PEAR

Predicting Gene Expression Alterations Induced by Small Molecules using Deep Tensor Factorization Refinements

We sought to develop a novel strategy for predicting gene perturbations induced by small molecules for a Kaggle competition. With an input of cell-type and small molecule, we are interested in predicting the differential expression of 18,211 genes. As participants, we were given access to raw and differential gene expression tables as well as single cell omics data. Our approach involved embedding the different data sources separately and simultaneously with the intention of training and concatenating flattened embeddings, and subsequently training a deep neural network with these values. We utilized single-cell ATAC-seq data from the omics files to better train the embeddings for cell-type. The model we constructed was successful in its task with a mean row-wise root mean squared error of below 0.3. Although we couldn't achieve ideal results, we feel that our model brings insight and can be integrated with other methods to yield better predictive power in the future.

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
# !pip install -q scanpy
# import scanpy as sc
import pandas as pd
import numpy as np
```

```
de_df = pd.read_parquet('/content/drive/My Drive/ML4FG_final/de_train.parquet')
print(de_df.shape)
GENE_NAMES = de_df.columns[5:]
cols = list(de_df.columns)
de_df.drop(columns=cols[2:5], inplace=True)
de_df.head()
```

```
(614, 18216)
```

	cell_type	sm_name	A1BG	A1BG-AS1	A2M	A2M-AS1	A2MP1	A4GALT
0	NK cells	Clotrimazole	0.104720	-0.077524	-1.625596	-0.144545	0.143555	0.073229
1	T cells CD4+	Clotrimazole	0.915953	-0.884380	0.371834	-0.081677	-0.498266	0.203559
2	T cells CD8+	Clotrimazole	-0.387721	-0.305378	0.567777	0.303895	-0.022653	-0.480681
3	T regulatory cells	Clotrimazole	0.232893	0.129029	0.336897	0.486946	0.767661	0.718590
4	NK cells	Mometasone Furoate	4.290652	-0.063864	-0.017443	-0.541154	0.570982	2.022829

5 rows × 18213 columns

Data Preprocessing

processed **CELL TYPE [ATAC-Seq]** Dataset

```
cell_type => gene level embedding
```

Wrangling & PCA

Wrangling

```
atac_df = pd.read_table('/content/drive/My Drive/ML4FG_final/atac_mapped.tsv')
print(atac_df.shape)
atac_df.head()
```

```
atac_df['idx'] = atac_df['gene'] + '_' + atac_df['location']
atac_df.head()
```

```
atac_df.drop(columns=['gene', 'location'], inplace=True)
atac_df.head()
```

```
pivot_df = atac_df.pivot_table(index='idx', columns='cell_type', values='normalized_count').reset_index()
pivot_df['gene'] = pivot_df.idx.apply(lambda x : x.split('_')[0])
pivot_df['position'] = pivot_df.idx.apply(lambda x : x.split('_')[1])
pivot_df['chrom'] = pivot_df.position.apply(lambda x : x.split(':')[0])
pivot_df.drop(columns=['idx'], inplace=True)
pivot_df.head()
```

```
# pivot_df.to_csv('/content/drive/My Drive/ML4FG_final/atac_pivot.tsv', sep='\t', index=False, encoding='utf-8')
```

```
pivot_df = pivot_df.fillna(pivot_df.mean(numeric_only=True))
print(sum(pivot_df.isnull().sum()), 'nulls')
```

