Parallel Attempt 1

October 25, 2023

[ ]:
```
import pandas as pd # standard
import numpy as np # standard
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score # for accuracy calculation
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import roc_auc_score

from joblib import Parallel, delayed

import matplotlib.pyplot as plt
import seaborn as sns

import thermogram_utilities

import warnings
warnings.filterwarnings("ignore")
```

[ ]:
```
df = pd.read_excel("/Users/avery/OneDrive/Documents/GitHub/
 ↪Clinical_TLB_2023-2024/lung_cancer_tlb.xlsx")

# replace NA with control
df['CancerType'] = np.where(df['CancerType'].isna(), 'Control',␣
 ↪df['CancerType'])

# get location of cut off values
lower_column_index = df.columns.get_loc("T51")
upper_column_index = df.columns.get_loc("T83.1")
label_column_index = df.columns.get_loc("CancerType")

column_indices = np.arange(lower_column_index, upper_column_index)
column_indices = np.append(column_indices, 0)
column_indices = np.append(column_indices, 1)



column_indices = np.append(column_indices, label_column_index)
```

```
df = df.iloc[:, column_indices]

# keep only Control and Adenocarcinoma for analysis
df_tree = df[(df['CancerType'] == 'Control') | (df['CancerType'] ==␣
↪'Adenocarcinoma')]
df_tree = df_tree.reset_index(drop=True)

df_tree1 = df_tree.rename(columns={'CancerType': 'Response'})

df_tree1 = df_tree1.drop(["sample_id", 'pub_id'], axis = 1)
```

[ ]:
```
def bootstrap_cv_sets(df):

    num_rows = df.shape[0]
    all_vals = np.arange(0, num_rows)
    train_indices = np.random.choice(num_rows, num_rows, replace = True)
    test_indices = np.setdiff1d(all_vals, train_indices)
    train_set = df.iloc[train_indices, : ]
    test_set = df.iloc[test_indices, : ]

    return train_set, test_set

def train_and_test(hyperparameters, train_set, test_set):
    rf = RandomForestClassifier(**hyperparameters)    # Initialize forest with␣
↪specified parameters

    train_set_labels = train_set['Response']
    train_set = train_set.drop("Response", axis=1)

    rf = rf.fit(train_set, train_set_labels)     # Train the classifier
    test_preds = rf.predict(test_set.drop("Response", axis = 1))                ␣
↪ # Make predictions on the test set
    balanced_acc = balanced_accuracy_score(test_set['Response'], test_preds)  #␣
↪Calculate balanced accuracy
    return balanced_acc
```

[ ]:
```
this, model = train_and_test(hyperparameter_combinations[17], train_set,␣
↪test_set)

model.get_params
```

[ ]:
```
<bound method BaseEstimator.get_params of RandomForestClassifier(max_depth=61,
max_features='log2', n_estimators=500)>
```

[ ]:
```
hyperparameter_combinations[17]
```

[ ]:
```
{'n_estimators': 500, 'max_depth': 61, 'max_features': 'log2'}
```

[ ]:
```
N = df_tree1.shape[0]

train_set , test_set = bootstrap_cv_sets(df_tree1)

hyperparameter_combinations = [
    {'n_estimators': 100, 'max_depth': None, 'max_features': None},
    {'n_estimators': 100, 'max_depth': N // 2, 'max_features': None},
    {'n_estimators': 100, 'max_depth': None, 'max_features': 'sqrt'},
    {'n_estimators': 100, 'max_depth': N // 2, 'max_features': 'sqrt'},
    {'n_estimators': 100, 'max_depth': None, 'max_features': 'log2'},
    {'n_estimators': 100, 'max_depth': N // 2, 'max_features': 'log2'},
    {'n_estimators': 250, 'max_depth': None, 'max_features': None},
    {'n_estimators': 250, 'max_depth': N // 2, 'max_features': None},
    {'n_estimators': 250, 'max_depth': None, 'max_features': 'sqrt'},
    {'n_estimators': 250, 'max_depth': N // 2, 'max_features': 'sqrt'},
    {'n_estimators': 250, 'max_depth': None, 'max_features': 'log2'},
    {'n_estimators': 250, 'max_depth': N // 2, 'max_features': 'log2'},
    {'n_estimators': 500, 'max_depth': None, 'max_features': None},
    {'n_estimators': 500, 'max_depth': N // 2, 'max_features': None},
    {'n_estimators': 500, 'max_depth': None, 'max_features': 'sqrt'},
    {'n_estimators': 500, 'max_depth': N // 2, 'max_features': 'sqrt'},
    {'n_estimators': 500, 'max_depth': None, 'max_features': 'log2'},
    {'n_estimators': 500, 'max_depth': N // 2, 'max_features': 'log2'},
    {'n_estimators': 1000, 'max_depth': None, 'max_features': None},
    {'n_estimators': 1000, 'max_depth': N // 2, 'max_features': None},
    {'n_estimators': 1000, 'max_depth': None, 'max_features': 'sqrt'},
    {'n_estimators': 1000, 'max_depth': N // 2, 'max_features': 'sqrt'},
    {'n_estimators': 1000, 'max_depth': None, 'max_features': 'log2'},
    {'n_estimators': 1000, 'max_depth': N // 2, 'max_features': 'log2'}
]
```

[ ]:
```
results = Parallel(n_jobs=-1)(delayed(train_and_test)(params, train_set,␣
↪test_set) for params in hyperparameter_combinations)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done   4 out of  24 | elapsed:    0.5s remaining:    3.0s
[Parallel(n_jobs=-1)]: Done   7 out of  24 | elapsed:    1.4s remaining:    3.6s
[Parallel(n_jobs=-1)]: Done  10 out of  24 | elapsed:    2.4s remaining:    3.4s
[Parallel(n_jobs=-1)]: Done  13 out of  24 | elapsed:    3.6s remaining:    3.0s
[Parallel(n_jobs=-1)]: Done  16 out of  24 | elapsed:    5.4s remaining:    2.7s
[Parallel(n_jobs=-1)]: Done  19 out of  24 | elapsed:    8.2s remaining:    2.1s
[Parallel(n_jobs=-1)]: Done  22 out of  24 | elapsed:   13.9s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed:   24.4s finished
```

[ ]:
```
train, test = bootstrap_cv_sets(df_tree1)

train_and_test(hyperparameter_combinations[0], train, test)
```

```
[ ]: 0.6041666666666667
```

```
[ ]: hyperparameter_combinations[0]
```

```
[ ]: {'n_estimators': 100, 'max_depth': None, 'max_features': None}
```

```
[ ]: results = []
     for i in range(5):
         train_set , test_set = bootstrap_cv_sets(df_tree1)
         hyperparam_result = Parallel(n_jobs=-1)(delayed(train_and_test)(params,␣
      ↪train_set, test_set) for params in hyperparameter_combinations)
         results.append(hyperparam_result)
```

```
[ ]: '''this, model = train_and_test(hyperparameter_combinations[17], train_set,␣
      ↪test_set)

     model.get_params

     hyperparameter_combinations[17]'''
```