Bioinformatics, 2022, 0–0
doi: 10.1093/bioinformatics/xxxxx

ECBME4060: Intro-Genomic Info Sci & Tech

Multimodal Single-Cell Integration for CITE-seq Data

Anurag Sharma, Fenisha Shah and Nikhil Kuppa

[1]Department of Biomedical Engineering, Columbia University
Submitted on: 19th December, 2022

Abstract

Motivation: CITEseq is a technique that allows researchers to measure both gene expression and protein levels within individual cells, providing valuable insights into the functional state of cells. However, accurately predicting protein levels from gene expression values is a challenging task, as protein levels are often regulated by multiple mechanisms beyond just gene expression. We aim to construct a model that will best predict the surface protein levels.

Results: Having implemented 4 Regression models, 2 Encoder-Decoder Neural Network models and 1 Deep Learning conv1d Encoder-Decoder model, we find the Deep Learning model to be the best model (based on R-squared value of 0.2064) to predict surface protein levels given the gene expression data.

Contact: A.Sharma - as6744@columbia.edu, F. Shah - fs2789@columbia.edu,
N.Kuppa - nkk2126@columbia.edu

Supplementary information: https://www.kaggle.com/competitions/open-problems-multimodal

1. Introduction

Single-cell analysis techniques allow researchers to study the molecular makeup of individual cells, providing a more nuanced understanding of cellular processes than traditional bulk analysis methods. In recent years, there has been a growing interest in integrating multiple types of molecular measurements, such as DNA, RNA, and protein, in a single cell. This type of multimodal single-cell analysis allows for a more comprehensive understanding of cellular behavior and can provide insights into how these different molecular measurements co-vary.

For this project, we are working with a dataset comprising single-cell multi-omics data collected from mobilized peripheral CD34+ hematopoietic stem and progenitor cells (HSPCs) from four different donors. The measurement for this data is taken over 10 days at 5 different time points each day. This is done to observe noticeable changes in the DNA, RNA, and proteins in the cells. DNA must be accessible (have chroma data) to produce RNA (GEX data), and RNA in turn is used as a template to produce a protein (ADT data). To execute the process described, these cells were collected into two assays for measurement purposes. Each assay measures two modalities. The first assay measures chromatin accessibility and gene expression, whereas the second assay measures gene expression and the surface protein level. The project provides us with one modality and expects us to predict the paired modality.

One approach to predicting the co-variation of DNA, RNA, and protein measurements in single cells is to use machine learning techniques. These techniques can identify patterns in the data and make predictions about the relationship between the different molecular measurements. By training a machine learning model on a dataset of single cells with DNA, RNA, and protein measurements, we can predict how these measurements co-vary and potentially identify new relationships that were previously unknown.

One specific application of multimodal single-cell integration is the prediction of protein levels in CITEseq data-given gene expression values. CITEseq is a technique that allows researchers to measure both gene expression and protein levels within individual cells, providing valuable insights into the functional state of cells. However, accurately predicting protein levels from gene expression values is a challenging task, as protein levels are often regulated by multiple mechanisms beyond just gene expression. By integrating multiple datasets and utilizing advanced machine learning techniques, multimodal single-cell integration can provide more accurate and comprehensive predictions of protein levels in CITEseq data

2. Methods

We started by conducting some Exploratory Data Analysis (EDA) to understand the datasets better. We soon realized that due to lack of target data for the provided test data, there was no way of testing the accuracy of our predictive model. Hence, we decided to split the train (inputs and targets) dataset provided into train and test subset data, and completely ignore the test dataset provided.

The unprocessed training input dataset's shape is (70988, 22050), and it has a considerable amount of zero-values. This led us to explore dimensionality reduction techniques in order to reduce the sparseness of the data, as well as save time and memory on our coding environment. We preprocessed our data in two different ways - Truncated Singular Value Decomposition (TSVD) and Principal Component Analysis

(PCA). Using this preprocessed input data, we built a CNN Encoder-Decoder model, and benchmarked it against a few regression

2.1. Exploratory Data Analysis (EDA)

We started the Exploratory Data Analysis process by understanding the meta dataset better and hence we plotted a bar plot as seen in figure 2.1.1 of how the number of different donor cells in the dataset varies with the experiment days.

