Building Context-Specific Metabolic Models through Open source Integration of RNA-Seq Data for Epithelial and Mesenchymal States tailored for Fasting studies.*

Process of constructing CSMs for epithelial and mesenchymal states using the CORDA algorithm.

Karthik

2024-07-07

Abstract

Context-specific metabolic models (CSMs) are crucial for representing metabolic processes in particular biological conditions. This report outlines the methodology for constructing CSMs tailored to epithelial and mesenchymal states using the Open source CORDA algorithms, integrated with RNA-Seq data. We describe the steps taken to refine generic metabolic models, ensuring that they reflect the metabolic activities specific to the studied contexts. This approach provides a robust framework for building detailed and condition-specific models for further metabolic analysis.

```
from cobra import Configuration
from cobra.io import read_sbml_model
import pandas as pd
import cobra
from sklearn.preprocessing import MinMaxScaler
from cobra.manipulation.delete import remove_genes, prune_unused_metabolites, prune_unused_r
import seaborn as sns
import matplotlib.pyplot as plt
from corda import CORDA
```

I am setting to the solver an open source linear programming alternative: "GLPK" to solve the stoichiometric matrix

```
config = Configuration()
config.solver = "glpk"
config
```

**

```
solver: glpk
tolerance: 1e-07
lower_bound: -1000.0
upper_bound: 1000.0
processes: 7
cache_directory: /Users/karthik/Library/Caches/cobrapy
max_cache_size: 104857600
cache_expiration: None
```

Here, parent model will be Recon3D, from which context specific models: Epithelial and Mesenchymal CSMs are created.

```
recon3d_model = read_sbml_model('/Users/karthik/Desktop/PHCCO IISc Internship/EMT/Recon3D.xm
recon3d_model
```

```
<Model Recon3D at 0x17648b810>
```

Then I read the EMT expression data for Mesenchymal and Epithelial each, to get the CSMs of based of each of the columns

```
expression_data = pd.read_csv(
    # "/Users/karthik/Desktop/PHCCO IISc Internship/emt_expression_data_with_recon_id.csv"
    "/Users/karthik/Desktop/PHCCO IISc Internship/EMT_FINAL_DATA.csv"
)
expression_data.head()
```

```
    Gene_ID  Mesenchymal  Epithelial
0  2978_AT1     1.590799   -7.624877
1  1571_AT1     6.046029    3.480441
2  1549_AT1     2.358338    3.101581
3  1548_AT1     2.358338    3.101581
4   949_AT1     8.881953    8.613039
```

The following cell was created to read the fasting expression data, which was assigned to the same variable as we read above. This expression data is used on each of Epithelial and Mesenchymal CSMs, to create Epithelial-Fasting CSM and Mesenchymal-Fasting CSM.

```
expression_data = pd.read_csv('/Users/karthik/Desktop/PHCCO IISc Internship/EMT/processed_s1
expression_data
```

```
             symbol   baseMean  log2fold_change   p_value  \
0           DDX11L1   0.258591         0.036116  0.879453
1            WASH7P  12.983945         0.008918  0.970470
2         LOC729737  21.638194         0.006211  0.982015
3           FAM138D   0.114211         0.033687  0.800134
4      LOC101928626   0.250746         0.107664  0.588727
...             ...        ...              ...       ...
25535        BCORP1   0.075737         0.027772  0.834668
```

```
25536        TXLNGY   0.210122        -0.012409  0.940506
25537         KDM5D   0.294809        -0.022351  0.911426
25538         PRORY   0.054897         0.027772  0.834668
25539      CSPG4P1Y   0.018250         0.021856  0.869524

       baseMean_normalized  -log10(p_value)       regulation
0                -0.059476         0.055788  Not significant
1                -0.056637         0.013018  Not significant
2                -0.054706         0.007882  Not significant
3                -0.059508         0.096837  Not significant
4                -0.059478         0.230086  Not significant
...                    ...              ...              ...
25535            -0.059517         0.078486  Not significant
25536            -0.059487         0.026638  Not significant
25537            -0.059468         0.040279  Not significant
25538            -0.059522         0.078486  Not significant
25539            -0.059530         0.060718  Not significant

[25540 rows x 7 columns]
```

