Sklearn 实战案例：20 类新闻分类

API

sklearn.naive_bayes.MultinomialNB(alpha = 1.0)

朴素贝叶斯分类

alpha：拉普拉斯平滑系数

from sklearn.model_selection import train_test_split 进行数据集分割

sklearn.datasets.fetch_20newsgroups 获取数据集

x_train, x_test, y_train, y_test = train_test_split(news.data, news.target, test_size=0.3)

获取训练集、测试集

使用 sklearn 对数据集做朴素贝叶斯分类，使用 tfidf 对文本数据进行特征抽取。训练模型，根据模型获取准确率

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

# 获取新闻的数据，20 个类别
news = fetch_20newsgroups(subset='all')

# 进行数据集分割
x_train, x_test, y_train, y_test = train_test_split(news.data, news.target, test_size=0.3)

# 对于文本数据，进行特征抽取
tf = TfidfVectorizer()

x_train = tf.fit_transform(x_train)
# 这里打印出来的列表是：训练集当中的所有不同词的组成的一个列表
print(tf.get_feature_names())

# 不能调用 fit_transform
x_test = tf.transform(x_test)

# estimator 估计器流程
mlb = MultinomialNB(alpha=1.0)

mlb.fit(x_train, y_train)

# 进行预测
y_predict = mlb.predict(x_test)

print("预测每篇文章的类别：", y_predict[:100])
print("真实类别为：", y_test[:100])
print("预测准确率为：", mlb.score(x_test, y_test))
```

sklearn 阅读内容

sklearn 转换器和估计器

- 转换器

1. 实例化 (实例化的是一个转换器类(Transformer))
2. 调用 fit_transform(对于文档建立分类词频矩阵，不能同时调用)

我们把特征工程的接口称之为转换器，其中转换器调用有这么几种形式

1. fit_transform
2. fit
3. transform

- 估计器(sklearn 机器学习算法的实现)

在 sklearn 中，估计器(estimator)是一个重要的角色，是一类实现了算法的 API

- ✧ 用于分类的估计器：

sklearn.neighbors k-近邻算法
sklearn.naive_bayes 贝叶斯
sklearn.linear_model.LogisticRegression 逻辑回归
sklearn.tree 决策树与随机森林

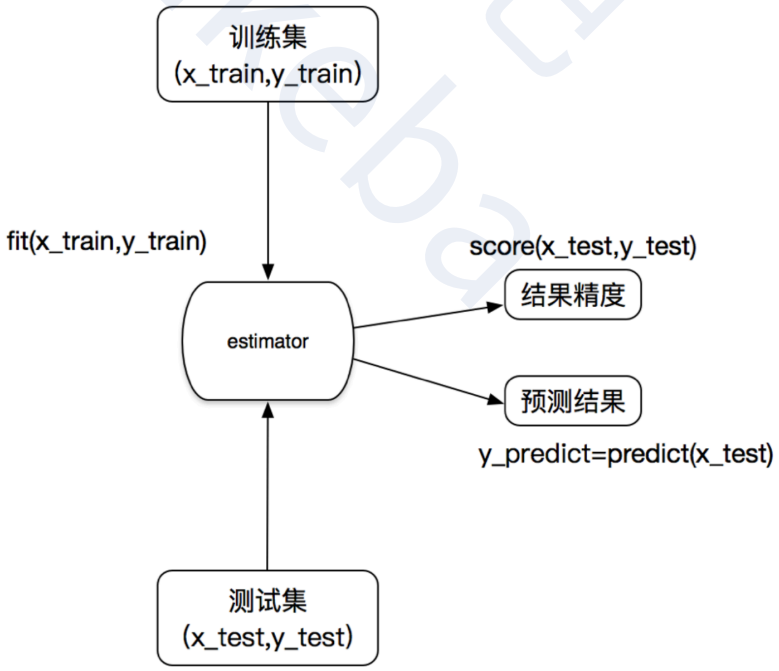
- ✧ 用于回归的估计器：

sklearn.linear_model.LinearRegression 线性回归
sklearn.linear_model.Ridge 岭回归

- ✧ 用于无监督学习的估计器

sklearn.cluster.KMeans 聚类

估计器工作流程



K-近邻算法 API

sklearn.neighbors.KNeighborsClassifier(n_neighbors=5,algorithm='auto')

- ✧ n_neighbors：int,可选（默认= 5），k_neighbors 查询默认使用的邻居数
- ✧ algorithm：{‘auto’，‘ball_tree’，‘kd_tree’，‘brute’}，可选用于计算最近邻居的算法：‘ball_tree’将会使用 BallTree，‘kd_tree’将使用 KDTree。‘auto’将尝试根据传递给 fit 方法的值来决定最合适的算法。（不同实现方式影响效率）

案例：鸢尾花种类预测

使用 sklearn.datasets.load_iris 获取鸢尾花数据集，
sklearn.model_selection.train_test_split 切分数据集
sklearn.preprocessing.StandardScaler 对数据做标准化处理（思考为什么需要做归一化，有哪儿些算法需要做归一化，哪儿些不需要）
使用 KNN 模型接口进行训练预测，获取准确率

代码：

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# 加载模块
iris = load_iris()

# x_train,x_test,y_train,y_test 为训练集特征值、测试集特征值、训练集目标值、测试集目标值
x_train, x_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=22)
# 3、特征工程：标准化
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
x_test = transfer.transform(x_test)
# 实例化 API
estimator = KNeighborsClassifier(n_neighbors=9)
estimator.fit(x_train, y_train)
# 5、模型评估
# 方法 1：比对真实值和预测值
y_predict = estimator.predict(x_test)
print("预测结果为:\n", y_predict)
print("比对真实值和预测值： \n", y_predict == y_test)
# 方法 2：直接计算准确率
score = estimator.score(x_test, y_test)
print("准确率为： \n", score)
```