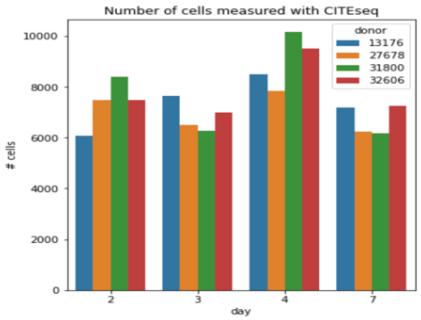


Figure (2.1.1): Distribution of CITEseq measured cells of donors over the days

To further understand the distribution of the entire data better, we observe the fractions of metadata's features in the form of a pie chart as shown in figure 2.1.2.

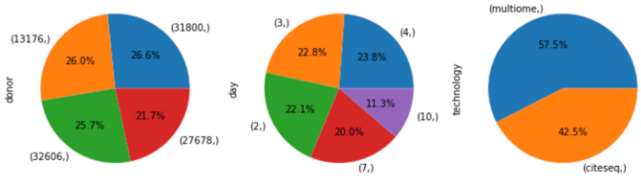


Figure (2.1.2): Distribution of Donors, Day-wise Data, and Technology used to collect the data

We observe that the cell data is fairly balanced. Nearly the same number of cells came from each donor (the five-digit numbers outside the first pie are the donor ids). Additionally, the experiment's days were evenly distributed too. Day 10, the final day, only receives 11% of the cells. The 10th day also happens to be the only one that is completely absent from the train data. Additionally, there is no data from donor 27678 in the train set.

After understanding the distribution of the metadata, we narrowed down our focus to CITEseq data.

gene_id	ENSG00000121410_A1BG	ENSG00000268895_A1BG-AS1	ENSG00000175899_A2M	ENSG00000245105_A2M-AS1	ENSG00000166535_A2ML1	ENSG0
cell_id						
45006fe3e4c8	0.0	0.0	0.0	0.0	0.0	
d02759a80ba2	0.0	0.0	0.0	0.0	0.0	
c016c6b0efa5	0.0	0.0	0.0	0.0	0.0	
ba7f733a4f75	0.0	0.0	0.0	0.0	0.0	
fbcf2443ffb2	0.0	0.0	0.0	0.0	0.0	

5 rows × 22050 columns

```
Shape: (70988, 22050)
Missing values: 0
Genes which never occur in train: 449
Zero entries in train: 78%
```

Figure (2.1.3): Snapshot of a snippet of CITEseq Inputs Dataset

2.2. Dimensionality Reduction (PCA & TSVD)

Due to the high level of sparsity in the dataset, we explored two different dimension reduction techniques. The first one was Principal Components Analysis (PCA). PCA is a statistical technique that is used to reduce the dimensionality of a dataset while retaining as much information as possible, and is often used for data compression, feature selection, and for noise reduction. It is a useful technique for applications like pattern recognition, and feature selection. However, PCA is a linear method, and may not be suitable for all types of data.

The second technique was Truncated Singular Value Decomposition (TSVD). TSVD, a variant of SVD, factorizes a matrix into the product of three matrices. SVD is a powerful technique that is used in a wide range of applications, including data compression, image processing, and natural language processing. TSVD only keeps the top k singular values and corresponding singular vectors, where k is a user-specified parameter. TSVD has several advantages over other dimensionality reduction techniques, as it can handle sparse matrices efficiently and is more robust to noise and outliers in the data. Despite a lot of similarities between TSVD and PCA, we use TSVD because it operates on the sample vectors directly instead of the covariance matrix; thus being more efficient. The dimensions we reduced the dataset to is 240.

2.3. Predictive Models

After preprocessing our data, we started designing models. We started off with testing various regression models because it is widely used to determine the relationship between predictors and target variables. The regression models we implemented were KNN, Ridge, ElasticNet and Lasso.

To use KNN regression, you need to choose a value for the parameter k and a distance metric to measure the distance between samples. The most common distance metric used in KNN is Euclidean distance, which is the straight-line distance between two points in space. To make a prediction for a new sample, the algorithm finds the k nearest neighbors to that sample using the chosen distance metric. It then calculates the average of the target values of those neighbors as the prediction for the new sample. We used KNN regression with k = 9, using kneighborsregressor from the scikit-learn package.