```
    0 nulls
```

```
pivot_df.head()
```

PCA

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import plotly.express as px

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(pivot_df.iloc[:,:-3])

# Perform PCA
pca = PCA()
pca_result = pca.fit_transform(scaled_data)


# Create a DataFrame with the principal components
pc_columns = [f'PC{i+1}' for i in range(pca_result.shape[1])]
pc_df = pd.DataFrame(data=pca_result, columns=pc_columns)

# Plot the top principal components
fig_pc = px.scatter(pc_df, x='PC1', y='PC2', color=pivot_df.gene, title='Top Principal Components')

# Add total variance explained as an annotation
# fig_pc.add_annotation(
#     x=-100,
#     y=85,
#     text=f'Total variance explained: {sum(pca.explained_variance_ratio_):.2%}',
#     showarrow=False,
#     font=dict(size=12, color='black')
# )

fig_pc.show()
```


```
# Calculate the explained variance ratio for each principal component
explained_variance_ratio = [round(x,4) for x in pca.explained_variance_ratio_]

# Create a DataFrame for EVR vs Top 10 PCs
num_top_pcs = 6
evr_vs_pcs_df = pd.DataFrame({'Principal Component': range(1, num_top_pcs + 1),
                              'Explained Variance Ratio': explained_variance_ratio[:num_top_pcs]})

# Plot EVR vs Top 10 PCs
fig_evr_vs_pcs = px.line(evr_vs_pcs_df, x='Principal Component', y='Explained Variance Ratio',
                         title='Scree Plot: Explained Variance Ratio vs Top 10 Principal Components',
                         labels={'Principal Component': 'Principal Components',
                                 'Explained Variance Ratio': 'Explained Variance Ratio'})

# Add total variance explained as an annotation
fig_evr_vs_pcs.add_annotation(
    x=2.5,
    y=1,
    text=f'Total variance explained: {sum(pca.explained_variance_ratio_):.2%}',
    showarrow=False,
    font=dict(size=12, color='black')
)

# Widen x-axis bounds
fig_evr_vs_pcs.update_layout(xaxis=dict(tickmode='linear', dtick=1, range=[0.5, num_top_pcs + 0.5]))

fig_evr_vs_pcs.show()
```

Mappings for input

```
def get_ctm(atac_pivot_df):
  ATACABLE_GENE_NAMES = set(atac_pivot_df.gene).intersection(GENE_NAMES)
  ADD_THESE_IN = set(GENE_NAMES) - set(atac_pivot_df.gene)
  print(len(ATACABLE_GENE_NAMES), len(ADD_THESE_IN))

  # List of columns to sum
  cell_types = ["B cells", "Myeloid cells", "NK cells", "T cells CD4+", "T cells CD8+", "T regulatory cells"]

  result_df = atac_pivot_df.groupby("gene")[cell_types].sum().reset_index().round(3)
  print(result_df.shape)
  result_df = result_df[result_df['gene'].isin(ATACABLE_GENE_NAMES)].set_index('gene')
  print(result_df.shape)

  # Now I need to add in all GENES that are also in the de_df file
  column_names = list(result_df.columns)
  row_indices = list(ADD_THESE_IN)
  missings_df = pd.DataFrame(0, index=row_indices, columns=column_names)
  for_embedding_df = pd.concat([result_df, missings_df]).reset_index()

  # Impute those missing values with cell-type avgs
  sorted_for_embedding_df = for_embedding_df.set_index('index').loc[GENE_NAMES].reset_index()
  column_avg = sorted_for_embedding_df.mean()
  sorted_for_embedding_df.replace(0, column_avg, inplace=True)

  cell_type_mappings = {}
  for ct in sorted_for_embedding_df.columns:
    if ct=='index':
      continue
    cell_type_mappings[ct] = np.array(sorted_for_embedding_df.loc[:, ct])

  return cell_type_mappings
```

Use as weights for embeddings in the model !!

DEPRECATED SECTION

processed **GENE EXPRESSION** dataset

```
cell_type [6 dimension embedding per gene] , small_molecule => genes
```

Wrangling

```
raw_gene_expression_df = pd.read_parquet('/content/drive/My Drive/ML4FG_final/unique_ge.parquet')
print(raw_gene_expression_df.shape)
raw_gene_expression_df.head()
```

```
(602, 21257)
```

cell_type	sm_name	A1BG	A1BG-AS1	A2M	A2M-AS1	A2MP1	A4GALT	AAAS

There are a decent amount of nulls in our dataset. We only want to keep genes with >=75% non-null values. Those with >25% missingness should be pruned altogether.