This algorithm creates a base model by removing all the unnecessary genes, reactions and dead-end metabolites from the parent model, recon3d. Then it returns the base model, from which all the context specific models are made.

```
# Extract the list of genes from the expression data
expression_genes = expression_data["Gene_ID"].tolist()

def filter_model_by_genes(model, genes):
    """
    Filter a COBRA model to include only reactions associated with a given list of genes (g
    """
    # Copy the model
    new_model = model.copy()

    # Get the list of genes in the model
    genes_in_model = [gene.id for gene in new_model.genes]

    # Find genes to remove
    genes_to_remove = list(set(genes_in_model) - set(genes))

    genes_to_remove_by_id = set()

    for gene in new_model.genes:
        if gene.name in genes_to_remove:
            genes_to_remove_by_id.add(gene.id)

    genes_to_remove_by_id = list(genes_to_remove_by_id)
```

```
    # Remove genes and their associated reactions
    remove_genes(new_model, genes_to_remove_by_id, remove_reactions=True)
    new_model, _ = prune_unused_reactions(new_model)
    new_model, _ = prune_unused_metabolites(new_model)

    return new_model

# Create the context-specific model
filtered_model = filter_model_by_genes(recon3d_model, expression_genes)

# Save or analyze the new model
cobra.io.write_sbml_model(filtered_model, "emt_base.xml")
```

```
model = filtered_model.copy()
model
```

```
<Model Recon3D at 0x106b8aed0>
```

```
model = read_sbml_model('emt_base.xml')
model
```

```
<Model Recon3D at 0x322dfbad0>
```

```
model = read_sbml_model('Mesenchymal_csm.xml')
model
```

```
<Model Recon3D at 0x17eca0310>
```

The following cell does

1. Normalizing gene expression using MinMaxScaler.
2. Confidence Levels for Genes are assigned confidence levels based on their normalized expression values using thresholds calculated from statistical properties of the data.
3. Gene-Confidence Mapping: For each reaction, the associated genes' confidence levels are retrieved.
4. Propagating Confidence: The reaction is assigned the lowest confidence level among its associated genes. If no associated genes are found, a default confidence level of -1 is assigned.

Justification: Using the lowest confidence among associated genes ensures that reactions are not over-confidently included if any associated gene suggests a lower confidence. Thresholds: Reflect realistic ranges for high, medium, and low expression, providing a balanced and scientifically meaningful distribution of confidence levels.