Ridge Regression reduces the model complexity by coefficient shrinkage, and uses the L2 regularization technique. The penalty is made up of the sum of squares of coefficients. We use 0.05 as the value of the hyperparameter alpha, as it is believed to result in the highest R-squared value. Lasso Regression uses L1 regularization, and unlike Ridge, it performs feature selection and also its penalty is made up of the sum of absolute values of coefficients. The smaller value of alpha, 0.3, is very effective due to the feature selection method this technique incorporates. ElasticNet Regression is a combination of both L2 and L1 regularization. After a couple iterations, we find that using alpha = 1 and l1_ratio = 1, gives us the best R-squared value. All of these regression analysis was done using scikit-learn's Ridge, Lasso and ElasticNet methods.

After regression analysis, we implemented the encoder-decoder neural network algorithm we sourced from Kaggle. The encoder-decoder architecture is a type of neural network that is used for sequence-to-sequence tasks, where the goal is to map an input sequence to an output sequence of different lengths. The encoder-decoder architecture consists of two main components: an encoder and a decoder.

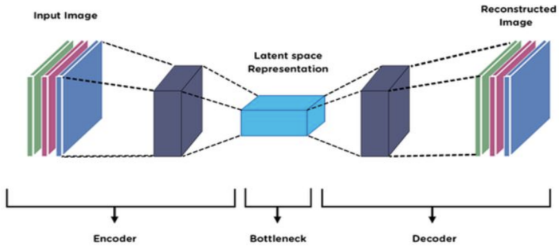


Figure (2.1.4): Encoder-Decoder Architecture Visualization

The encoder takes in the input sequence and converts it into a

fixed-length context vector, which contains a summary of the information in the input sequence. The context vector is then passed to the decoder, which uses it to generate the output sequence.

The kaggle notebook's algorithm's architecture uses an encoder to reduce the dimensions from 240 to 120 to 60 and eventually to 30, and then the decoder reconstructs the dimensions from 30 to 70 and finally the target dimensions of 140. Taking inspiration from this model, and also keeping in mind the limited memory resources available, we made modifications by eliminating a few intermediate layers and executed a similar model. At the encoder's side of the algorithm, we reduced the dimensions from 240 to 60 in one layer, and reconstructed it from 60 to 140 in one layer. PyTorch was used for this architecture, and we used k-fold validation to account for any imbalances in the dataset (experimenting with various k's we found that k=5 gives best results).

We then architected our own Convolutional Neural Networks based Encoder-Decoder where we used Conv1d instead since our data is essentially in the form of gene-signal. We used the 240 principal components from PCA transformed training data as the input to our CNN model. This time, we used Tensorflow to implement the architecture and also used tensorflow pipeline/graphs for the whole workflow. Having fit different hyper-parameters, we found that the best batch size is 128 with 5 epochs, as the model was overfitting if the number of epochs > 5. Also found that the best learning rate was 0.0005. We plotted some of the latent features against each other to visualize some parts of the latent vector (Only 2 plots are shown but we can also analyze it for all dimensions in the future). These latent dimensions represent the intermediate stages where gene expression is being transformed into proteins, thereby mimicking the intermediate stages of transcription.

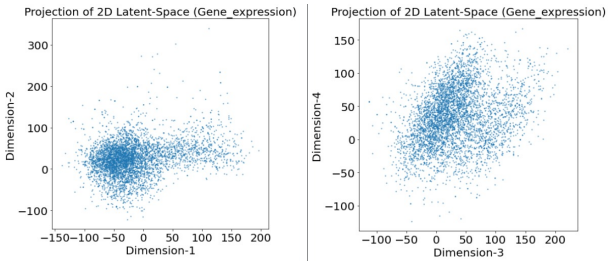
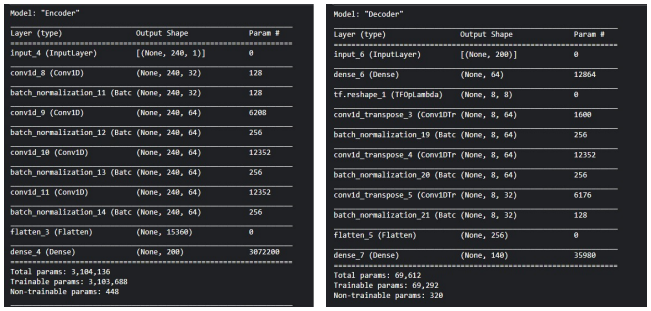


