情感分类器

问题分析

给定的数据集，根据影评来设计情感分类器。首先看下了 aclImdb 的数据集的 README 得知这个数据集合包含了50000 条数据，分别被分为了 25k 训练集，25k 测试集，训练和测试分别包含 neg 负面评价 12500 条 和 pos 正面 12500 条，neg 定义为小于 4分的电影，pos 定义为大于 7分的。另外训练中 upsup 文件夹包含 50000条为分类的文本数据作为无监督学习训练，不过和这次情感分类无关。目前我们要做的就是训练 train 文件夹下的 neg 和 pos 然后用test 的neg 和 pos 去验证。

分类器除了当节课学的逻辑回归，之前还学过朴素贝叶斯、决策树、随机森林等等。作为分类器来看看不同的算法对这次情感分类的效果。由于要使用大量算法，而且之前算法基本熟悉，这次直接采用 SKLearn 中现成的库去实现以上说的几种算法来比较结果。

设计思路

导入SKLearn 中所需要的类。

```
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

使用 SKLearn 引入了 朴素贝叶斯 MultinomialNB、决策树 DecisionTreeClassifier、随机森林 RandomForestClassifier和逻辑回归 DecisionTreeClassifier 4大分类器方法。CountVectorizer 和 TfidTransformer 主要是用来分词和提取文本特征。另外GridSearchCV 和 RandmizedSearchCV 主要是为了后面调参使用。

首先我们通过 datasets.load_files 方法把训练路径和测试路径中的数据加载出来，它会根据文件夹来对数据集分类。由于训练路径中还有 unsup 文件夹，所以我们指定只选择 neg 和pos 两个。

```
MOVIE_PATH = '/Users/jackrex/Desktop/AILesson/L4/aclImdb/train'
MOVIE_TEST_PATH = '/Users/jackrex/Desktop/AILesson/L4/aclImdb/test'
train_movie_data = datasets.load_files(MOVIE_PATH, 'Movie Comments', ['neg', 'pos'], True, True, None, 'strict', 42)
test_movie_data = datasets.load_files(MOVIE_TEST_PATH, 'Movie Test Comments', ['neg', 'pos'], True, True, None,
                                      'strict', 42)
```

其次声明和定义了5中不同的验证方式，包含贝叶斯 MultinomialNB、决策树 DecisionTreeClassifier、随机森林RandomForestClassifier和逻辑回归 DecisionTreeClassifier，另外也单纯使用了训练集的一个朴素交叉验证来对比。

```
if __name__ == '__main__':

    bayesian = MultinomialNB()
    logistic_regression = LogisticRegression()
    random_forest = RandomForestClassifier()
    decision_tree = DecisionTreeClassifier()

    split_bayesian()

    classifier_report(bayesian)
    classifier_report(logistic_regression)
    classifier_report(random_forest)
    classifier_report(decision_tree)
```

交叉验证级，类似第一次贝叶斯交叉的方式。

```
def split_bayesian():
    print('split_bayesian')
    t = time.time()
    x_train, x_test, y_train, y_test = train_test_split(train_movie_data.data, train_movie_data.target, test_size=0.25, random_state=33)
    count_vec = CountVectorizer()
    x_count_train = count_vec.fit_transform(x_train)
    x_count_test = count_vec.transform(x_test)

    mnb_count = MultinomialNB()
    mnb_count.fit(x_count_train, y_train)
    y_count_predict = mnb_count.predict(x_count_test)
    print(classification_report(y_test, y_count_predict, target_names=train_movie_data.target_names))
    print('Time usage: ' + str(time.time() - t))
```

交叉验证逻辑并不复杂，首先我们使用 train_test_split 把train 的数据随机分为 25：75 份，然后使用CountVectorizer 使用fit_transform 进行分词和统计词频转化为词频矩阵，测试集 x_count_test 只是用 tansform 进行文档矩阵转换。

定义朴素贝叶斯 MultinomialNB，调用fit 函数进行训练 train集合，使用predict 来测试 x_count_test，最后使用classificaton_report 将预测报告导出。

classifier_report 和上述和上述类似，不过可以自定义传入分类器， 训练train 文件夹，用 test 来测试。具体实现如下图所示：

```
def classifier_report(classifier):
    t = time.time()
    print(type(classifier))
    count_vec = CountVectorizer()
    x_train = count_vec.fit_transform(train_movie_data.data)
    x_test = count_vec.transform(test_movie_data.data)

    classifier.fit(x_train, train_movie_data.target)

    predicted = classifier.predict(x_test)
    print(classification_report(test_movie_data.target, predicted, target_names=train_movie_data.target_names))
    print('Time usage: ' + str(time.time() - t))
```

并且打出了消耗时间。

结果&调优

按照上述代码执行得到结果：

```
split_bayesian
              precision    recall  f1-score   support

         neg       0.83      0.88      0.86      3183
         pos       0.87      0.81      0.84      3067

   micro avg       0.85      0.85      0.85      6250
   macro avg       0.85      0.85      0.85      6250
weighted avg       0.85      0.85      0.85      6250

