Documentation

Purpose: Identifies differentially expressed genes (DEGs) associated with overall survival (OS) in stage I and III breast cancer patients using Cox proportional hazards regression with L1/L2 regularization.

- Inputs:
  - clinical_data: DataFrame containing patient clinical information.
  - mrna_tpm: DataFrame containing RNA expression data.
  - Down_DEGs: DataFrame containing downregulated DEGs.
  - Up_DEGs: DataFrame containing upregulated DEGs.
- Outputs:
  - Prints coefficients for DEGs at different alpha values.
  - Plots coefficients vs. alpha values, highlighting important genes.
  - Prints best parameters and scores from grid search.
  - Prints time-dependent AUC for risk scores.

Step-by-Step Explanation:

1. Data Preparation:
   - Filters data for stage I/III patients with OS data.
   - Subsets DEGs from expression data.
   - Prepares survival data (time and event).
   - Encodes event data (likely using a label encoder, not shown).
   - Normalizes DEG expression data using standard scaling.
2. Cox Model Fitting and Analysis and Grid Search for Hyperparameter Tuning:
   - Fits CoxnetSurvivalAnalysis with L1 regularization (Lasso only/Initialization Step).

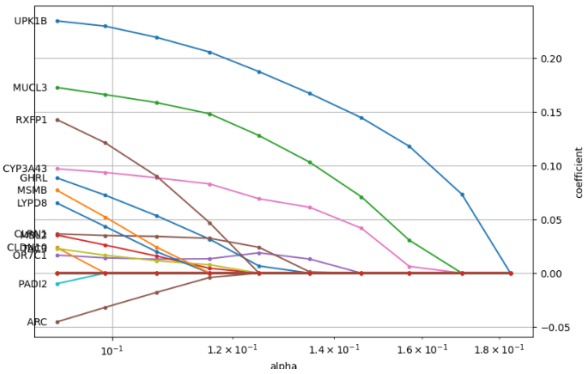
```
#Testing the range of alphas and their corresponding number of genes selected
estimator=CoxnetSurvivalAnalysis(n_alphas=10, l1_ratio=1, alpha_min_ratio=0.01 ,verbose = 10)
estimator.fit(X,Y)
```

   - Creates hazard coefficients for the DEGs at different alphas.
   - Prints non-zero coefficients per alpha, showing the number of genes that have non-zero coefficients.

```
coefficients_lasso=pd.DataFrame(estimator.coef_, index=X.columns, columns=np.round(estimator.alphas_, 5))
alphas=estimator.alphas_

for i, col in enumerate(coefficients_lasso.columns):
    non_zero = np.sum(coefficients_lasso[col] != 0)  # Count non-zeros in each column
    print(f"Column {i+1}: Number of non-zero coefficients: {non_zero}")
```

```
Column 1: Number of non-zero coefficients: 0
Column 2: Number of non-zero coefficients: 12
Column 3: Number of non-zero coefficients: 24
Column 4: Number of non-zero coefficients: 40
Column 5: Number of non-zero coefficients: 50
Column 6: Number of non-zero coefficients: 62
Column 7: Number of non-zero coefficients: 65
Column 8: Number of non-zero coefficients: 70
Column 9: Number of non-zero coefficients: 76
Column 10: Number of non-zero coefficients: 76
```

- Plots coefficients vs. alphas, highlighting important genes.



3. :

- Performs grid search with cross-validation to find the best alpha and l1_ratio.

```python
#Steps in elastic net parameter
l1_ratios = np.arange(1, -1, -0.05)
#number of cross fold validations (testing will be 20% and will repeat 5 times to cover all the samples)
cv = KFold(n_splits=5, shuffle=True, random_state=0)
gcv = GridSearchCV((CoxnetSurvivalAnalysis(alpha_min_ratio=1)),
    param_grid={"alphas": [[v] for v in alphas], "l1_ratio" : l1_ratios},
    cv=cv,
    error_score=0.5,
    n_jobs=-1, #to work on the GPU
).fit(X, Y)
cv_results = pd.DataFrame(gcv.cv_results_)

# Display the relevant information
print("Best Parameters:", gcv.best_params_)
print("Best Score:", gcv.best_score_)
print("Grid Search Results:")
print(cv_results[['params', 'mean_test_score', 'std_test_score']])
```