Figure (2.1.5): Intermediate stages of Transcription



Model: "Encoder"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 240, 1)]	0
conv1d_8 (Conv1D)	(None, 240, 32)	128
batch_normalization_11 (Batc	(None, 240, 32)	128
conv1d_9 (Conv1D)	(None, 240, 64)	6208
batch_normalization_12 (Batc	(None, 240, 64)	256
conv1d_10 (Conv1D)	(None, 240, 64)	12352
batch_normalization_13 (Batc	(None, 240, 64)	256
conv1d_11 (Conv1D)	(None, 240, 64)	12352
batch_normalization_14 (Batc	(None, 240, 64)	256
flatten_3 (Flatten)	(None, 15360)	0
dense_4 (Dense)	(None, 200)	3072200

Total params: 3,104,136
Trainable params: 3,103,688
Non-trainable params: 448

Model: "Decoder"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 200)]	0
dense_6 (Dense)	(None, 64)	12864
tf.reshape_1 (TFOpLambda)	(None, 8, 8)	0
conv1d_transpose_3 (Conv1DTr	(None, 8, 64)	1600
batch_normalization_19 (Batc	(None, 8, 64)	256
conv1d_transpose_4 (Conv1DTr	(None, 8, 64)	12352
batch_normalization_20 (Batc	(None, 8, 64)	256
conv1d_transpose_5 (Conv1DTr	(None, 8, 32)	6176
batch_normalization_21 (Batc	(None, 8, 32)	128
flatten_5 (Flatten)	(None, 256)	0
dense_7 (Dense)	(None, 140)	35980

Total params: 69,612
Trainable params: 69,292
Non-trainable params: 320

Figure (2.1.6): Summary of the Conv1D Encoder-Decoder CNN architecture

3. Results

1. Preprocessing for Dimensional reduction

We use matplotlib's spy() function to examine how sparse the data frame is. This function plots a dark point for the non-zero cells and leaves corresponding zero-cell positions white. For the sake of visualization, we only plot the first 800 rows of the transformed dataset (22050 --> 240 columns) to show how there are no zero-elements in the dataset anymore.

We can observe from the above graphs in Figure (3.1.1) that the white space is eradicated from the plot after performing TSVD transformation, which confirms that the matrix is no longer sparse, and the dataset is ready for use.

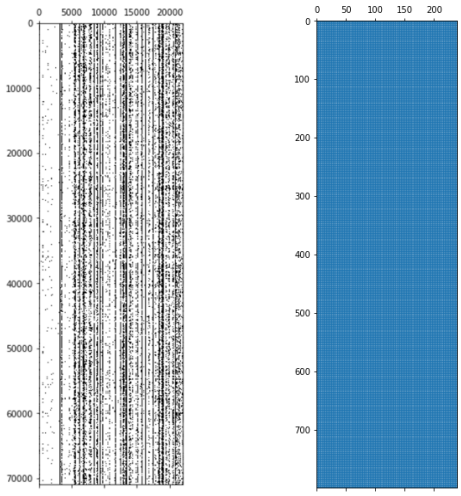


Figure (3.1.1): Before and after performing Truncated Singular Value Decomposition on the dataset.

2. Split data into train and test

We use the train_test_split function of the sklearn package in python to split the train dataset provided to us. We split the preprocessed train data into two subsets: train set (75% of the data), and test set (25% of the data). This is what we use for the rest of our modeling. The resulting dimensions are

- x_train: (53241, 240)
- y_train: (53241, 140)
- x_test: (17747, 240)
- y_test: (17747, 140)

3. Modeling

We run seven different models to predict data. To observe which model works best for the given data, we measure the differences between the models by comparing their root mean squared error (RMSE) as well as their R-squared value. We also take into account the best loss for the neural network models we implemented.