Once we have a dataset with only genes s.t. each gene has >=75% non-null values, all null values will be replaced by the mean expression value for that gene, per cell type.

```
cell_count = int(raw_gene_expression_df.shape[0]*0.75)
print(cell_count, 'is our threshold per gene of non-null values')
# any genes with < # will be removed altogether
# any genes with >= # will have any remaining nulls imputed with the mean value of the respective column
```

```
451 is our threshold per gene of non-null values
```

```
value_frame = raw_gene_expression_df.iloc[:,2:]
cells_keep_bool_df, cells_countnull_df = sc.pp.filter_genes(value_frame, min_cells=cell_count)
print(cells_keep_bool_df[cells_keep_bool_df==True].shape, 'genes are KEPT, rest removed')
```

```
(10297,) genes are KEPT, rest removed
```

```
keep_genes = list(cells_keep_bool_df[cells_keep_bool_df==True].index)
keep_genes[:5]
```

```
['A1BG-AS1', 'A2M', 'A2M-AS1', 'AAAS', 'AACS']
```

```
# Here are the non null counts for all genes
nulls_df = pd.DataFrame(cells_countnull_df)
nulls_df['percent_nonnull'] = nulls_df[0]/raw_gene_expression_df.shape[0]
nulls_df.head()
```

	0	percent_nonnull
A1BG	434	0.720930
A1BG-AS1	506	0.840532
A2M	459	0.762458
A2M-AS1	484	0.803987
A2MP1	346	0.574751

```
selected_cols = list(raw_gene_expression_df.columns)[:2] + keep_genes
assert len(selected_cols) == cells_keep_bool_df[cells_keep_bool_df==True].shape[0] + 2
```

```
gene_expr_df = raw_gene_expression_df[selected_cols]
print(sum(gene_expr_df.isnull().sum()), 'nulls')
gene_expr_df.head()
```

```
437302 nulls
```

	cell_type	sm_name	A1BG-AS1	A2M	A2M-AS1	AAAS	AACS	AAGAB	AAK
0	B cells	Alvocidib	NaN	NaN	NaN	NaN	NaN	3.944556	Na
1	B cells	CHIR-99021	5.491961	5.550509	NaN	4.974828	5.108318	5.144436	4.76481
2	B cells	Crizotinib	4.779984	5.367214	4.364774	5.156859	5.289055	5.142470	5.31749
3	B cells	Dactolisib	5.340497	NaN	4.201869	5.056410	5.371682	5.128500	5.02104
4	B cells	Foretinib	5.187629	5.183072	5.672760	5.184502	5.281209	5.149990	5.01500

5 rows × 10299 columns

```
# gene_expr_imputed_df = gene_expr_df.fillna(gene_expr_df.mean(numeric_only=True))
# gene_expr_imputed_df.head()
```

```
# # Group by 'cell_type' and transform to fill NaN values with the mean of each group
# # gene_expr_imputed_df_ = gene_expr_df.iloc[:,:].groupby('cell_type').transform(lambda x: x.fillna(x.mean()))
# # gene_expr_imputed_df_.head()
# print(sum(gene_expr_df.isnull().sum()))

# frames = []
# for cell_type, frame in gene_expr_df.groupby('cell_type'):
#   print(sum(frame.isnull().sum()), end=' > ')
#   temp = frame.fillna(frame.mean(numeric_only=True))

#   frames.append(temp)
#   print(sum(temp.isnull().sum()))

# gene_expr_imputed_df = pd.concat(frames).reset_index(drop=True)
# print(gene_expr_imputed_df.shape)
# gene_expr_imputed_df.head()


# Group by "cell_type" and calculate the mean for each numeric column
means_per_cell_type = gene_expr_df.groupby('cell_type').mean()

# Iterate through each numeric column in the DataFrame
for column in gene_expr_df.select_dtypes(include='number').columns:
    # Identify null values in the column
    null_values = gene_expr_df[column].isnull()

    # Replace null values with the mean value corresponding to the "cell_type"
    gene_expr_df.loc[null_values, column] = gene_expr_df.loc[null_values, 'cell_type'].map(means_per_cell_type[column])

# Now, df contains the imputed values
print(sum(gene_expr_df.isnull().sum()), 'nulls')
gene_expr_imputed_df = gene_expr_df.fillna(gene_expr_df.mean(numeric_only=True))
print(sum(gene_expr_imputed_df.isnull().sum()), 'nulls')
gene_expr_imputed_df.head()
```

```
<ipython-input-87-f9e78ebb91d1>:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a f

30 nulls
0 nulls
```

