August 7, 2023

INTRODUCTION

CITE-Seq is a method that allows the simultaneous measurement of gene expression and protein levels in a single cell. By combining single cell RNA sequencing (scRNA seq) and a method to bind an oligonucleotide 'barcode' to surface proteins using antibodies, it is possible to collect counts of RNA and a selection of proteins for a single cell. The method has broad applications from the identification of cell types to the analysis of immune responses. Stoeckius et al. (2018)

For this assignment, a data set consisting of 13953 gene expression (GEX) RNA counts and 134 antibody-derived tag (ADT) protein counts were provided for 66175 cells were provided with various quality control and data preproccessing steps already applied. In addition to preprocessing, the dataset was already divided into training and testing sets for the purpose of cross-validation. Finally, for each cell, additional information regarding that cell was provided including cell type, batch, percent of gene counts associated with mitochondrial genes, and estimated phase in the cell cycle.

The goal was to build a model that would given a particular cell's gene expression data would predict the same cell's ADT protein counts as accurately as possible. Given the high dimensionality of the GEX input and the fact that the gene counts are 'sparse' clearly, some form of dimension reduction was necessary. In addition, a 2D visualization of the gene expression dataset showed that observations were largely clustered together. In light of this clustering, I ultimately chose to use a model that first assigns each cell a class based on its gene expression and then fits a regression model for each class. I experimented with 2 classification models: KNN and Kmeans to separate the dataset into classes and then fit a linear ordinary least squares model for each class. Both of these methods performed better than a simple OLS model, however, the K-means model outperformed KNN model in spite of requiring less information to train.

METHODS

Preprocessing

The data provided had already undergone a number of preprocessing steps. Cells were filtered based on the percent of mitochondrial genes, gene counts and number of genes detected. Gene expression counts were also normalized with respect to the cell size calculated using scran. Finally, these normalized counts were log+1 transformed. These methods and a few others are described in Luecken and Theis (2019). I decided to also investigate some of the data correction methods discussed. Namely, I wanted to see if correcting for batch and biological effects could improve the accuracy of the basic linear model that was provided. To do this a regression model is fit using the variable being corrected to predict the associated gene expression then the residual of the predicted and actual gene expression is used. I used ScanPy's built-in 'regress out' function with respect to percent mitochondrial genes and dummy variables for cell phase and batch. Unfortunately, none of these improved the accuracy of the model so they were not used in my final model. I think that this wasn't effective for cell phase and mitochondrial genes since the effects that these factors had on gene expression also affected ADT protein counts. The tutorial also describes that if each batch has differing amounts of cell types then correcting for the batch will also somewhat correct for the effect that cell type has on gene expression. Comparing the visualization of observations based on cell type and then based on batch, I think that that is what happened here. It appears that each batch contained largely different cell types and that regression by batch then largely removed the effect that cell type had on gene expression which hurt rather than improved the model.