Time usage: 5.546748161315918
<class 'sklearn.naive_bayes.MultinomialNB'>
              precision    recall  f1-score   support

         neg       0.78      0.88      0.82     12500
         pos       0.86      0.75      0.80     12500

   micro avg       0.81      0.81      0.81     25000
   macro avg       0.82      0.81      0.81     25000
weighted avg       0.82      0.81      0.81     25000

Time usage: 10.952964782714844
```

交叉验证的 朴素贝叶斯准确率 0.85 由于数据集小，消耗时间也短，所有数据集的测试集会大导致准确率结果略低为 81%。同时可以看到右侧 support 样本数量，花费时间 10.952。

```
<class 'sklearn.linear_model.logistic.LogisticRegression'>
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/l
  FutureWarning)
              precision    recall  f1-score   support

         neg       0.86      0.87      0.87     12500
         pos       0.87      0.86      0.87     12500

   micro avg       0.87      0.87      0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000

Time usage: 20.50180697441101
<class 'sklearn.ensemble.forest.RandomForestClassifier'>
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/fores
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
              precision    recall  f1-score   support

         neg       0.72      0.82      0.77     12500
         pos       0.79      0.68      0.73     12500

   micro avg       0.75      0.75      0.75     25000
   macro avg       0.75      0.75      0.75     25000
weighted avg       0.75      0.75      0.75     25000

Time usage: 17.940468072891235
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
              precision    recall  f1-score   support

         neg       0.72      0.71      0.71     12500
         pos       0.71      0.72      0.72     12500

   micro avg       0.71      0.71      0.71     25000
   macro avg       0.71      0.71      0.71     25000
weighted avg       0.71      0.71      0.71     25000

Time usage: 49.92205500602722
```

逻辑回归平均成功率为 87% 耗时 20s，随机森林准确率 75% 耗时17s，决策树准确率71%耗时 49s。

对比上述4中不同分类器可以看出逻辑回归是属于准确率效果最高的一个，耗时也较少 20s，属于最优算法。并且处于未调参的情况下。如果调参数，准确率还可以上升。

分词优化

使TfidfVectorizer

TfidfVectorizer用于统计vectorizer中每个词语的TF-IDF值TF-IDF倾向于过滤掉常见的词语，保留重要的词语。TF-IDF的主要思想是：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TF-IDF实际上是：TF * IDF

代码实现：

```
count_vec = TfidfVectorizer(max_features=None, strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}', ngram_range=(1, 2))
x_train = count_vec.fit_transform(train_movie_data.data)
print('weight is ' + str(x_train.toarray()))
print(count_vec.get_feature_names())
x_test = count_vec.transform(test_movie_data.data)
```

加入 TFIDF 结果

```
<class 'sklearn.naive_bayes.MultinomialNB'>
              precision    recall  f1-score   support

         neg       0.83      0.92      0.87     12500
         pos       0.91      0.81      0.86     12500

   micro avg       0.87      0.87      0.87     25000
   macro avg       0.87      0.87      0.86     25000
weighted avg       0.87      0.87      0.86     25000

Time usage: 41.37119483947754
<class 'sklearn.linear_model.logistic.LogisticRegression'>
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-pack
  FutureWarning)
              precision    recall  f1-score   support

         neg       0.90      0.88      0.89     12500
         pos       0.88      0.90      0.89     12500

   micro avg       0.89      0.89      0.89     25000
   macro avg       0.89      0.89      0.89     25000
weighted avg       0.89      0.89      0.89     25000
```

可以看到LogicsticRegression neg 效果
提高到了 90%，同时由于使用词频 NB 的neg效果也得到一定提高。而且能看到 NB 的 pos 分类效果91% 好于 LR 的89%

优化调参

LogisticRegression 有很多参数，最开始我们使用的是默认的参数，得到结果是87%，首先我们看看其拥有的参数

LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0,fit_intercept=True, intercept_scaling=1, class_weight='balanced', random_state=None, solver='liblinear ', max_iter=2000, multi_class='ovr', verbose=0, warm_start=False, n_jobs=-1)

具体的就不细讲了，看注释文档或网上资料很多，Refer
https://blog.csdn.net/jark_/article/details/78342644
这里penalty 惩罚项我们选择l2，应为l1 只支持liblinear，l2 除了liblinear 还支持 newton-cg、sag和lbfgs等。
几个重要可调的参数还有 tol(停止求解的标准) ,C(正则化系数λ的倒数 越小的数值表示越强的正则化) ,class_weight(权重 选择balanced 让类库自己计算类型权重) ,solver multi_class （优化算法选择参数）

这里我们开启 class_weight 为 balanced 默认为 None 得到测试结果为：

```
              precision    recall  f1-score   support

         neg       0.86      0.87      0.87     12500
         pos       0.87      0.86      0.87     12500

   micro avg       0.87      0.87      0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000