	cell_type	sm_name	A1BG-AS1	A2M	A2M-AS1	AAAS	AACS	AAGAB	AAK
0	B cells	Alvocidib	5.213740	5.151513	4.537179	5.131510	5.244142	3.944556	5.05938
1	B cells	CHIR-99021	5.491961	5.550509	4.537179	4.974828	5.108318	5.144436	4.76481
2	B cells	Crizotinib	4.779984	5.367214	4.364774	5.156859	5.289055	5.142470	5.31749
3	B cells	Dactolisib	5.340497	5.151513	4.201869	5.056410	5.371682	5.128500	5.02104
4	B cells	Foretinib	5.187629	5.183072	5.672760	5.184502	5.281209	5.149990	5.01500

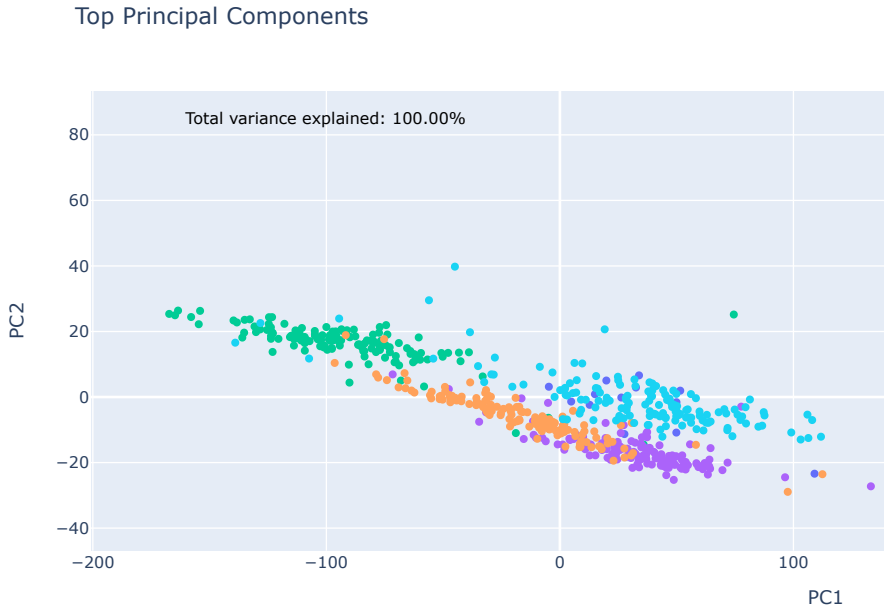
5 rows × 10299 columns

```
gene_expr_imputed_df.to_csv('/content/drive/My Drive/ML4FG_final/geneexp_impute_embed.tsv', index=False, sep='\t')
```

