MMG1345H: Foundational Computational Biology (FCB) II **Winter 2022**

Assignment 2:

Part 1: Dimensionality reduction

In this part, you will play with some dimensionality reduction methods described in Prof. Campbell's lecture, applying them to a single-cell RNA-seq (scRNA-seq) dataset (Villani *et al.* 2017) from human blood cells. Specifically, the scRNA-seq gene expression data (in units of 'transcripts per kilobase of transcript length per million reads'; or TPM) which includes 22,430 genomic features (e.g. genes, lncRNAs) from 1,078 human dendritic cells (DCs) and monocytes.

Dendritic cells (DCs) and monocytes are broad cell types that each encompasses multiple specialized cell subtypes playing key roles in pathogen sensing, phagocytosis, and antigen presentation. However, the identities and interrelationships of these subtypes are not fully understood, as they have historically been defined by a combination of morphology, physical properties, localization, functions, developmental origins, and expression of a limited set of surface markers. Using scRNA-seq, together with follow-up profiling and functional and phenotypic characterization of prospectively isolated subsets, the authors identified and validated six DC subtypes and four monocyte subtypes, and thus revised the taxonomy of these cells.

Here, you will apply different dimensionality reduction methods to visualize the scRNA-seq data in two dimensions. Because the purpose of this assignment is to help you develop a holistic view of various dimensionality reduction and machine learning approaches, **you may use any Python or R packages you prefer**. A table of recommended packages (Table 1) is provided for your consideration. All input files are zipped and stored in the *input.zip* file.

0) Data preparation

a. Load the gene expression data matrix, *M*, from the tab-delimited text file *expression_matrix.txt*. Each row of the matrix *M* is a genomic feature (e.g. gene, lncRNA) and each column of the matrix is a cell.

b. Load the cell labels table from the tab-delimited text file *cell_labels.txt*. The first column, "NAME", contains the cell names that are also used to label cells (as column names) in the data matrix *M*. The second column, "CellType", contains the six DC subtypes and four monocyte subtypes identified by Villani *et al.* Use the "CellType" column to determine the "true" cell type of each cell.

1) Principal component analysis

a. Apply Principal Component Analysis (PCA) and plot the projected data points on the two principal components (PCs) which explain the most variance of the input data. You do NOT need to scale the data (e.g. do not normalize the data to have mean = 0 and

variance = 1) In the plot: 1) include the variance explained in the axis labels, i.e. if your first PC explains 50% of the variance and is shown on the x-axis, your x-axis should be labelled as PC1 (50%), 2) colour each datapoint (cell) with the "true" cell type that the data point represents, and 3) summarize the cell types and the associated colours in the legend of the plot. Your PCA plot should look similar to the image *sample_pca.png* in the assignment folder.

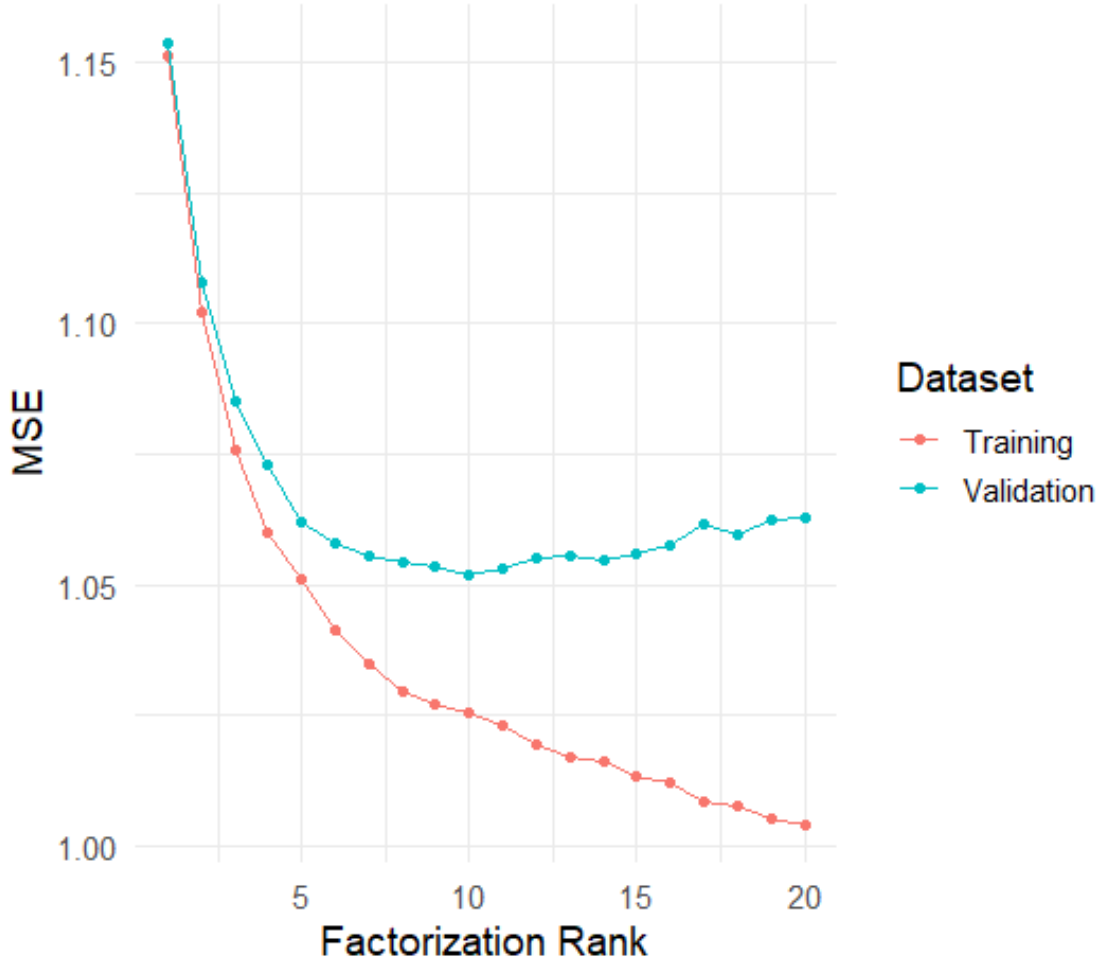
b. Generate a scree plot of the first 10 principal components, ranked again by the variance explained in descending order. How many principal components contribute more than 1% to the variance explained? How much of the variance is explained by the first 10 PCs? Do you think the total variance explained by the first 10 PCs is high enough? If you think the total variance explained is not high enough, what could be one reason causing the low explanatory power of the PCA?

