

模型标准化

1. 数据输入输出
 - 1.1. 配置数据库登录信息
 - 1.2. 数据读取
 - 1.3. 数据导入
2. 数据探索
 - 2.1. 实例化
 - 2.2. 输出数值型数据的统计信息
 - 2.3. 输出非数值型数据的统计信息
 - 2.4. 输出变量相关性热力图
 - 2.5. 输出各变量的分布图
 - 2.6. 输出变量关系图
3. 数据预处理
 - 3.1. 实例化
 - 3.2. 数据无量纲化
 - 3.3. 空值填充
 - 3.4. 数据编码
 - 3.5. 对数转换
4. 特征选择
 - 4.1. 实例化
 - 4.2. 基于统计特征选择
 - 4.2.1. 方差筛选
 - 4.2.2. 过滤法
 - 4.3. 基于模型特征选择
 - 4.3.1. 包装法
 - 4.3.2. 嵌入法
5. 模型训练
 - 5.1. 实例化
 - 5.2. XGBoost模型
 - 5.3. LightGBM模型
6. 模型评估
 - 6.1. 单模型评估
 - 6.1.1. 实例化
 - 6.1.2. 评估分数
 - 6.1.3. 评估指标可视化
 - 6.2. 多模型评估对比
 - 6.2.1. 实例化
 - 6.2.2. 评估分数对比
 - 6.2.3. 评估指标可视化对比
7. 模型保存和加载
 - 7.1. 实例化
 - 7.2. 模型保存
 - 7.3. 模型加载
8. 预测数据
 - 8.1. 实例化
 - 8.2. 数据预处理
 - 8.3. 特征选择
 - 8.4. 模型预测
9. 日志生成
 - 9.1. 实例化
 - 9.2. 日志输出到控制台和写入日志文件

模型标准化

1. 数据输入输出

data_io.py

1.1. 配置数据库登录信息

- 先在config.ini配置文件上将数据库的登录信息配置完成，示例如下：

```
1 [server]
2 user = dbc
3 password = pass
4 host = 192.168.253.231
5 port = 22
6 dbname = server1
```

- 将配置文件的地址传入data_io.database类并实例化后可实现数据输入输出

path: 配置文件的地址，参数类型为string

```
1 from data_io import database
2 path = r'D:\troywu666\business_stuff\民生对公项目\模型标准化\config.ini'
3 database = Database(path)
```

1.2. 数据读取

- 输入sql语句和数据库类型可实现数据读取

sql: 所使用的sql查询语句，参数类型为string

server: 数据库类型，可选的有 'teradata' 和 'mysql' 两种

```
1 sql = 'SELECT empmo, ename, mgr FROM emp'
2 database.read_data(sql = sql, server = 'teradata')
```

1.3. 数据导入

- 在使用Database(path).data_to_sql函数时传入数据和数据库类型可实现数据读取

data: 需要储存到数据库的数据，类型为pandas.DataFrame

server: 数据库类型，可选的有 'teradata' 和 'mysql' 两种

```
1 database.data_to_sql(data = data, server = 'teradata')
```

2. 数据探索

2.1. 实例化

- 传入需要探索的数据进行并实例化

data: 需要探索的数据, 类型为pandas.DataFrame

```
1 from data_exploring import Explore
2 explore = Explore(data)
```

2.2. 输出数值型数据的统计信息

- 统计数据的 '0%', '12.5%', '25%', '37.5%', '50%', '62.5%', '75%', '87.5%', '95%', '99%', '100%' 分位数、数据个数、偏度、峰度、平均值、标准差、空值个数、空值比例

```
1 explore_frame = explore.describe_num()
```

2.3. 输出非数值型数据的统计信息

- 统计数据的非空值个数、出现频率最高的数据、出现频率最高的数据、唯一值个数、空值个数、空值比例

```
1 explore_frame = explore.describe_obj()
```

2.4. 输出变量相关性热力图

figsize: 设定图像的大小, 参数类型为tuple

```
1 explore.corr_and_plot(figsize = (32, 18))
```

2.5. 输出各变量的分布图

```
1 explore.distplot()
```

2.6. 输出变量关系图

- 该图展现变量两两之间的关系 (线性或非线性, 有无较明显的相关关系)

vars: 想要研究的变量, 数据类型为list

hue: 使用指定变量为分类变量画图, 数据类型是string

```
1 explore.pairplot(vars = df.columns[:-1], hue = 'age')
```