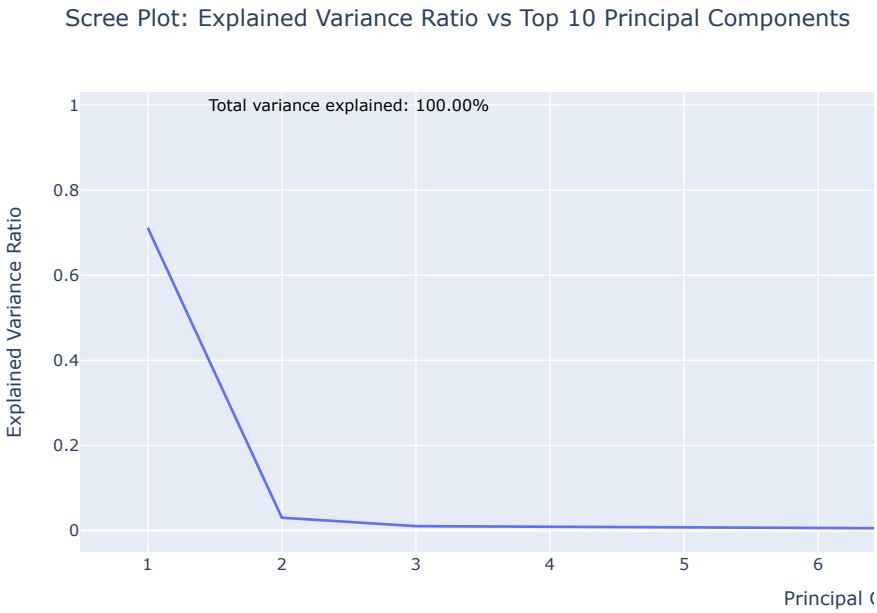
PCA

```
gene_expr_imputed_df = pd.read_table('/content/drive/My Drive/ML4FG_final/geneexp_impute_embed.tsv')
```

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import plotly.express as px

# Extract relevant data
data = gene_expr_imputed_df.iloc[:, 2:]

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Perform PCA
pca = PCA()
pca_result = pca.fit_transform(scaled_data)


# Create a DataFrame with the principal components
pc_columns = [f'PC{i+1}' for i in range(pca_result.shape[1])]
pc_df = pd.DataFrame(data=pca_result, columns=pc_columns)

# Plot the top principal components
fig_pc = px.scatter(pc_df, x='PC1', y='PC2', color=gene_expr_imputed_df.cell_type, title='Top Principal Components')

# Add total variance explained as an annotation
fig_pc.add_annotation(
    x=-100,
    y=85,
    text=f'Total variance explained: {sum(pca.explained_variance_ratio_):.2%}',
    showarrow=False,
    font=dict(size=12, color='black')
)

fig_pc.show()
```



```
# Calculate the explained variance ratio for each principal component
explained_variance_ratio = [round(x,4) for x in pca.explained_variance_ratio_]

# Create a DataFrame for EVR vs Top 10 PCs
num_top_pcs = 12
evr_vs_pcs_df = pd.DataFrame({'Principal Component': range(1, num_top_pcs + 1),
                              'Explained Variance Ratio': explained_variance_ratio[:num_top_pcs]})

# Plot EVR vs Top 10 PCs
fig_evr_vs_pcs = px.line(evr_vs_pcs_df, x='Principal Component', y='Explained Variance Ratio',
                         title='Scree Plot: Explained Variance Ratio vs Top 10 Principal Components',
                         labels={'Principal Component': 'Principal Components',
                                 'Explained Variance Ratio': 'Explained Variance Ratio'})

# Add total variance explained as an annotation
fig_evr_vs_pcs.add_annotation(
    x=2.5,
    y=1,
    text=f'Total variance explained: {sum(pca.explained_variance_ratio_):.2%}',
    showarrow=False,
    font=dict(size=12, color='black')
)

# Widen x-axis bounds
fig_evr_vs_pcs.update_layout(xaxis=dict(tickmode='linear', dtick=1, range=[0.5, num_top_pcs + 0.5]))

fig_evr_vs_pcs.show()
```



Encodings

```
gene_expr_imputed_df = pd.read_table('/content/drive/My Drive/ML4FG_final/geneexp_impute.tsv')
print(gene_expr_imputed_df.shape)
gene_expr_imputed_df.head()
```

```
(602, 10299)
```

	cell_type	sm_name	A1BG-AS1	A2M	A2M-AS1	AAAS	AACS	AAGAB	AAK
0	B cells	Alvocidib	5.213740	5.151513	4.537179	5.131510	5.244142	3.944557	5.05938