Root mean squared error (RMSE) is a measure of the quality of a predictive model. It is calculated as the square root of the average of squared differences between the predicted values and the true values. RMSE is used to evaluate the performance of a predictive model because it gives a sense of the magnitude of the error made by the model. A lower RMSE indicates a better fit of the model to the data.

The coefficient of determination, also known as the R-squared value, is a measure of the goodness of fit of a predictive model. The R-squared value describes how well the model can explain the variance

in the dependent variable based on the variance in the independent variables. A higher R-squared value indicates that the model is a better fit for the data and can explain more of the variance in the dependent variable.

Model	*Best Loss*	*RMSE*	*R-Squared*
KNN Regression	-	*2.1596*	*-0.069*
Ridge Regression	-	*2.0946*	*-0.01*
ElasticNet Regression	-	*2.0486*	*0.023*
Lasso Regression	-	*2.199*	*-9.24E-05*
Encoder-Decoder (240->120->60->30 : 30->70->140) NN	*2.9243*	*1.673*	*0.1682*
Encoder-Decoder (240->60 : 60->140) NN	*2.3624*	*2.06541*	*3.36E-05*
Encoder-Decoder (240->200 : 200->140) CNN (Conv1d)	***2.26418***	***1.66***	***0.2064***

Table 1. *Benchmark results of predictive models used*

The above table indicates that the convolutional network we built was successfully benchmarked against the other models as it resulted in the lowest RMSE (1.66) as well as the highest R-squared value (0.2) amongst the models implemented.

4. Analysis

Amongst all the models, we see that deep learning neural networks in general perform much better than the machine learning regression models as indicated by the lower RMSE values and higher R-squared values. Within regression models, we observe that ElasticNet performs the best with L1 regularization, which is in contrast to the Lasso regression model which solely functions on L1 regularization. We also note that the CNN Conv1d Encoder-Decoder we had constructed performs far better than the other Neural Networks models. This is because it is a deep learning model which is capable of learning features and trends in much more detail than a neural network.

5. Conclusion

We started off with the aim of using Multi-ome data's chromatin accessibility to predict gene expression and thereby predict surface protein levels using CITE-seq data, but due to limited memory resources, we had to narrow down the scope of our project to using CITE-seq data's gene expression to predict surface protein levels. This project was a great insight into understanding the need for preprocessing, being able to distinguish which compression method works in what scenario, and also understand how various models compare against each other based on certain parameters. We found that Truncated SVD worked best to compress our data because of how sparse it was, and also implemented 7 predictive models (4 regression, 2 neural networks, 1 Conv1d convolutional neural network) and found that amongst these 7 models, upon comparing the R-squared value, the deep learning conv1d model works the best in predicting the surface protein levels given gene expression data.

6. Github Repository Link (Source Code)

Attached below is the link to the github code repository for this project

https://github.com/anurag19997/Genomics_project

References

[1] Velten, L., Haas, S., Raffel, S. et al. Human haematopoietic stem cell lineage commitment is a continuous process. Nat Cell Biol 19, 271–281 (2017). https://doi.org/10.1038/ncb3493

[2]Aitken, Kyle, et al. "Understanding How Encoder-Decoder Architectures Attend." Advances in Neural Information Processing Systems 34 (2021): 22184-22195.

[3]Ye, Jong Chul, and Woon Kyoung Sung. "Understanding geometry of encoder-decoder CNNs." International Conference on Machine Learning. PMLR, 2019.

[4] https://openproblems.bio/neurips_docs/data/about_multimodal/

[5]https://www.kaggle.com/code/ravishah1/citeseq-rna-to-protein-encoder-decoder-nn/notebook#Predict-&-Submit

[6]https://www.kaggle.com/code/leohash/complete-eda-of-mmscel-integration-data

[7]https://blog.quantinsti.com/linear-regression-models-scikit-learn/

[8]https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

[9]https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b

[10]https://stats.stackexchange.com/questions/239481/difference-between-scikit-learn-implementations-of-pca-and-truncatedsvd

[11]https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955

[12]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D

[13]https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491#:~:text=What%20is%20the%20difference%20between,PCA%20skips%20less%20significant%20components.