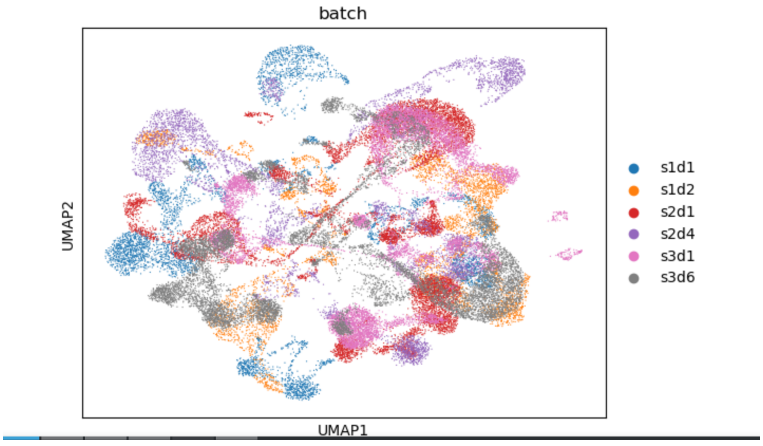

Figure 1: Dimension-Reduced visualization of gene expression colored by batch

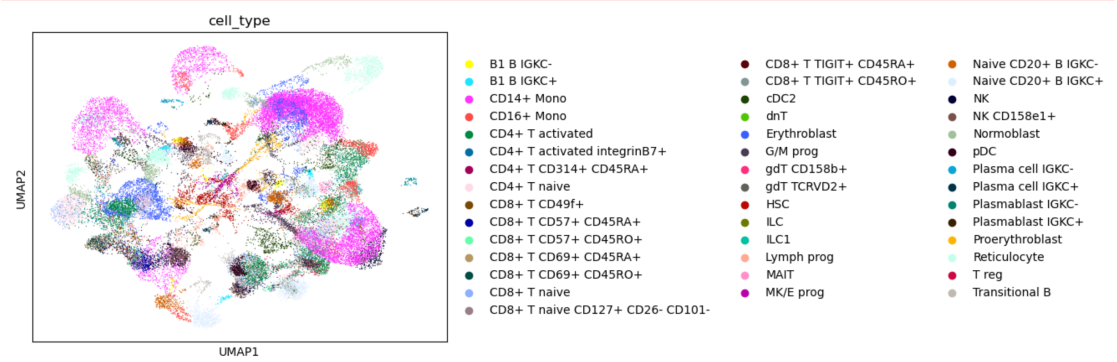

Figure 2: Dimension-Reduced visualization of gene expression colored by cell type

Dimension Reduction

Truncated SVD was used to reduce the dimensions of the gene expression data. It was chosen over PCA thanks to its relative efficiency over sparse matrices like the gene expression data. SVD refers to singular value decomposition where a matrix M can be rewritten as the product of 3 matrices ie. $M = U\Sigma V^*$. The center matrix Σ is a diagonal matrix containing the singular values of M while U and V^* contain the left and right singular vectors of M. The singular values are the square root of eigenvalues of the matrices $M * M^T, M^T * M$ while the left and right singular vectors are the associated eigenvectors with respect to these matrices. Using these we can make the SVD of the matrix M. In truncated SVD we take the n largest singular values and their associated singular vectors to create the best approximation of the original matrix with n features.

Classification

Upon seeing Figure 2 it seemed to me that each cell type seemed to have its own gene expression profile. I thought it might be possible that each cell type might have a unique relationship between GEX and ADT protein counts. I decided to test this idea by supposing we could build a perfect cell type classifier from the GEX data and then fitting a simple linear model for each cell type. I tested this, training a model for each cell type on the training subset and then using that model on the testing subset. I found that this method was significantly more accurate than just training a single model for all cell types together. I also found that a KNN classifier was able to correctly classify observations to their provided cell type from the dimension-reduced GEX data around 75% of the time..

K Nearest Neighbors

K nearest neighbors is a supervised classification algorithm that predicts the class of an observation by first calculating its distance from other observations. A wide range of distance equations can be used in this case I used the standard Euclidean distance $d = |\sqrt{(X-Y)^2}|$. Then the K observations in the training set with the smallest distance to the chosen observation are selected and the most common class label in these K observations is the predicted class. I fit a KNN classifier on the training GEX and provided class labels and then had it predict the class for each observation in the training set to get clusters of data to fit regression models on.

K-means

I was curious after completing my KNN model whether an unsupervised method like K-means might perform better. The size of the clusters my KNN model produced were very inconsistent in size since most of the cells in the dataset belonged to a few types and as a result, some of the clusters didn't even have enough observations to fit a regression model. Others were large enough that I thought that multiple regression models might be able to make better predictions. So I adapted my code for the KNN model to work with K-means (with the help of Chat-GPT). It works similarly to KNN but since it is unsupervised it does not require the class labels. Instead, it starts by selecting k observations these observations become the initial clusters. For the remaining points, the mean of each cluster is calculated, then the observation that is nearest by some distance formula (in this case Euclidean again) to one of those means is added to the cluster whose mean it is nearest to. Once all the observations are assigned it then reassigns observations to different clusters if the mean of another cluster has shifted to be nearer to that observation than the cluster the observation originally belonged to. Once no more points can be reassigned the algorithm is complete. The implementation I used in SKlearn has some tricks to reach convergence quicker like picking optimal starting points but the concept remains the same. We ultimately end up with k clusters with minimized variance inside each cluster.

