实验报告

实验介绍

实验背景

双相障碍属于心境障碍的一种疾病，英文名称为 Bipolar Disorder（BD），表示既有躁狂发作又有抑郁发作的一类疾病。目前病因未明，主要是生物、心理与社会环境诸多方面因素参与其发病过程。当前研究发现，在双相障碍发生过程中遗传因素、环境或应激因素之间的交互作用、以及交互作用的出现时间点等都产生重要的影响；临床表现按照发作特点可以分为抑郁发作、躁狂发作或混合发作。

双相障碍检测，即通过医学检测数据预测病人是否双相障碍，或双相障碍治疗是否有效。医学数据包括医学影像数据与肠道数据。由于缺少医学样本且特征过多，因此选取合适的特征对双模态特征进行整合并训练合适的分类器进行模型预测具有较强的现实需求与医学意义。本实验需要完成少样本、多特征下的监督学习。

实验要求

a) 实现双模态特征选择与提取整合。

b) 选择并训练机器学习模型进行准确分类。

c) 分析不同超参数以及特征选择方法对模型的结果影响。

数据集分析

医疗数据集存放在 `DataSet.xlsx` 中，共包括 39 个样本和 3 张表，表 `Feature1` 为医学影像特征，表 `Feature2` 为肠道特征，表 `label` 为样本类标。

导入数据并查看样本特征：

```
#导入医疗数据
data_xls = pd.ExcelFile('DataSet.xlsx')
data={}

#查看数据名称与大小
for name in data_xls.sheet_names:
    df = data_xls.parse(sheet_name=name,header=None)
    print("%-8s 表的 shape:"%name,df.shape)
    data[name] = df
```

输出：

```
Feature1 表的 shape: (39, 6670)
Feature2 表的 shape: (39, 377)
label    表的 shape: (39, 1)
```

可以看到，医疗数据中的样本和特征数量存在着极大的**不平衡。**其中医疗影像数据共 6670 维，肠道数据共 377 维，而样本仅有 39 个，其中正样本标签为 1 ，负样本标签为 -1 。

因此，**特征的筛选和组合以及机器学习模型的选择优化**对提高模型的性能极其重要。

数据预处理

特征归一化

使用Min-Max将特征归一化到 [0 , 1] 区间内

```
# 初始化一个 scaler，并将它施加到特征上
scaler = MinMaxScaler()
feature1 = pd.DataFrame(scaler.fit_transform(feature1_raw))
feature2 = pd.DataFrame(scaler.fit_transform(feature2_raw))
```

PCA降维

由于特征数维度较高(6670 + 377)，因此可以进行特征降维，这里分别对两类不同特征采用PCA方法，保留0.9的方差信息（测试的样本较少，此步骤可以考虑省略）。

```
pca = PCA(n_components=0.9)
feature1 = pca.fit_transform(feature1)
print("Feature1 降维后的维度:", pca.n_components_)

pca = PCA(n_components=0.9)  # 重新初始化PCA，以避免信息泄漏
feature2 = pca.fit_transform(feature2)
print("Feature2 降维后的维度:", pca.n_components_)
```

输出：

```
2024-11-17 16:49:18.644800 Feature1 降维后的维度: 30
2024-11-17 16:49:18.648700 Feature2 降维后的维度: 27
```

特征筛选与融合

根据统计特征值和label的皮尔孙相关系数 对两类特征分别进行排序筛选特征并融合

```
# 根据相关系数进行筛选
select_feature1 = SelectKBest(lambda X, Y:
tuple(map(tuple,np.array(list(map(lambda x:pearsonr(x, Y), X.T))).T)),
k=12
).fit(feature1, np.array(label).flatten()).get_support(indices=True)

select_feature2 = SelectKBest(lambda X, Y:
tuple(map(tuple,np.array(list(map(lambda x:pearsonr(x, Y), X.T))).T)),
k=10
).fit(feature2, np.array(label).flatten()).get_support(indices=True)

# 查看排序后特征
print("select feature1 name:", select_feature1)
print("select feature2 name:", select_feature2)

# 双模态特征选择并融合
new_features = pd.concat([feature1[feature1.columns.values[select_feature1]],
```

```
feature2[feature2.columns.values[select_feature2]]],axis=1)
```

数据切分

将数据集切分为训练数据、验证数据、测试数据

```
def data_split(features,labels):
    X_train, X_val, X_test,y_train, y_val, y_test=None, None,None, None, None,
None

    # 将 features 和 label 数据切分成训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2, random_state=0, stratify=labels)

    # 将 X_train 和 y_train 进一步切分为训练集和验证集
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=0, stratify=y_train)

    return X_train, X_val, X_test,y_train, y_val, y_test
```

