state=2022, )

[12]: from sklearn.feature_extraction.text import TfidfVectorizer as TFIDF
      tfidf = TFIDF(use_idf=1, stop_words='english', ngram_range=(2, 3), min_df=20, max_df=0.5, analyzer='word', max_features

[13]: tfidf.fit(train_x)

[13]: TfidfVectorizer(max_df=0.5, max_features=5000, min_df=20, ngram_range=(2, 3),
                      stop_words='english', use_idf=1)

[14]: tfidf.vocabulary_

[14]: {'went film': 4834,
       've seen': 4650,
       'seen films': 4035,
       'sex scenes': 4105,
       'scene involving': 3949,
       'film reason': 1536,
       'gets stars': 1701,
       'marlon brando': 2799,
       'point film': 3520,
       'film good': 1470,
       'complete set': 692,
       'previously released': 3568,
       'complete collection': 689,
       'don believe': 990,
       'star trek': 4233,
       'don feel': 998,
       'did make': 828,
       'video quality': 4682,
       'bit better': 400,
```

将文本转化为数字向量。

```
In [15]: train_vec = tfidf.transform(train_x)
         train_vec.toarray().shape

Out[15]: (42779, 5000)

In [16]: test_vec = tfidf.transform(test_x)
```

3. 使用决策树和SVM训练

```
In [18]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import roc_auc_score
         dt = DecisionTreeClassifier(max_depth=10)
         dt.fit(np.array(train_vec.todense()), np.array(train_y))
         y_pred = dt.predict(np.array(test_vec.todense()))
         roc_auc_score(y_pred, test_y)

Out[18]: 0.6290368491643882
```

决策树的预测结果AUC为0.629

```
from sklearn.svm import LinearSVC
svc =LinearSVC()
svc.fit(np.array(train_vec.todense()), np.array(train_y))
y_pred = svc.predict(np.array(test_vec.todense()))
roc_auc_score(y_pred, test_y)

0.6404753407109138
```

线性SVM的预测结果AUC为0.640

4.Bagging集成

```
In [26]: bagging_dt = Bagging(base_models=DecisionTreeClassifier(max_depth=3), n_models=100)
         bagging_dt.fit(np.array(train_vec.todense()), np.array(train_y))

         ..........................................................................................

In [27]: y_pred = bagging_dt.predict(np.array(test_vec.todense()))
         roc_auc_score(y_pred, test_y)

Out[27]: 0.7635381593714928
```

100个决策树模型的bagging集成，预测结果AUC为0.763

```
In [36]: from sklearn.svm import LinearSVC
         bagging_svc = Bagging(base_models=LinearSVC(C=0.01), n_models=100)
         bagging_svc.fit(np.array(train_vec.todense()), np.array(train_y))

         ..........................................................................................

In [37]: y_pred = bagging_svc.predict(np.array(test_vec.todense()))
         roc_auc_score(y_pred, test_y)

Out[37]: 0.7636121246140892
```

100个线性SVM的bagging集成，预测结果AUC为0.764

5.AdaBoost集成

```
[52]: adaboost_dt = AdaBoost(base_models=DecisionTreeClassifier(max_depth = 3), n_models=100)
      adaboost_dt.fit(np.array(train_vec.todense()), np.array(train_y))
      y_pred = adaboost_dt.predict(np.array(test_vec.todense()))
      roc_auc_score(y_pred, test_y)

      ..........................................................................................

[52]: 0.8052593424068099
```

100个决策树模型的AdaBoost集成，预测结果AUC为0.805

```
[60]: adaboost_svc = AdaBoost(base_models=LinearSVC(C=0.01), n_models=100)
      adaboost_svc.fit(np.array(train_vec.todense()), np.array(train_y))
      y_pred = adaboost_svc.predict(np.array(test_vec.todense()))
      roc_auc_score(y_pred, test_y)

      ..........................................................................................

[60]: 0.8331189234283753
```

100个线性SVM模型的AdaBoost集成，预测结果为0.833

6.预测

读取测试集数据，用上述集成模型进行预测，并将预测结果取平均值作为最终的结果，并输出csv文件。

```
[67]: test_df = pd.read_csv('./data/test.csv', sep='\t')
      test_x = tfidf.transform(test_df['reviewText'])

[68]: bagging_dt_prob = bagging_dt.predict_proba(test_x)

[69]: adaboost_dt_prob = adaboost_dt.predict_proba(test_x)

[71]: test_df['Predicted'] = (bagging_dt_prob + adaboost_dt_prob) / 2
      test_df.to_csv("result.csv", columns = ['Id', 'Predicted'], index=False)
```

7.总结

实验中对比了决策树和SVM的Bagging和AdaBoost的预测结果的AUC值：

	Base	Bagging	AdaBoost
决策树	0.629	0.763	0.805
线性SVM	0.640	0.764	0.833

可以看到Bagging 和 AdaBoost集成后的预测效果都比单独的模型要好。而且实验中AdaBoost集成比Bagging效果更好。因为AdaBoost给每个分类器不同的权重，对于分类误差小的分类器会有更大的权重，而Bagging只是训练样本的不同，所以在多个模型的集成下，AdaBoost的效果要更好。