# Classifying Single-Cell Types from Mouse Brain RNA-Seq Data using Machine Learning Algorithms

Hankyu Jang and Samer Al-Saffar

## Abstract:

The cerebral cortex carries the cognitive and sensory functions of the mammalian body as well as any social behavior. Normal brain function depends on a variety of differentiated cell types, such as neurons, glia, and vasculature. Definitive identification of these cell types has proven to be more difficult than other cells of the mammalian body due to their high specialization. Here we attempt to classify cell types from RNA-seq data obtained from brain cells of mice using machine learning approaches. We are using four different supervised machine learning algorithms, and our goal is to determine which one is the best for classifying untagged cells.

*Keywords: Single-Cell RNA-Seq | Support Vector | Random Forests | KNN | Neural Network*

## Introduction:

The brain is the most complex organ in all vertebrates and most invertebrates, its cortex carries out highly specialized cognitive functions of the nervous system. It consists of highly differentiated cells that run all the electrophysiological functions besides regulating other demands of the brain, such as nutrients and energy supply, waste disposal, immunity needs, and also serves as a

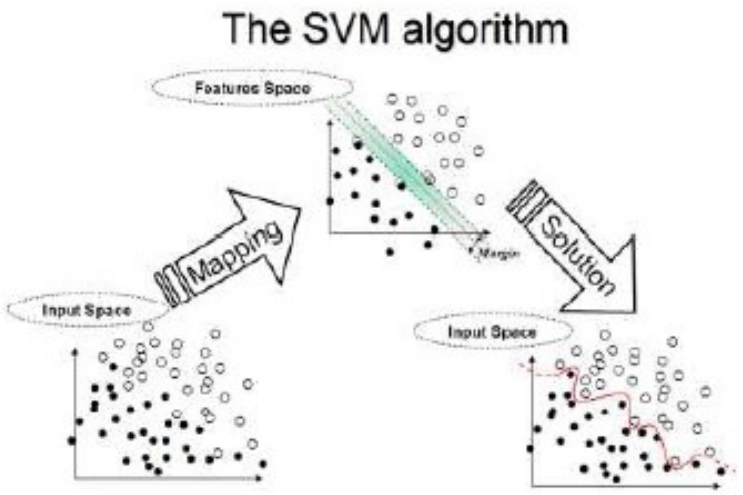
protective framework. While these cells have all been classified according to their location, function, molecular markers, morphology, and heterogeneity, there is no classification system where one cell can be definitely identified under all studied properties (1). A more recent approach for classification is to utilize the advancement in single-cell biology, and particularly when applying RNA-seq techniques, as these proved to be a powerful discriminatory tool to differentiate cells types (2). This method has been successfully applied to characterize cells in other organs. The brain, however, exhibits more complexity and variety of cells and thus it is still a challenge to classify these cell types both experimentally and computationally.

The field of machine learning has been widely utilized in the many applications of genetics and genomics, a review of these can be found in reference (3). In this study, we attempt to classify brain cortex cells in mice by applying different machine learning algorithms on quantitative RNA-seq data obtained from a previous study (1). We used four supervised algorithms with labeled training data set and unlabeled test data set. Our goal is to test the suitability of these different algorithms in obtaining the "best" classification of cell types. The four algorithms utilized in this study are: Support Vector Machine using Radial Basis Function Kernel, Random Forests, K-Nearest Neighbor, and Multi-Layer Perceptron (Neural Network).

**Support Vector Machines (SVM) Algorithm:**

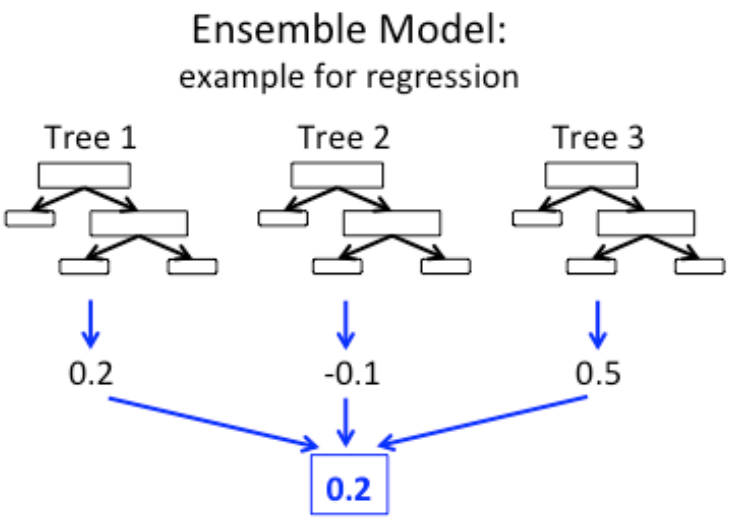
One of the simplest and most common algorithms used in machine learning, it was first developed in the 1960s and later got popularized and extensively adopted in the 1990s when it further got improved, it is still widely used today when performance is needed without much adjustment (4). SVM are supervised machine learning methods used for classification and regression analysis. It can efficiently perform both linear and non-linear classification. In practice, the SVM algorithm