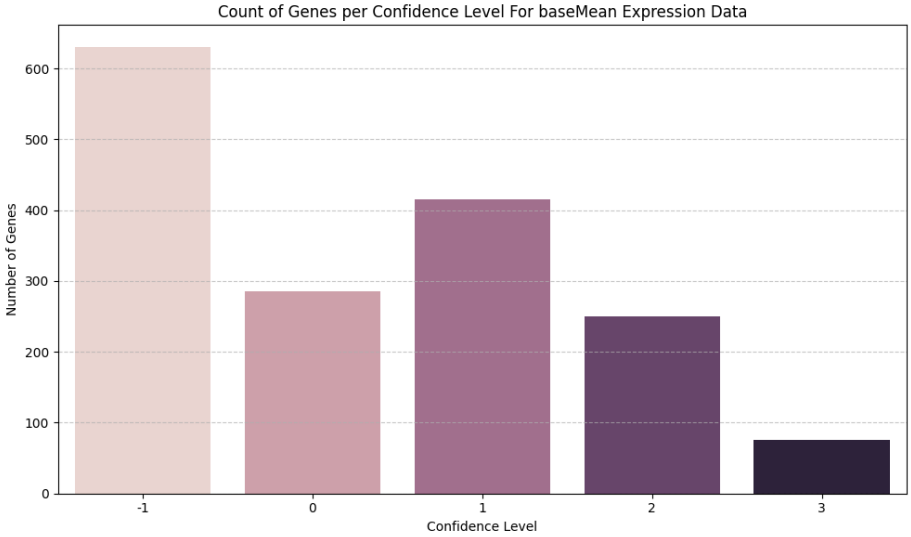
```
id_column_name = "symbol"
column_name = 'baseMean'

# Normalize the 'Sensitive' column
```

```
scaler = MinMaxScaler()
expression_data[f'Normalized_{column_name}'] = scaler.fit_transform(expression_data[[column_

# Calculate statistics for confidence level thresholds
mean = expression_data[f'Normalized_{column_name}'].mean()
std_dev = expression_data[f'Normalized_{column_name}'].std()

# highest_confidence_threshold = mean + std_dev

highest_confidence_threshold = expression_data[f'Normalized_{column_name}'].quantile(0.90)
lowest_confidence_threshold = mean - std_dev

percentile_25 = expression_data[f'Normalized_{column_name}'].quantile(0.25)
percentile_50 = expression_data[f'Normalized_{column_name}'].quantile(0.50)
percentile_75 = expression_data[f'Normalized_{column_name}'].quantile(0.75)

# Assign confidence levels to genes
def assign_gene_confidence(value):
    if value >= highest_confidence_threshold:
        return 3
    elif value >= percentile_75:
        return 2
    elif value >= percentile_50:
        return 1
    elif value >= percentile_25:
        return 0
    elif value <= lowest_confidence_threshold:
        return -1
    else:
        return 0

expression_data['Gene_Confidence_Level'] = expression_data[f'Normalized_{column_name}'].appl

# Initialize a dictionary to store reaction confidence levels
reaction_confidence = {}

# Assign confidence levels to reactions based on associated genes
for reaction in model.reactions:
    reaction_id = reaction.id
    associated_genes = [gene.name for gene in reaction.genes]

    # Get the confidence levels of the associated genes
    gene_confidences = expression_data.loc[expression_data[id_column_name].isin(associated_g

    # Propagate the lowest confidence level to the reaction
    if not gene_confidences.empty:
```

```
        reaction_confidence[reaction_id] = gene_confidences.min()
    else:
        reaction_confidence[reaction_id] = -1  # Assign -1 if no associated genes found

# Assign highest confidence level to specific biomass reactions
for biomass_reaction_id in ['BIOMASS_maintenance_noTrTr', 'BIOMASS_maintenance', 'BIOMASS_re
    if biomass_reaction_id in reaction_confidence:
        reaction_confidence[biomass_reaction_id] = 3

# Convert the dictionary to a DataFrame
reaction_confidence_df = pd.DataFrame(list(reaction_confidence.items()), columns=['Reaction_

# Save the result to a CSV file
reaction_confidence_df.to_csv(f"reaction_{column_name}_confidence_levels.csv", index=False)

# Print to check
print(reaction_confidence_df)
```

```
             Reaction_ID  Confidence_Level
0     EX_5adtststerone_e                -1
1              EX_5fthf_e                -1
2              EX_5mthf_e                -1
3                2AMADPTm                 0
4               2OXOADPTm                 0
...                   ...               ...
1652            HMR_0795                  0
1653            HMR_2817                  2
1654            HMR_2821                  2
1655            HMR_2857                  2
1656            HMR_4772                  1

[1657 rows x 2 columns]
```

```
# Plot Count of Genes per Confidence Level
plt.figure(figsize=(10, 6))
sns.countplot(data=reaction_confidence_df, x='Confidence_Level', hue='Confidence_Level', leg
plt.xlabel('Confidence Level')
plt.ylabel('Number of Genes')
plt.title(f'Count of Genes per Confidence Level For {column_name} Expression Data')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Confidence levels are converted to dict to match the argument type for CORDA reconstruction.

```
# Convert to dictionary
reaction_confidence_dict = reaction_confidence_df.set_index('Reaction_ID')['Confidence_Level
```

```
# Display the dictionary
print(reaction_confidence_dict)
```

```
{'EX_5adtststerone_e': -1, 'EX_5fthf_e': -1, 'EX_5mthf_e': -1, '2AMADPTm': 0, '2OXOADPTm': 0
```

Now, I recontruct the models. The `build` method in CORDA uses confidence scores to modify the emt base model, retaining high-confidence reactions and possibly excluding or down-weighting low-confidence reactions. The goal is to produce a smaller, more accurate model that reflects the specific biological context.

```
opt = CORDA(model, reaction_confidence_dict)
opt.build()
print(opt)
```

```
build status: reconstruction complete
Inc. reactions: 762/1657
 - unclear: 138/286
 - exclude: 247/630
 - low and medium: 301/665
 - high: 76/76
```

Output being stored in `opt`, I use the model from the variable, to get `new_model` which could be any Context specific model, based on column which is picked,

ie.., epithelial or mesenchymal CSM in my case.

```
new_model = opt.cobra_model(column_name)
new_model
```

```
<Model Recon3D at 0x30144e390>
```

Stored the model following the standard `SBML` format as `.xml`

```
cobra.io.write_sbml_model(new_model, f"mesenchymal_fasting_integrated_csm.xml")
```