3. 数据预处理

preprocessing.py

3.1. 实例化

- 传入需要预处理的数据进行并实例化

data: 需要处理的数据, 类型为pandas.DataFrame

```
1 from preprocessing import Transformer
2
3 process = Transformer(data)
```

3.2. 数据无量纲化

method: 无量纲化的策略, 有 'min_max' 和 'standard' 两种取值, 'min_max' 为归一化, 'standard' 为标准化的

返回结果: 已训练好的数据转化器 (方便后续储存及对测试集数据进行转化) 和已无量纲化的训练数据

```
1 transformer, tranformed_data = process.scaler(method = 'min_max')
```

3.3. 空值填充

- 可通过设定按不同策略进行填充, 如均值、中位数、众数和设定值

strategy: 填充策略, 可选 'mean', 'median', 'most_frequent', 'constant'

fill_value: 当strategy选为 'constant' 时, fill_value需要额外设定, 默认值是None

返回结果: 已训练好的数据转化器 (方便后续储存及对测试集数据进行转化) 和已填充空值的训练数据

```
1 strategy = 'constant'
2 fill_value = 9
3 transformer, tranformed_data = process.fillna(strategy = strategy, \
4                                              fill_value = fill_value)
```

3.4. 数据编码

- 可将分类型和连续型数据进行编码

method: 数据编码的方法, 有 'onehot', 'ordinal' 和 'binarizer' 三种可填方式, 参数类型为 string

'onehot' 使用于无序分类变量进行独热编码

'ordinal' 使用于**有序**分类变量进行编码

'binarizer' 使用于将**连续型**变量进行二值化

threshold: 当method选为'binarizer'时, threshold为二值化的阈值。默认值为None, 参数类型为float

返回结果: 已训练好的数据转化器 (方便后续储存及对测试集数据进行转化) 和已编码的训练数据

```
1 method= 'binarizer'
2 threshold = 3
3 transformer, tranformed_data = process.encoder(method = method, \
4                                              threshold = threshold)
```

3.5. 对数转换

- 使特征数据经过对数族转换, 使其转换后的分布为最接近正态分布

lmbda: 需要转换时使用的lambda值

返回结果: 已训练好的数据转化器 (方便后续储存及对测试集数据进行转化) 和已对数转换的训练数据

```
1 lmbda = None
2 transformer, tranformed_data = process.box_cox(lmbda = lmbda)
```

4. 特征选择

feature_selection.py

4.1. 实例化

- 传入需要筛选的数据并进行实例化

fea: 需要筛选的特征数据, 类型为pandas.DataFrame

```
1 from feature_selection import Selector
2 select = Selector(fea)
```

4.2. 基于统计特征选择

4.2.1. 方差筛选

- 通过特征本身的方差来对特征进行筛选

threshold: 方差的阈值, 表示将舍弃方差小于阈值的特征。默认值为0, 参数类型为float

返回结果: 已训练好的特征筛选器 (方便后续储存及对测试集数据进行特征筛选) 和已筛选特征的训练数据

```
1 selector, selected_data = select.variance(threshold = 0.1)
```

4.2.2. 过滤法

- 根据各种统计检验中的分数以及相关性的各项指标来选择特征

y: 训练数据的额label, 参数类型为维度是1的numpy.array

k: 选择需要保留的特征个数, 参数类型为int

method: 特征过滤的策略,

有 'chi2_filter'、'f_clas_filter'、'f_reg_filter'、'mutual_clas_filter' 和 'mutual_reg_filter' 五种可填方式, 参数类型为string

'chi2_filter': 适用于在分类问题中计算特征和标签的卡方统计量进行筛选

'f_clas_filter': 适用于在分类问题中计算特征和标签的F统计量进行筛选

'f_reg_filter': 适用于在回归问题中计算特征和标签的F统计量进行筛选

'mutual_clas_filter': 适用于在分类问题中计算特征和标签的互信息量进行筛选

'mutual_reg_filter': 适用于在回归问题中计算特征和标签的互信息量进行筛选

返回结果: 已训练好的特征筛选器 (方便后续储存及对测试集数据进行特征筛选) 和已筛选特征的训练数据

```
1 y = data.target
2 k = 100
3 method = 'chi2_filter'
4 selector, selected_data = select.filter(y = y, k = k, method = method)
```