K-近邻总结

优点：简单，易于理解，易于实现，无需训练

缺点：

- ✧ 懒惰算法，对测试样本分类时的计算量大，内存开销大
- ✧ 必须指定 K 值，K 值选择不当则分类精度不能保证
- ✧ 使用场景：小数据场景，几千～几万样本，具体场景具体业务去测试

➢ 什么是交叉验证(cross validation)

交叉验证：将拿到的训练数据，分为训练和验证集。以下图为例：将数据分成 5 份，其中一份作为验证集。然后经过 5 次(组)的测试，每次都更换不同的验证集。即得到 5 组模型的结果，取平均值作为最终结果。又称 5 折交叉验证。

交叉验证目的：**为了让被评估的模型更加准确可信**

➢ 超参数搜索-网格搜索(Grid Search)

通常情况下，**有很多参数是需要手动指定的（如 k-近邻算法中的 K 值），这种叫超参数**。但是手动过程繁杂，所以需要对模型预设几种超参数组合。**每组超参数都采用交叉验证来进行评估。最后选出最优参数组合建立模型。**

➢ 模型选择与调优 API

sklearn.model_selection.GridSearchCV(estimator, param_grid=None,cv=None)

对估计器的指定参数值进行详尽搜索

estimator：估计器对象

param_grid：估计器参数(dict){“n_neighbors”:[1,3,5]}

cv：指定几折交叉验证

fit：输入训练数据

score：准确率

结果分析：

- ✧ best*score*:在交叉验证中验证的最好结果_
- ✧ best*estimator*：最好的参数模型
- ✧ cv*results*:每次交叉验证后的验证集准确率结果和训练集准确率结果

鸢尾花案例增加 K 值调优

- 使用 GridSearchCV 构建估计器

```
# 1、获取数据集
iris = load_iris()
# 2、划分数据集
x_train, x_test, y_train, y_test = train_test_split(iris.data,
iris.target, random_state=22)
# 3、特征工程：标准化
# 实例化一个转换器类
transfer = StandardScaler()
# 调用 fit_transform
x_train = transfer.fit_transform(x_train)
```

```
x_test = transfer.transform(x_test)
# 4、KNN 预估器流程
#     1）实例化预估器类
estimator = KNeighborsClassifier()

# 5、模型选择与调优——网格搜索和交叉验证
# 准备要调的超参数
param_dict = {"n_neighbors": [1, 3, 5]}
estimator = GridSearchCV(estimator, param_grid=param_dict, cv=3)
#     2）fit 数据进行训练
estimator.fit(x_train, y_train)
# 5、评估模型效果
# 方法 a：比对预测结果和真实值
y_predict = estimator.predict(x_test)
print("比对预测结果和真实值：\n", y_predict == y_test)
# 方法 b：直接计算准确率
score = estimator.score(x_test, y_test)
print("直接计算准确率：\n", score)
```

- 然后进行评估查看最终选择的结果和交叉验证的结果

```
print("在交叉验证中验证的最好结果：\n", estimator.best_score_)
print("最好的参数模型：\n", estimator.best_estimator_)
print("每次交叉验证后的准确率结果：\n", estimator.cv_results_)
```