[[10933  1567]
 [ 1765 10735]]

Process finished with exit code 0
```

恩，貌似没太大变化，想了下因为分类问题是二元分类，neg or pos 而且样本数量一致，所以 balanced 之后权重事平均的，没有变化。由于参数较多，我们使用自动调参数来选出最优的结果。

自动调参

使用GridSearchCV

首先我们手动设定一组参数候选值，然后网格搜索会穷举各种参数组合，根据设定的评分机制找到最好的那一组设置。 我们设定了penalty C solver multi_class 等作为输入参数来穷举，具体代码如下：

```
tuned_parameters = [{'penalty': ['l1', 'l2'],
                     'C': [0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
                     'solver': ['liblinear'],
                     'multi_class': ['ovr']},
                    {'penalty': ['l2'],
                     'C': [0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100],
                     'solver': ['lbfgs'],
                     'multi_class': ['ovr', 'multinomial']}]

classifier = GridSearchCV(LogisticRegression(tol=1e-6), tuned_parameters, cv=10, n_jobs=-1)

classifier.fit(x_train, train_movie_data.target)

print('Best parameters set found:', classifier.best_params_)

predicted = classifier.predict(x_test)
print(classification_report(test_movie_data.target, predicted, target_names=train_movie_data.target_names))
print(metrics.confusion_matrix(test_movie_data.target, predicted))
```

得到最好的参数选择 C: 0.05 multi_class: ovr penalty: l2 solver: liblinear 成功率 88% 比之前略好

```
Best parameters set found: {'C': 0.05, 'multi_class': 'ovr', 'penalty': 'l2', 'solver': 'liblinear'}
              precision    recall  f1-score   support

         neg       0.88      0.88      0.88     12500
         pos       0.88      0.88      0.88     12500

   micro avg       0.88      0.88      0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

使用RandomizedSearchCV

```
tuned_parameters = {'C': uniform(loc=0, scale=4),
                    'multi_class': ['ovr', 'multinomial']}

classifier = RandomizedSearchCV(LogisticRegression(penalty='l2', solver='lbfgs', tol=1e-6),
                        tuned_parameters, cv=10, scoring='accuracy', n_iter=30)
```

结果如下：

```
Best parameters set found: {'C': 0.07328269490945383, 'multi_class': 'ovr'}
             precision    recall  f1-score   support

        neg       0.88      0.88      0.88     12500
        pos       0.88      0.88      0.88     12500

   micro avg       0.88      0.88      0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000

[[11037  1463]
 [ 1541 10959]]

Process finished with exit code 0
```

LogisticRegressionCV

LogisticRegressionCV 和LogisticRegression 不同主要在于正则化系数 C，
LogisticRegressionCV 使用了交叉验证来选择正则化系数C，相当于自动给出最优参数：

代码：

```
logistic_cv = LogisticRegressionCV(cv=5, max_iter=100, n_jobs=-1)
```

传入 classifier_report 就行,开始训练。

```
<class 'sklearn.linear_model.logistic.LogisticRegressionCV'>
             precision    recall  f1-score   support

        neg       0.91      0.91      0.91     12500
        pos       0.91      0.91      0.91     12500

   micro avg       0.91      0.91      0.91     25000
   macro avg       0.91      0.91      0.91     25000
weighted avg       0.91      0.91      0.91     25000

Time usage: 561.0744822025299
```

可以看到训练效果的结果成功率**提升到 91%**，整个训练过程花费了大约 9分钟。如果继续提高cv 和 max_iter 效果可能更高。

思考和改进方案

通过实验进行几种方案的对比

在不刻意调参数的情况下z，贝叶斯 MultinomialNB、决策树 DecisionTreeClassifier、随机森林RandomForestClassifier和逻辑回归 DecisionTreeClassifier 这四种中，对于二维分类问题，整体来说 决策树准确率最差，逻辑回归最好，通过几次调参，逻辑回归在验证 25000 条数据**最高准确率达到 91%** 。

Precision：逻辑回归> 贝叶斯> 随机森林 > 决策树
从时间维度上来说 ：贝叶斯>逻辑回归>随机森林>决策树

在一定条件下朴素贝叶斯的准确率甚至高于逻辑回归。

贝叶斯：
学习和预测的效率高，且易于实现，对于概率独立事件的分类问题有较好的处理。

逻辑回归：
逻辑回归的准确率和效率都比较高。 而且逻辑回归有很多调整模型的方式，在调好参数的情况下可以得到很高的准确率。而且比较方便观测样本概率分部。

决策树、随机森林：
结合之前的几次实验数据来看，决策树可能更适合小数据集，计算复杂度低，数据大精度不高，但是直观上来说比较快速容易的解释和说明一类数据的属性。

可改进方案

1. 对于不同的数据集选择最适合的算法。
2. 对于参数调优方面，可以使用GridSearchCV RandomSearchCV 等自动调优，匹配出最好的参数，但是可能耗时较长
3. 优化前期的文本特征也比较重要，可以提高准确率。