```
# gene_expr_imputed_df['cell_type_embed'] = gene_expr_imputed_df['cell_type'].apply(lambda x : cell_type_mappings[x] if x in cel
# gene_expr_imputed_df.head()


n_celltypes = gene_expr_imputed_df.cell_type.value_counts().shape[0]
n_sm = gene_expr_imputed_df.sm_name.value_counts().shape[0]
print(f'num celltypes: {n_celltypes}\nnum sm: {n_sm}')

gene_expr_imputed_df.head()
```

```
num celltypes: 6
num sm: 144
```

	cell_type	sm_name	A1BG-AS1	A2M	A2M-AS1	AAAS	AACS	AAGAB	AAK
0	B cells	Alvocidib	5.213740	5.151513	4.537179	5.131510	5.244142	3.944557	5.05938
1	B cells	CHIR-99021	5.491961	5.550509	4.537179	4.974828	5.108318	5.144436	4.76481
2	B cells	Crizotinib	4.779984	5.367214	4.364774	5.156859	5.289055	5.142470	5.31749
3	B cells	Dactolisib	5.340497	5.151513	4.201869	5.056410	5.371682	5.128501	5.02104
4	B cells	Foretinib	5.187629	5.183072	5.672760	5.184502	5.281210	5.149990	5.01500

5 rows × 10299 columns

```
encoded_data = pd.get_dummies(gene_expr_imputed_df, columns=['cell_type', 'sm_name'])
encoded_data.head()
```

	A1BG-AS1	A2M	A2M-AS1	AAAS	AACS	AAGAB	AAK1	AAMDC	AAMP
0	5.213740	5.151513	4.537179	5.131510	5.244142	3.944557	5.059385	5.189637	4.302887
1	5.491961	5.550509	4.537179	4.974828	5.108318	5.144436	4.764812	5.327592	5.120109
2	4.779984	5.367214	4.364774	5.156859	5.289055	5.142470	5.317490	5.297750	5.356219
3	5.340497	5.151513	4.201869	5.056410	5.371682	5.128501	5.021046	5.096577	5.080998
4	5.187629	5.183072	5.672760	5.184502	5.281210	5.149990	5.015003	5.348046	5.320029

5 rows × 10447 columns

```
encoded_data.iloc[:,-151:-144]
```

	ZZEF1	cell_type_B cells	cell_type_Myeloid cells	cell_type_NK cells	cell_type_T cells CD4+	cell_type cells CD
0	5.164732	1	0	0	0	
1	5.332543	1	0	0	0	
2	5.361841	1	0	0	0	
3	5.493635	1	0	0	0	
4	5.223923	1	0	0	0	
...	...	...	...	...	...	
597	5.192352	0	0	0	0	
598	5.317637	0	0	0	0	
599	5.147752	0	0	0	0	
600	5.213854	0	0	0	0	
601	5.282995	0	0	0	0	

PEAR MODEL

data & parameters

```
atac_pivot_df = pd.read_table('/content/drive/My Drive/ML4FG_final/atac_pivot.tsv')
print(atac_pivot_df.shape)
atac_pivot_df.head()
```

```
(106930, 9)
```

	B cells	Myeloid cells	NK cells	T cells CD4+	T cells CD8+	T regulatory cells	gene	posi
0	9.666737	651.266850	26.186500	61.92950	8.024200	37.697043	A2ML1	chr12:884 885
1	64.651140	269.505400	146.144850	357.11100	16.876087	13.182861	A2ML1	chr12:886 886
2	16.931800	27.936155	201.119110	134.24794	40.717847	9.311357	A2MP1	chr12:924 924

```
cell_type_mappings = get_ctm(atac_pivot_df)
print('ctm', cell_type_mappings['B cells'].shape)
```

```
12986 5225
(24884, 7)
(12986, 6)
ctm (18211,)
<ipython-input-6-fd05a1364e88>:22: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a fu
  column_avg = sorted_for_embedding_df.mean()
```

```
gene_expr_imputed_df = de_df
print('dedf', gene_expr_imputed_df.shape)
gene_expr_imputed_df.head()
```

