原 Python sklearn数据分析中常用方法

2017年08月01日 15:50:15 青盏 阅读数：3175 标签： python 数据分析 更多

一、数据处理

随机划分训练集和测试集：

```python
from sklearn.model_selection import train_test_split

X_all = data_train.drop(['Survived', 'PassengerId'], axis=1) #只包含特征集，不包含预测目标
y_all = data_train['Survived'] #只包含预测目标

num_test = 0.20 # 测试集占据比例，，如果是整数的话就是样本的数量
# 注意返回值：(X_train,y_train)训练集的特征和label || (X_test,y_test)训练集的特征和label
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=num_test, random_state=23)
# random_state参数表示随机种子，如果为0或不填，每次随机产生的随机数组不同。
```

```python
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
# sss对象用于划分数据集
X = train[0::, 1::]
# X为特征集
y = train[0::, 0]
# y为Label集
for train_index, test_index in sss.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

文本特征提取：

sklearn.feature_extraction.text 文本相关的特征抽取

```
text.CountVectorizer：将文本转换为每个词出现的个数的向量
text.TfidfVectorizer：将文本转换为tfidf值的向量
text.HashingVectorizer：文本的特征哈希
```

```python
# CountVectorizer
ar1 = '今天 今天 天气 不错 我们 愉快 玩耍'
ar2 = '今天 锻炼 舒服 天气 一般'
ar3 = '天气 糟糕'
text = [ar1,ar2,ar3]
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
ct = CountVectorizer()
print(ct.fit_transform(text).todense())
print(ct.vocabulary_)
```

```
[[0 1 2 1 1 1 1 0 0 0]
 [1 0 1 1 0 0 0 0 1 1]
 [0 0 0 1 0 0 0 1 0 0]]
{'今天': 2, '天气': 3, '不错': 1, '我们': 5, '愉快': 4, '玩耍': 6, '锻炼': 9, '舒服': 8, '一般': 0, '糟糕': 7}
```


文本特征向量化，其实就是将所有文本中出现的单词组成一个词典，这个词典可以作为一个向量，对每个样例中出现的次数进行统计，从而每个样例割量。如上图。但如果只是统计词频是不够的，因为常用的语言中有些单词出现频率特别高，但是却没有啥意义。如"the、to"。因此我们需要降低这类单TF-IDF思想是一个词语在一篇文章中出现次数越多, 同时在所有文档中出现次数越少, 越能够代表该文章。

```python
# TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
transformer = TfidfTransformer()
```

```
tfidf = transformer.fit_transform(ct.fit_transform(text))
print(tfidf.todense())
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf2 = TfidfVectorizer()
re = tfidf2.fit_transform(text)
print(re.todense())
```

```
[[ 0.          0.3874216   0.58928822  0.22881744  0.3874216   0.3874216
   0.3874216   0.          0.          0.        ]
 [ 0.50461134  0.          0.38376993  0.29803159  0.          0.          0.
   0.          0.50461134  0.50461134]
 [ 0.          0.          0.          0.50854232  0.          0.          0.
   0.861037    0.          0.        ]]
```


如果词库很大时，生成的词向量维度过大，可以使用hash方法对其进行降维.hash后的词向量就无法解释其意义。

```
from sklearn.feature_extraction.text import HashingVectorizer
vectorizer2=HashingVectorizer(n_features = 6,norm = None)
print(vectorizer2.fit_transform(text).todense())
```

```
[[ 0. -1. -1.  1.  0. -2.]
 [ 0.  1.  1.  1.  0.  0.]
 [ 0.  0.  0.  1. -1.  0.]]
```

二、模型选择

```
# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

逻辑回归:

```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_log
```

```
# 查看特征系数
coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

SVC支持向量机:

```
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
```

```
# # Linear SVC
# linear_svc = LinearSVC()
# linear_svc.fit(X_train, Y_train)
# Y_pred = linear_svc.predict(X_test)
# acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
# acc_linear_svc
```

K近邻学习KNN：