Regression

For each predicted class by the chosen classifier, a separate Ordinary Least Squares model was fit on observations for that class. Ordinary Least Squares works by fitting a line that minimizes the sum of squared residuals. It ultimately results in the following prediction equation.

$$y_i = \beta_{i0} + \sum_{n=1}^{n} \beta_{in} x_n$$

Since the target y in this case is multidimensional each linear regression model has $m \times (n+1)$ coefficients where m is the number of targets y has and n is the number

of features x has (after dimensionality reduction). In some cases, particularly in the model using the KNN classifier, the number of observations in a particular predicted class during training was lower than the number of features. In this case, I had the model simply predict the mean value within that class. I also experimented with using the polynomial features transformer in SKlearn to fit higher-degree models, however, this did not improve accuracy in any cases. I fit individual models for each class since I found it easier to implement but it would be possible to implement an identical model using the predicted class as dummy variables if you define

$$\omega_k = \begin{cases} 0 \text{ if the current observation} \notin \text{the ith predicted class} \\ 1 \text{ if the current observation} \in \text{the ith predicted class} \end{cases}$$

. Then we could express all the models as a single equation:

$$y_i = \sum_{k=1}^{k} \omega_k * m_k$$

where

$$m_k = \begin{cases} \beta_{ik0} + \sum_{n=1}^{n} \beta_{ikn} * x_n \text{ if n\_obs in the kth class} \geq \text{ n\_features} \\ \mu_{ik} \text{ if n\_obs in the kth class} < \text{n\_features} \end{cases}$$

Where μ_{ik} is the mean value for y_i within the class and n_features is after any dimension reduction or polynomial feature transformation.

RESULTS

After finding ideal hyperparameters for each model, the KNN model was able to achieve an RMSE of 3.64 on the test set, while the K-means model achieved 3.58. The KNN model performed best when using 55 features and 22 neighbors to classify observations while the K-means model performed best with 115 features and generating 13 clusters. There are a few factors that might have led to K-means outperforming the KNN model. First of all, it seems to indicate that there is not a particularly unique relation between GEX and ADT counts for each cell type. Rather that cells with similar GEX counts had similar ADT counts. Also, it seems likely the extreme imbalance in class size that KNN classifier produced gave it a disadvantage compared to K-means. Also, K-means performed better at a higher number of features than KNN, likely partially due to more features for KNN meant more classes that could only predict the mean rather than a full linear model. I think that the KNN could be improved by reducing the number of classes by grouping some of the cell types.

CONCLUSION

I enjoyed this project a lot. I wanted to try to explore avenues that I thought other students might not explore. In some cases, it didn't work out like with the preprocess-

ing steps that I investigated. But my per-class regression model did improve accuracy over the baseline linear model provided. Similar accuracy to what I achieved could probably be obtained with a neural network or some other simpler flexible regression model, but using the output of a classifier to fit multiple regression models was new to me and I found it fun to code, even though it pretty difficult to iron out all the bugs initially.

REFERENCES

Luecken, M. D., and F. J. Theis, 2019, Current best practices in single-cell rna-seq analysis: a tutorial: Molecular Systems Biology, **15**, e8746.

Stoeckius, M., S. Zheng, B. Houck-Loomis, S. Hao, B. Z. Yeung, W. M. Mauck, P. Smibert, and R. Satija, 2018, Cell hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics - genome biology.

(ChatGPT was used during the debugging stage of coding and to convert my KNN code to K-means. I also occasionally asked it to help me summarize some concepts so I could think about what aspects were important to explain)