模型选择

这里选用了四种常见的监督学习模型进行训练并对比效果

- 朴素贝叶斯
- 支持向量机
- 决策树
- 随机森林

```
def train_predict(learner, X_train, y_train, X_val, y_val):
    results = {}

    # 使用训练集数据来拟合学习器
    start = time()  # 获得程序开始时间
    learner = learner.fit(X_train, y_train)
    end = time()  # 获得程序结束时间

    # 计算训练时间
    # results['train_time'] = end - start

    # 得到在验证集上的预测值
    start = time()  # 获得程序开始时间
    predictions_val = learner.predict(X_val)
    predictions_train = learner.predict(X_train)
    end = time()  # 获得程序结束时间

    # 计算预测用时
    # results['pred_time'] = end - start
    \\其它指标的写入省略
```

```
# 初始化三个模型
clf_A = tree.DecisionTreeClassifier(random_state=42)
clf_B = naive_bayes.GaussianNB()
clf_C = svm.SVC()
clf_D = RandomForestClassifier(random_state=42)

data_path = "DataSet.xlsx"  # 数据集路径
new_features,label = data_processing_and_feature_selecting(data_path)
#数据划分
X_train, X_val, X_test,y_train, y_val, y_test = data_split(new_features,label)

# 收集学习器的结果
results = {}
for clf in [clf_A, clf_B, clf_C, clf_D]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    results[clf_name] = train_predict(clf, X_train, y_train, X_val, y_val)

# 打印三个模型得到的训练验证结果
print("高斯朴素贝叶斯模型结果:", results['GaussianNB'])
print("支持向量机模型结果:", results['SVC'])
print("决策树模型结果:", results['DecisionTreeClassifier'])
print("随机森林分类模型结果:", results['RandomForestClassifier'])
```

输出：

```
高斯朴素贝叶斯模型结果: {'acc_train': 0.9167, 'acc_val': 0.7143, 'recall_train':
0.8182, 'recall_val': 0.6667, 'f_train': 0.9, 'f_val': 0.6667}
支持向量机模型结果: {'acc_train': 1.0, 'acc_val': 1.0, 'recall_train': 1.0,
'recall_val': 1.0, 'f_train': 1.0, 'f_val': 1.0}
决策树模型结果: {'acc_train': 1.0, 'acc_val': 0.8571, 'recall_train': 1.0,
'recall_val': 1.0, 'f_train': 1.0, 'f_val': 0.7895}
随机森林分类模型结果: {'acc_train': 1.0, 'acc_val': 1.0, 'recall_train': 1.0,
'recall_val': 1.0, 'f_train': 1.0, 'f_val': 1.0}
```

	朴素贝叶斯	**SVC**	**决策树**	**随机森林**
准确率	71.43%	100%	85.71%	100%
F-score	0.6667	1.0	0.7895	1.0

可以看到，SVC和随机森林方法都取得了不错的效果

参数优化

创建、训练模型，并对参数进行优化并保存最佳模型

对于SVC，可以优化 `kernel` `gamma` `degree` `coef` 参数

对于随机森林，可以优化 `max_depth` `min_samples_split` `max_features` 参数

```
def search_model(X_train, y_train,X_val,y_val, model_save_path):
    """
```

```
    :param X_train, y_train: 训练集数据
    :param X_val,y_val: 验证集数据
    :param save_model_path: 保存模型的路径和名称
    :return:
    """
    # ---------------------- 实现模型创建、训练、优化和保存等部分的代码 ------------------
----
    #创建监督学习模型 （SVC和随机森林）

    # clf = RandomForestClassifier(random_state=42,bootstrap=True)
    # parameters = {
    #     'max_depth': [5,6,7,None],
    #     'min_samples_split': range(2, 6),
    #     'max_features': ['auto', 'sqrt'],
    # }

    clf=svm.SVC()
    parameters = {
    'kernel': ['rbf', 'poly'],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4],
    'coef0': [0.0, 0.5, 1]
    }

    # 创建一个fbeta_score打分对象 以F-score为例
    scorer = make_scorer(fbeta_score, beta=1)

    # 在分类器上使用网格搜索，使用'scorer'作为评价函数
    kfold = KFold(n_splits=10) #切割成十份

    # 同时传入交叉验证函数
    grid_obj = GridSearchCV(clf, parameters, scorer, cv=kfold)

    #绘制学习曲线
    plot_learning_curve(clf, X_train, y_train, cv=kfold, n_jobs=4)

    # 用训练数据拟合网格搜索对象并找到最佳参数
    grid_obj.fit(X_train, y_train)

    # 得到estimator并保存
    best_clf = grid_obj.best_estimator_
    joblib.dump(best_clf, model_save_path)

    # 使用没有调优的模型做预测
    predictions = (clf.fit(X_train, y_train)).predict(X_val)
    best_predictions = best_clf.predict(X_val)

    # 调优后的模型
    print ("best_clf\n------")
    print (best_clf)

    # 汇报调参前和调参后的分数
    print("\nUnoptimized model\n------")
    print("Accuracy score on validation data:
{:.4f}".format(accuracy_score(y_val, predictions)))
```

```
    print("Recall score on validation data: {:.4f}".format(recall_score(y_val,
predictions)))
    print("F-score on validation data: {:.4f}".format(fbeta_score(y_val,
predictions, beta = 1)))
    print("\nOptimized Model\n------")
    print("Final accuracy score on the validation data:
{:.4f}".format(accuracy_score(y_val, best_predictions)))
    print("Recall score on validation data: {:.4f}".format(recall_score(y_val,
best_predictions)))
    print("Final F-score on the validation data:
{:.4f}\n".format(fbeta_score(y_val, best_predictions, beta = 1)))
```