```
# knn = KNeighborsClassifier(n_neighbors = 3)
# knn.fit(X_train, Y_train)
# Y_pred = knn.predict(X_test)
# acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
# acc_knn
```

朴素贝叶斯分类器：

```
# gaussian = GaussianNB()
# gaussian.fit(X_train, Y_train)
# Y_pred = gaussian.predict(X_test)
# acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
# acc_gaussian
```

感知机：

```
# perceptron = Perceptron()
# perceptron.fit(X_train, Y_train)
# Y_pred = perceptron.predict(X_test)
# acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
# acc_perceptron
```

随机梯度下降法：

```
# sgd = SGDClassifier()
# sgd.fit(X_train, Y_train)
# Y_pred = sgd.predict(X_test)
# acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
# acc_sgd
```

决策树：

```
# # Decision Tree
# decision_tree = DecisionTreeClassifier()
# decision_tree.fit(X_train, Y_train)
# Y_pred = decision_tree.predict(X_test)
# acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
# acc_decision_tree
```

随机森林：

```
# # Random Forest
# random_forest = RandomForestClassifier(n_estimators=100)
# random_forest.fit(X_train, Y_train)
# Y_pred = random_forest.predict(X_test)
# random_forest.score(X_train, Y_train)
# acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
# acc_random_forest
```

```
# 基于准确率搜索最佳参数的随机森林
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.model_selection import GridSearchCV

# Choose the type of classifier.
clf = RandomForestClassifier()

```

```python
# Choose some parameter combinations to try
parameters = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(clf, parameters, scoring=acc_scorer)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_

# Fit the best algorithm to the data.
clf.fit(X_train, y_train)
```

遍历模型方法：

```python
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression

classifiers = [
    KNeighborsClassifier(3),
    SVC(probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis(),
    LogisticRegression()]

log_cols = ["Classifier", "Accuracy"]
log      = pd.DataFrame(columns=log_cols)

sss = StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
# sss对象用于划分数据集
X = train[0::, 1::]
# X为特征集
y = train[0::, 0]
# y为Label集

acc_dict = {}

for train_index, test_index in sss.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    for clf in classifiers:
        name = clf.__class__.__name__
        clf.fit(X_train, y_train)
        train_predictions = clf.predict(X_test)
        acc = accuracy_score(y_test, train_predictions)
```

```
        if name in acc_dict:
            acc_dict[name] += acc
        else:
            acc_dict[name] = acc

for clf in acc_dict:
    acc_dict[clf] = acc_dict[clf] / 10.0
    # 计算平均准确率
    log_entry = pd.DataFrame([[clf, acc_dict[clf]]], columns=log_cols)
    log = log.append(log_entry)

plt.xlabel('Accuracy')
plt.title('Classifier Accuracy')

sns.set_color_codes("muted")
sns.barplot(x='Accuracy', y='Classifier', data=log, color="b")
# 画条形图分析
```

叠加多层（2）模型教程

三、模型评估

使用k折交叉验证法：

```
from sklearn.cross_validation import KFold

def run_kfold(clf):
    kf = KFold(891, n_folds=10)
    outcomes = []
    fold = 0
    for train_index, test_index in kf:
        fold += 1
        X_train, X_test = X_all.values[train_index], X_all.values[test_index]
        y_train, y_test = y_all.values[train_index], y_all.values[test_index]
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        outcomes.append(accuracy)
        print("Fold {0} accuracy: {1}".format(fold, accuracy))
    mean_outcome = np.mean(outcomes)
    print("Mean Accuracy: {0}".format(mean_outcome))
run_kfold(clf)
```

四、其他

保存模型：

```
Pickle:
>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0:1])
joblib:
>>> from sklearn.externals import joblib
>>> joblib.dump(clf, 'filename.pkl')
>>> clf = joblib.load('filename.pkl')
```