c. One common pattern you will find in many gene expression publications which use PCA to evaluate gene expression data is that only the high variance genes (i.e. genes with high variance on expression across samples) are subject to the analysis. Here, we explore the effect of such a feature selection process. Repeat steps 1 and 2 with the top 500 high variance genes and compare the results and plots to the ones you generated in steps 1 and 2: 1) Does PCA perform better when restricted to only the top 500 high variance genes? 2) If the PCA does perform better, what could be a reason explaining this phenomenon?

b) Nonnegative matrix factorization

1. Apply Nonnegative matrix factorization (NMF). Please be aware that NMF will take a significant amount of time to compute, so you may want to debug your code with a smaller data set, e.g. a subset from the data, to save time.

2. Recall that NMF factorizes the data matrix *M* (A×B) into two smaller matrices *W* (A×*r*) and *H* (r×B), a critical parameter in NMF algorithms is the factorization rank, *r*. It defines the number of basis effects used to approximate the target matrix. To help you pick the optimal *r*, we randomly hid 30% of the data matrix M to the NMF algorithm, creating a validation set which was later used to evaluate a model's performance with rank *r*. This operation was achieved by masking the randomly chosen 30% of the M as NA, effectively creating a sparse matrix which is accepted by the *nnmf()* function in the R package NNLM. **Why do we have to randomly hide data from the matrix rather than hold out entire cells or genes?** After the NMF modelling, the reconstructed matrix (*W* × *H*) was then compared to the training set (the unmasked 70% of the matrix *M*) to compute the mean squared error (MSE) for the training set and compared to the validation set (the previously masked 30% of the matrix *M*) to compute the MSE for the validation set. The figure below summarizes the MSEs for training and validation sets generated for models with rank (*r*) parameters ranging from 1 to 20. **Which *r* was the best and why?**



3. Try three random initializations. Generate a bar plot on the variance explained by each of the NFM runs. Do you see a difference in terms of the variance explained?

4. Generate a heatmap to visualize the *H* matrix. Do you see any patterns that correlate with cell types? If you do, please provide two examples.

c) t-distributed Stochastic Neighbor Embedding

1. Apply t-distributed Stochastic Neighbor Embedding (t-SNE). Please be aware that t-SNE will take a significant amount of time to compute, so you may want to debug your code with a smaller data set, e.g. a subset from the data, to save time.

2. Try three different perplexity values (2, 4 and 12) for t-SNE and report on the differences. What does perplexity tell about the embeddings?

3. Try three random initializations and report the difference.

4. Show samples in the reduced data space (i.e. the two dimensions) coloured again by the cell types.

d) Compare and contrast

1. Discuss which method is more suitable for the kind of data you are working with. How do different dimensionality reduction methods affect the data separation visually? Which method performed best at separating the DC monocyte subtypes and why?