测试结果

1. SVC

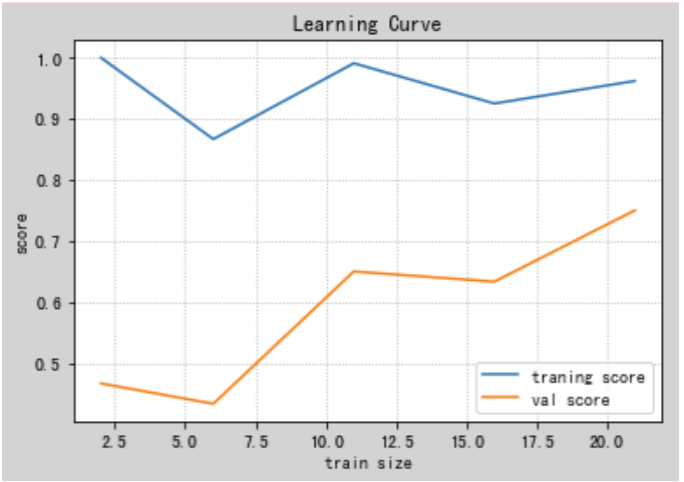
输出:

```
best_clf
------
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=1,
decision_function_shape='ovr', degree=2, gamma='scale', kernel='poly',
max_iter=-1, probability=False, random_state=None, shrinking=True,tol=0.001,
verbose=False)

Unoptimized model
------
Accuracy score on validation data: 1.0000
Recall score on validation data: 1.0000
F-score on validation data: 1.0000

Optimized Model
------
Final accuracy score on the validation data: 1.0000
Recall score on validation data: 1.0000
Final F-score on the validation data: 1.0000

Accuracy on test data: 0.8750
Recall on test data: 0.6667
F-score on test data: 0.9091
```



2. 随机森林

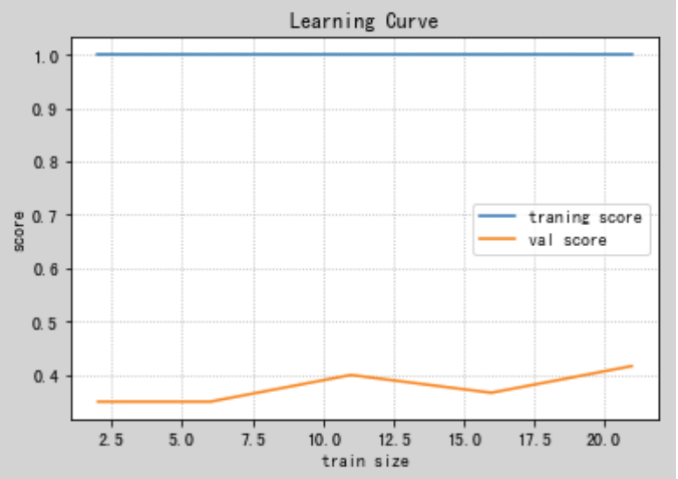
输出

```
best_clf
------
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
criterion='gini', max_depth=5, max_features='auto',max_leaf_nodes=None,
max_samples=None, in_impurity_decrease=0.0,
min_impurity_split=None,min_samples_leaf=1, min_samples_split=4,
min_weight_fraction_leaf=0.0, n_estimators=100,n_jobs=None,
oob_score=False,random_state=42, verbose=0, warm_start=False)

Unoptimized model
------
Accuracy score on validation data: 1.0000
Recall score on validation data: 1.0000
F-score on validation data: 1.0000

Optimized Model
------
Final accuracy score on the validation data: 1.0000
Recall score on validation data: 1.0000
Final F-score on the validation data: 1.0000

Accuracy on test data: 0.7500
Recall on test data: 0.3333
F-score on test data: 0.7143
```



SVC的效果明显由于随机森林。因此，选用SVC作为最终模型，且应用搜索得到的最佳参数。

加载模型并预测结果

```
# 加载模型
model = joblib.load(model_path)

def predict(new_features):

    # 获取输入图片的类别
    y_predict = model.predict(new_features)

    # 返回图片的类别
    return y_predict
```

上传至MO平台进行验证，效果良好：