- 最终结果

```
比对预测结果和真实值：
 [ True  True  True  True  True  True  True False  True  True  True  True
  True  True  True  True  True  True False  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True]
直接计算准确率：
 0.947368421053
在交叉验证中验证的最好结果：
 0.973214285714
最好的参数模型：
 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
每次交叉验证后的准确率结果：
 {'mean_fit_time': array([ 0.00114751,  0.00027037,  0.00024462]),
'std_fit_time': array([  1.13901511e-03,   1.25300249e-05,   1.11011951e-
05]), 'mean_score_time': array([ 0.00085751,  0.00048693,  0.00045625]),
```

```
'std_score_time': array([  3.52785082e-04,   2.87650037e-05,
5.29673344e-06]), 'param_n_neighbors': masked_array(data = [1 3 5],
             mask = [False False False],
       fill_value = ?)
, 'params': [{'n_neighbors': 1}, {'n_neighbors': 3}, {'n_neighbors': 5}],
'split0_test_score': array([ 0.97368421,  0.97368421,  0.97368421]),
'split1_test_score': array([ 0.97297297,  0.97297297,  0.97297297]),
'split2_test_score': array([ 0.94594595,  0.89189189,  0.97297297]),
'mean_test_score': array([ 0.96428571,  0.94642857,  0.97321429]),
'std_test_score': array([ 0.01288472,  0.03830641,  0.00033675]),
'rank_test_score': array([2, 3, 1], dtype=int32), 'split0_train_score':
array([ 1.        ,  0.95945946,  0.97297297]), 'split1_train_score':
array([ 1.        ,  0.96      ,  0.97333333]), 'split2_train_score':
array([ 1.  ,  0.96,  0.96]), 'mean_train_score': array([ 1.        ,
0.95981982,  0.96876877]), 'std_train_score': array([ 0.        ,
0.00025481,  0.0062022 ])}
```

Kaggle 初步实战-预测 facebook 签到位置

https://www.kaggle.com/navoshta/grid-knn/data

步骤分析

- 对于数据做一些基本处理（这里所做的一些处理不一定达到很好的效果，我们只是简单尝试，有些特征我们可以根据一些特征选择的方式去做处理）
  - 1 缩小数据集范围 DataFrame.query()
  - 2 选取有用的时间特征
  - 3 将签到位置少于 n 个用户的删除

    place_count = data.groupby('place_id').count()

    tf = place_count[place_count.row_id > 3].reset_index()

    data = data[data['place_id'].isin(tf.place_id)]
- 分割数据集
- 标准化处理
- k-近邻预测

代码过程

- 获取数据集

```
# 1、获取数据集
facebook = pd.read_csv("./FBlocation/train.csv")
```

- 缩小数据的范围、选择有用的时间特征和取出标签较少的地点

```
# 2、基本的数据处理，拿到特征值和目标值
# 1）缩小数据范围
facebook = facebook.query("x > 1.0 & x <1.25 & y > 2.0 & y < 2.25")
# 2）选取有用的时间特征
time_value = pd.to_datetime(facebook["time"], unit="s")
time_value = pd.DatetimeIndex(time_value)
facebook["day"] = time_value.day
facebook["hour"] = time_value.hour
facebook["weekday"] = time_value.weekday
# 3）去掉签到较少的地点
place_count = facebook.groupby("place_id").count()
place_count = place_count[place_count["row_id"] > 3]
facebook = facebook[facebook["place_id"].isin(place_count.index)]
```

- 取出数据的特征值和目标值

```
# 4）拿到特征值 x 和目标值 y
x = facebook[["x", "y", "accuracy", "day", "hour", "weekday"]]
y = facebook["place_id"]
```

- 划分成训练集合测试集

```
# 3、数据集的划分
x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=6)
```

- 标准化处理

```
# 4、特征工程：标准化
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
```

```
x_test = transfer.transform(x_test)
```

- K 近邻算法模型进行预测

```
# 5、KNN 预估器流程
estimator = KNeighborsClassifier()
# 6、模型评估
# 方法 1：比对真实值和预测值
y_predict = estimator.predict(x_test)
print("预测结果为:\n", y_predict)
print("比对真实值和预测值： \n", y_predict == y_test)
# 方法 2：直接计算准确率
score = estimator.score(x_test, y_test)
print("准确率为： \n", score)
# 7、交叉验证和网格搜索的结果
print("在交叉验证中验证的最好结果:\n", estimator.best_score_)
print("最好的参数模型:\n", estimator.best_estimator_)
print("每次交叉验证后的准确率结果:\n", estimator.cv_results_)
```