Part 2: Machine learning

You will apply different classification algorithms. You will work with the MNIST (Modified National Institute of Standards and Technology) database, which is a large database of handwritten digits (Fig. 1) that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

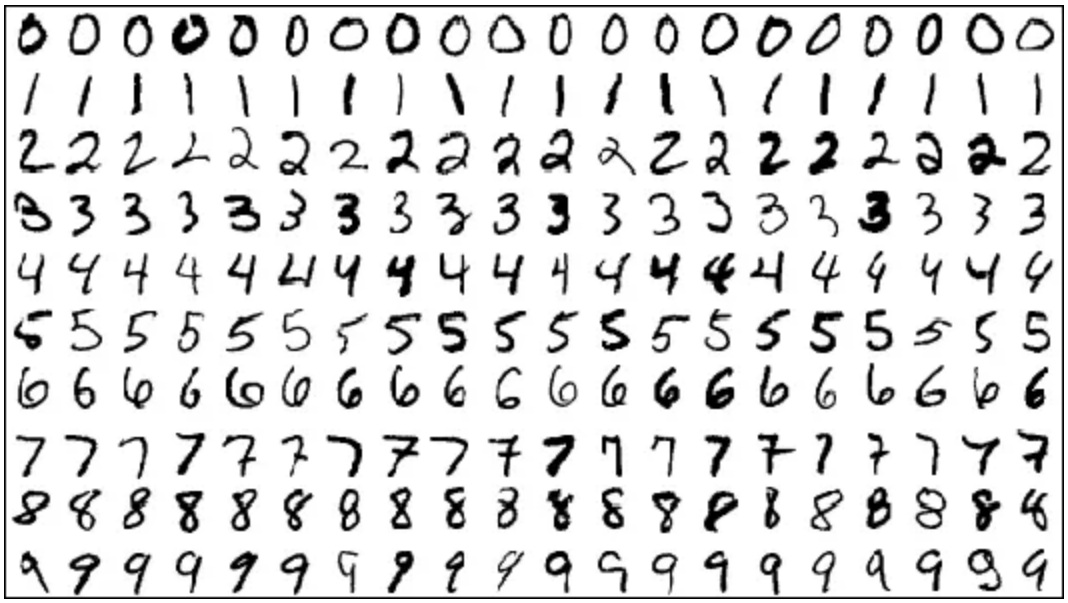


Fig. 1. Sample images from MNIST test dataset.

The database consists of 55,000 training images and 10,000 test images. Each image has a size of 28 by 28 pixels, but the input data provided to you is a flattened array of the pixels, rows are concatenated into a 784 vector. Therefore, the training data (*mnist_train.csv*) size is 55,000 by 785 where the first column is the label of the digit of each image, and the testing data (*mnist_test.csv*) size is 10,000 by 785.

You will train classification algorithms on the training dataset and use the test dataset to report the performance. Notice that this is a multi-class problem so you should report both the average accuracy across classes (one number), as well as provide a 10-by-10 confusion matrix on your test set. You should also report the training time of your classifiers. **Again, you may use any of the available machine learning packages in Python or R.**

1. Apply a multinomial logistic regression algorithm with no, L1 or L2 regularization separately. What is your overall performance? How do different regularizations affect your performance? You should report three sets (no regularization, L1, L2) of performance metrics.

2. Apply a multiclass support vector machine algorithm with a linear or radial basis function (RBF) kernel separately. What is your overall performance? How do different kernels affect your performance? You should report two sets (linear or with RBF kernel) of performance metrics.

3. Apply a fully connected neural network with a single hidden layer of 800 units. Your neural network should have this structure:
    a. Input layer: 784 units, representing 784 pixels for each image,
    b. Hidden layer: 800 units
    c. Output layer: 10 units, representing the 10 digits from 0 to 9.

    Please use ReLu as the activation function for the input and hidden layers and softmax for the output layer. What is your overall performance with and without regularization (L1, L2 = 0.001)? How do different learning rates (0.001, 0.01 and 0.1) affect the model's performance and runtime? You only need to report performance metrics for the best performing condition.

4. Apply a convolutional neural network, the architecture is up to you. Notice that the more layers you add, the longer it will take to train. Please briefly describe your setup (e.g. number of layers, number of units in each layer, activation functions, learning rate and regularization) in the PDF write-up. Are you able to beat the other three methods? You only need to report one set of performance metrics.

Pro-tip: When it comes to machine learning, there are many great packages available. Keras (Table 1) is one of the most user-friendly options with intuitive APIs and extensive developer guides and documentation.

Submission

Please submit a zip file (first_lastname.zip) with

1. Program source code (in .R or .py format).
2. A README file on how to install necessary packages and run your code.
3. Output plots. You may embed them in your PDF write-up.
4. A PDF write-up.

Table 1. Recommended R and Python packages.

Package	Platform	Relevant Features
Stats	R	PCA
kernlab	R	Support Vector Machines (SVM)
fastICA	R	Independent Component Analysis (ICA)
NMF, *NNLM*	R	Nonnegative Matrix Factorization (NMF)
Rtsne	R	t-SNE Analysis
scikit-learn	Python	PCA, NMF, t-SNE Analysis, SVM
keras	R, Python	SVM, Logistic Regression

References

Villani, Alexandra-Chloé, Rahul Satija, Gary Reynolds, Siranush Sarkizova, Karthik Shekhar, James Fletcher, Morgane Griesbeck, et al. 2017. “Single-Cell RNA-Seq Reveals New Types of Human Blood Dendritic Cells, Monocytes, and Progenitors.” *Science* 356 (6335). https://doi.org/10.1126/science.aah4573.