```
dedf (614, 18213)
```

	cell_type	sm_name	A1BG	A1BG-AS1	A2M	A2M-AS1	A2MP1	A4GALT
0	NK cells	Clotrimazole	0.104720	-0.077524	-1.625596	-0.144545	0.143555	0.073229
1	T cells CD4+	Clotrimazole	0.915953	-0.884380	0.371834	-0.081677	-0.498266	0.203559
2	T cells CD8+	Clotrimazole	-0.387721	-0.305378	0.567777	0.303895	-0.022653	-0.480681
3	T regulatory cells	Clotrimazole	0.232893	0.129029	0.336897	0.486946	0.767661	0.718590
4	NK cells	Mometasone Furoate	4.290652	-0.063864	-0.017443	-0.541154	0.570982	2.022829

5 rows × 18213 columns

```
cell_types = list(set(de_df.cell_type))
small_molecules = list(set(de_df.sm_name))
name_representations = cell_types + small_molecules


n_celltypes = gene_expr_imputed_df.cell_type.value_counts().shape[0] # Number of unique cell types = 6
n_sm = gene_expr_imputed_df.sm_name.value_counts().shape[0] # Number of unique sm = 144
n_genes = gene_expr_imputed_df.shape[1] - 2  # Number of genes
n_celltype_factors = len(cell_type_mappings.keys()) # 6
n_sm_factors = 128
n_nodes = 256

print(n_celltypes, n_celltype_factors, n_sm, n_sm_factors, n_genes, n_nodes)
```

```
6 6 146 128 18211 256
```

setup embeddings with ATAC-based weightings

- first, reduce dimensionality of ATAC weightings to top 6 PCs

```
import numpy as np
from sklearn.decomposition import PCA

ctm = cell_type_mappings

# combine arrays into a single matrix
data_matrix = np.vstack(list(ctm.values()))

pca = PCA(n_components=6)
reduced_data = pca.fit_transform(data_matrix)

for i, key in enumerate(ctm.keys()):
    ctm[key] = reduced_data[:, i]

data_matrix_ctm = np.vstack(list(ctm.values()))
```

encodings, scalings, train-test splitting

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import MinMaxScaler
from keras.layers import Input, Dense, concatenate, Embedding, Flatten
from keras.models import Model
from keras.optimizers import Adam

# one hot encode
encoded_data = pd.get_dummies(gene_expr_imputed_df, columns=['cell_type', 'sm_name'])

x_cols = encoded_data.columns[-(n_celltypes+n_sm):]
y_cols = filter(lambda x : x not in x_cols, encoded_data.columns)

X = encoded_data.loc[:, x_cols]
y = encoded_data.loc[:, y_cols]

# normalize
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

model

```
from keras.layers import Input, Dense, concatenate, Embedding, Flatten, Reshape

def build_model_atac(n_celltype_factors, n_sm_factors, n_nodes, n_genes, n_layers, ctm):

    celltype_input = Input(shape=(n_celltypes,), name="celltype_input")
    celltype_embedding = Embedding(n_celltypes, n_celltype_factors,
      weights=[ctm], input_length=1, trainable=True, name="celltype_embedding")
    celltype = Flatten()(celltype_embedding(celltype_input))

    sm_input = Input(shape=(n_sm,), name="sm_input")
    sm_embedding = Dense(n_sm_factors, name="sm_embedding")(sm_input)
    sm = Dense(n_sm_factors, name="sm")(sm_embedding)

    concatenated = concatenate([celltype, sm])
    for i in range(n_layers):
        concatenated = Dense(n_nodes, activation='relu', name="dense_{}".format(i))(concatenated)

    output_layer = Dense(n_genes, name="output")(concatenated)

    model = Model(inputs=[celltype_input, sm_input], outputs=output_layer)
    return model
```

cross-fold validation, training, and evaluation

```
import keras.backend as K

def mean_rowwise_rmse(y_true, y_pred):
    squared_error = K.square(y_true - y_pred)
    rowwise_squared_error = K.mean(squared_error, axis=1)
    rowwise_rmse = K.sqrt(rowwise_squared_error)