4.3. 基于模型特征选择

4.3.1. 包装法

- 特征选择和算法训练同时进行, 反复迭代创建模型, 并在每次迭代时保留最佳特征或剔除最差特征, 下一次迭代时, 算法使用上一次建模中保留的特征来构建下一个模型

estimator: 用于特征筛选的算法, **该算法为sklearn中的算法**, 具有 feature_importances_ 或 coef_ 属性

y: 训练数据的额label, 参数类型为维度是1的numpy.array

n: 选择需要保留的特征个数, 参数类型为int

step: 每次迭代中希望移除的特征个数, 参数类型为int

返回结果: 已训练好的特征筛选器 (方便后续储存及对测试集数据进行特征筛选) 和已筛选特征的训练数据

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 y = data.target
4 n= 100
5 step = 20
6 selector, selected_data = select.wrapper(RandomForestClassifier(), \
7                                           y = y, n = n, step = step)

```

4.3.2. 嵌入法

- 特征选择和算法训练同时进行，得到各个特征的权值系数，根据权值系数从大到小选择特征

estimator: 用于特征筛选的算法，**该算法为sklearn中的算法**，具有 `feature_importances_` 或 `coef_` 属性

y: 训练数据的标签，参数类型为维度是1的numpy.array

threshold: 权值系数的阈值，参数类型为float

返回结果: 已训练好的特征筛选器（方便后续储存及对测试集数据进行特征筛选）和已筛选特征的训练数据

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 y = data.target
4 threshold = 0.5
5 selector, selected_data = select.embedded(RandomForestClassifier(), \
6                                           y = y, threshold = threshold)

```

5. 模型训练

train_val.py

5.1. 实例化

- 传入需要训练数据和验证数据占训练数据的比例，并进行实例化

train: 训练数据的特征，参数类型为pandas.DataFrame

target: 训练数据的标签，参数类型为维度是1的numpy.array

test_size: 验证数据占训练数据的比例，参数类型为float

```

1 from train_val import Model_training
2
3 training = Model_training(train, target, test_size = 0.3)

```

5.2. XGBoost模型

num_boost_round: 表示boosting的迭代数量，参数类型为int

early_stopping_rounds: 早停参数，参数类型为int

params: 包含训练参数的字典

返回结果：已训练好的算法模型（方便后续储存及对测试集数据进行预测）、验证集标签值和验证集预测值

```
1 num_boost_round = 1000
2 early_stopping_rounds = 100
3 params = {'booster': 'gbtree',
4           'objective': 'binary:logistic',
5           'mat_depth': 3,
6           'subsample': 0.7,
7           'colsample_bytree': 0.7,
8           'eta': 0.1,
9           'lambda': 0.1,
10          'eval_metric': 'auc',
11          'scale_pos_weight': 0.1}
12 model_xgb, y_true, y_pred = training.xgb_model(num_boost_round =
num_boost_round,
13                                              early_stopping_rounds =
early_stopping_rounds,
14                                              params = params)
```

5.3. LightGBM模型

num_boost_round: 表示boosting的 迭代数量，参数类型为int

early_stopping_rounds: 早停参数，参数类型为int

params: 包含训练参数的字典

categorical_feature: 为属于多分类的特征名列表。默认值为 'auto'，需要强调特征名时参数类型为list

返回结果：已训练好的算法模型（方便后续储存及对测试集数据进行预测）、验证集标签值和验证集预测值

```
1 num_boost_round = 1000
2 early_stopping_rounds = 100
3 params = {'boosting': 'gbdt',
4           'objective': 'binary',
5           'mat_depth': 3,
6           'feature_fraction': 0.7,
7           '#is_unbalance': True,
8           'eta': 0.1,
9           'lambda_l1': 0.1,
10          'metric': 'auc',
11          'scale_pos_weight': 0.1}
12 model_lgb, y_true, y_pred = training.lgb_model(num_boost_round, \
13                                              early_stopping_rounds, \
14                                              params = params)
```