- Prints best parameters and scores.

```
Best Parameters: {'alphas': [0.18342394045167265], 'l1_ratio': 0.09999999999999992}
Best Score: 0.6462710583896871
Grid Search Results:
                                                params  mean_test_score  \
0    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.583829
1    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.569498
2    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.529025
3    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.509011
4    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.525941
..                                                 ...              ...
395  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
396  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
397  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
398  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
399  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
```

4. Risk Score Calculation:

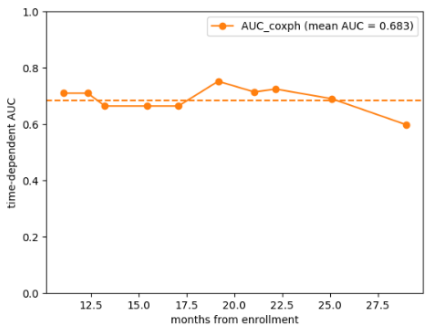
   - Selects genes with non-zero coefficients.
   - Calculates risk scores by multiplying gene expression with coefficients and summing.

```python
#Risk Score Sum(TPM(Gene) * Beta(Gene))

#Select columns of genes with non-zero coefficients
mrna_tpm_lasso = mrna_tpm_DEGs[non_zero_matrix.index]
#Ensure that they are in the same order
sum(non_zero_matrix.index == mrna_tpm_lasso.columns) == 97 #Should equal 97

#Multiply each gene expression values by its coefficient
for i in range(len(non_zero_matrix.index)):
    mrna_tpm_lasso.iloc[:,i] = mrna_tpm_lasso.iloc[:,i] * non_zero_matrix[i]

#Sum the row scores of each patient (total risk score)
risk_score= mrna_tpm_lasso.sum(axis=1)
```

   - Gene signature is the column of coefficients produced from the lassocox function and what is calculated for each patient is the hazard based on his/her signature

5. Time-Dependent AUC Evaluation:

   - Splits data into training and testing sets.
   - Calculates and plots time-dependent AUC for risk scores.



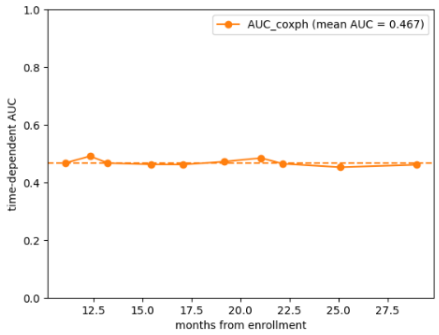
6. External Validation:

- Reads external validation data.
- Filters and normalizes expression data to match selected genes.

```
#You will find this file provided in the repository
external_validation=pd.read_csv("SurvivalMatrix62254.csv")
#filtering the expression matrix to include only the genes with nonzero coefficients, + applying standard scaler
external_validation_exp = external_validation.loc[:, external_validation.columns.isin(non_zero_matrix.index)]
s_scaler=StandardScaler()
stored_forcolumns=external_validation_exp
external_validation_exp=pd.DataFrame(s_scaler.fit_transform(external_validation_exp))
external_validation_exp.columns=stored_forcolumns.columns
#Making a seperate survival dataframe that will be later needed by the sksurv package
external_validation_surv = external_validation.iloc[:,0:3]
```

- Calculates risk scores for external data.
- Evaluates time-dependent AUC on external data. (Train/Test split was performed only for the function as it is usually used for train/test direct evaluation)

```
auc, mean_auc = cumulative_dynamic_auc(y_train, y_test, X_test, times)

plt.plot(times, auc, marker="o", color=f"C1", label=f"AUC_coxph (mean AUC = {mean_auc:.3f})")
plt.xlabel("months from enrollment")
plt.ylabel("time-dependent AUC")
plt.axhline(mean_auc, color=f"C1", linestyle="--")
plt.legend()
plt.ylim(0, 1)
```



Key Functions:

- plot_coefficients(coefs, n_highlight): Creates a plot of coefficients vs. alphas.
- cumulative_dynamic_auc(y_train, y_test, X_test, times): Calculates time-dependent AUC.
- CoxnetSurvivalAnalysis(alpha_min_ratio=1, l1_ratio=1)

Important Variables:

- clinical_data, mrna_tpm, Down_DEGs, Up_DEGs: Input DataFrames.
- survival_data: DataFrame containing survival data.
- X: DataFrame containing normalized DEG expression data.
- Y: Array containing survival data in records format.
- estimator: CoxnetSurvivalAnalysis estimator.
- coefficients_lasso: DataFrame of coefficients for DEGs at different alphas.

Additional Notes:

- File annotations that will be needed can be found in the readme file