    # calculate mean of rowwise root mean squared error
    mean_rowwise_rmse = K.mean(rowwise_rmse)

    return mean_rowwise_rmse


import itertools
kf = KFold(n_splits=5, shuffle=True, random_state=42)

best_layer_val = None
best_mrrmse = float('inf')

for n_layers in [8,2]:
  accuracy_values = []
  mrrmse_values = []
  for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    model = build_model_atac(n_celltype_factors, n_sm_factors, n_nodes, n_genes, n_layers, data_matrix_ctm)
    model.compile(optimizer='adam', loss=mean_rowwise_rmse, metrics=['accuracy'])

    # train model
    trained_model = model.fit([X_train_fold[:, :n_celltypes], X_train_fold[:, n_celltypes:]], y_train_fold, epochs=50, batch_siz
    accuracy_values.append(np.mean(trained_model.history.get('accuracy', 0.5)))

    # evaluate model on validation set
    mrrmse = model.evaluate([X_val_fold[:, :n_celltypes], X_val_fold[:, n_celltypes:]], y_val_fold, verbose=0)
    mrrmse_values.append(mrrmse)

  average_mrrmse = np.mean(mrrmse_values)
  average_acc = np.mean(accuracy_values)

  print(f"Layers: {n_layers}, Average MRRMSE: {average_mrrmse}, Average accuracy: {average_acc}")

  if average_mrrmse < best_mrrmse:
    best_mrrmse = average_mrrmse
    best_layer_val = n_layers

print(f"Best Num Layers: {best_layer_val}")
```

```
Layers: 8, Average MRRMSE: 0.674391302652657, Average accuracy: 0.027586851300671695
Layers: 2, Average MRRMSE: 0.6743998857215047, Average accuracy: 0.018748117343522608
Best Num Layers: 8
```

```
test_loss, test_acc = model.evaluate([X_test[:, :n_celltypes], X_test[:, n_celltypes:]], y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")
```

```
4/4 [==============================] - 0s 25ms/step - loss: 1.0447 - accuracy: 0.0163
Test Loss: 1.044743537902832
Test Accuracy: 0.016260161995887756
```

```
model.summary()
```

```
Model: "model_128"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 celltype_input (InputLayer  [(None, 6)]                  0         []
 )

 sm_input (InputLayer)       [(None, 146)]                0         []

 celltype_embedding (Embedd  (None, 6, 6)                 36        ['celltype_input[0][0]']
 ing)
```

```
 sm_embedding (Dense)        (None, 128)               18816     ['sm_input[0][0]']

 flatten_128 (Flatten)       (None, 36)                0         ['celltype_embedding[0][0]']

 sm (Dense)                  (None, 128)               16512     ['sm_embedding[0][0]']

 concatenate_128 (Concatena  (None, 164)               0         ['flatten_128[0][0]',
 te)                                                             'sm[0][0]']

 dense_0 (Dense)             (None, 256)               42240     ['concatenate_128[0][0]']

 dense_1 (Dense)             (None, 256)               65792     ['dense_0[0][0]']

 output (Dense)              (None, 18211)             4680227   ['dense_1[0][0]']

==================================================================================================
Total params: 4823623 (18.40 MB)
Trainable params: 4823623 (18.40 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Predict

```
import numpy as np

def predict_de(data):

    ct = data['cell_type']
    sm = data['small_molecule']

    cell_type_input = []
    for name in cell_types:
        if name == ct:
            cell_type_input.append(1)
        else:
            cell_type_input.append(0)


    reshaped_data1 = np.expand_dims(np.array(cell_type_input), axis=0)

    sm_input = []
    for name in small_molecules:
        if name == sm:
            sm_input.append(1)
        else:
            sm_input.append(0)

    reshaped_data2 = np.expand_dims(np.array(sm_input), axis=0)

    if sum(cell_type_input) == 0 and sum(sm_input) == 0:
      print('*** warning: neither input seen in training - results not likely to be accurate')

    predictor = [reshaped_data1, reshaped_data2]
    return final_model.predict(predictor)[0]


trial = {
    'cell_type': 'NK cells',
    'small_molecule': 'Tivantinib'
}

pred = predict_de(trial)
```