6. 模型评估

evaluation.py

6.1. 单模型评估

6.1.1. 实例化

- 传入验证集数据的标签以及预测值

y_true: 验证集标签, 数据类型为维度是1的numpy.array

y_pred: 验证集的预测值, 数据类型为维度是1的numpy.array

```
1 from evaluation import Metrics
2
3 metrics = Metrics(y_true, y_pred)
```

6.1.2. 评估分数

- 输出 准确率、召回率、精确率、AUC分数、f1分数

```
1 accuracy_score, recall_score, \
2 precision_score, roc_auc_score, f1_score = metrics.eval_score()
```

6.1.3. 评估指标可视化

- 输出ROC曲线、混淆矩阵、KS曲线、PR图、提升曲线、累计增益曲线、概率校准曲线

```
1 metrics.eval_plot()
```

6.2. 多模型评估对比

6.2.1. 实例化

- 传入验证集数据的标签以及预测值

y_true: 验证集标签, 参数类型为维度是1的numpy.array

y_pred: 不同模型下验证集的预测值, 数据类型为dictionary, 其中dictionary的keys的参数类型为 string, values的参数类型为维度是1的numpy.array, 如 {'xgb':

y_pred_xgb, 'lgb': y_pred_lgb}

```
1 from evaluation import Metrics_comparison
2
3 metrics_comparison = Metrics_comparison(y_true, y_pred)
```

6.2.2. 评估分数对比

- 输出 准确率、召回率、精确率、AUC分数、f1分数

```
1 compare_frame = metrics_comparison.compare_score()
```

6.2.3. 评估指标可视化对比

- 输出ROC曲线、混淆矩阵、KS曲线、PR图、提升曲线、累计增益曲线、概率校准曲线的对比图

```
1 | metrics_comparison.compare_plot()
```

7. 模型保存和加载

model_io.py

7.1. 实例化

```
1 | from model_io import Model_pickle
2 |
3 | model_pickle = Model_pickle()
```

7.2. 模型保存

model: 需要保存的模型

path: 模型存放的路径, 参数类型为string

model_name: 模型的命名, 参数类型为string

```
1 | model = model_lgb
2 | path = r'D:\troywu666\business_stuff\民生对公项目\模型标准化
3 | model_name = 'model_lgb'
4 | model_pickle.dump(model = model, path = path, model_name = model_name)
```

7.3. 模型加载

path: 模型存放的路径, 参数类型为string

model_name: 已保存模型的命名, 参数类型为string

```
1 | path = r'D:\troywu666\business_stuff\民生对公项目\模型标准化
2 | model_name = 'model_xgb'
3 | model_pickle.load(mpath = path, model_name = model_name)
```

8. 预测数据

predict.py

8.1. 实例化

```
1 from predict import Prediction
2
3 predictor = Prediction()
```

8.2. 数据预处理

- 基于**已训练好的**数据处理器对测试数据进行数据预处理

transformer: 已训练好的数据处理器

test: 测试集数据, 参数类型为pandas.DataFrame

```
1 transformed_data = predictor.data_preprocess(transformer, test)
```

8.3. 特征选择

- 基于**已训练好的**特征选择器对测试数据进行特征筛选

selector: 已训练好的特征选择器

test: 测试集数据, 参数类型为pandas.DataFrame

```
1 selected_data = predictor.feature_select(selector, test)
```

8.4. 模型预测

- 基于**已训练好的**模型对测试数据进行预测

model: 已训练好的模型

test: 测试集数据, 参数类型为pandas.DataFrame

model_cate: 模型的类别, 可选类别有 'xgb' 和 'lgb'

pred_leaf: 是否输出叶子节点

```
1 y_pred = predictor.model_predict(model, test, model_cate = 'xgb', \
2                                   pred_leaf = False)
```

9. 日志生成

print_in_log.py

9.1. 实例化

- 初始化日志生成器, 生成器可以同时将日志输出到控制台和写入日志文件

path: 日志存放的位置, 参数类型为string

logfile_name: 日志的命名, 参数类型为string

```
1 from print_in_log import Save_log
2
3 path = r'D:/troywu666/business_stuff/民生对公项目/模型标准化'
4 logfile_name = 'test'
5 save_log = Save_log(path = path, logfile_name = logfile_name)
```

9.2. 日志输出到控制台和写入日志文件

statement: 日志内容, 参数类型为string

```
1 save_log.logging(statement = 'Predict complete.')
```