functions using kernel classifiers. There are mainly three kinds of kernel SVMs: linear, polynomial, and radial kernel, the last was used in this study. By using the different types of kernels, the classifiers transform the original data according to the kernel type. Using the transformed dataset SVM can find non-linear decision boundary. Here radial kernel transforms the data using a Gaussian kernel that may give better classification result.




**Random Forests Algorithm:**

Random forest is perhaps the most powerful and most popular machine learning algorithm and has gained huge popularity in machine learning applications during the last decade due to their good classification performance, scalability, and ease of use (6). A random forest can be seen as an ensemble of decision trees (a non-parametric supervised learning method used for classification and regression). The idea behind random forest is to combine weak learners to build a more robust model, a strong learner, that has a better generalization error and is less susceptible to overfitting. This unique combination of tree classifiers provided the random forest classifier with prediction accuracy and model interpretability that is superior to other machine learning methods.



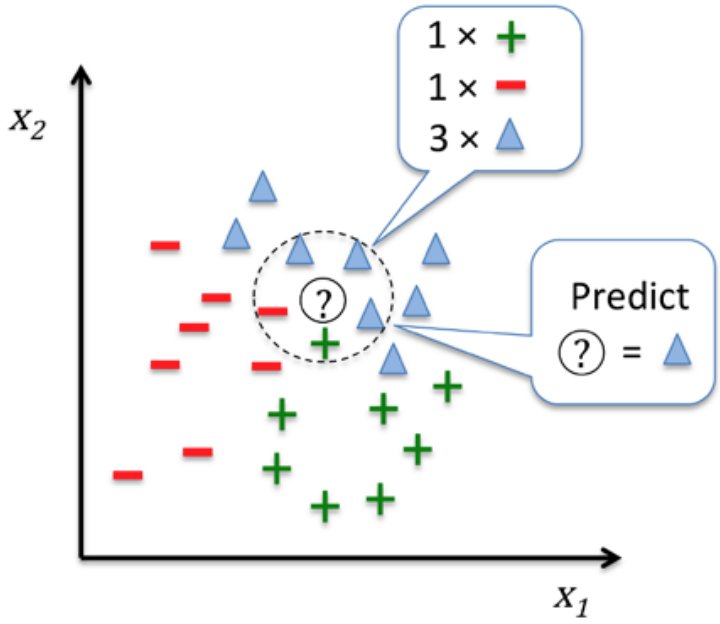

The random forest algorithm can be summarized in four simple steps: 1) Draw a random bootstrap sample of size *n* (randomly choose *n* samples from the training set with replacement). Bootstrap is an ensemble technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model; 2) Build a decision tree from the bootstrap sample. At each node: Randomly select *d* features without replacement. Then split the node using the feature that provides the best split according to the objective function; 3) Steps 1 and 2 are repeated for *k* times; 4) Aggregate the prediction by each tree to assign the class label by that has been predicted by the majority of classifiers (7).

**K-nearest Neighbor's Algorithm** (**KNN**):

KNN is fundamentally different from the other supervised learning algorithms. It is a typical example of a lazy learner. It is called so not because of its apparent simplicity, but because the model representation for KNN is the entire training dataset; it has no model other than storing (memorize) the entire dataset, so there is no learning required. The KNN algorithm itself is fairly straightforward and can be summarized by the following steps: 1) Choose the number of *k* and a

distance metric; 2) Find the *k* nearest neighbors of the sample that we want to classify; 3) Assign the class label by majority vote.

The following figure illustrates how a new data point (**?**) is assigned the triangle class label based on majority voting among its five nearest neighbors.



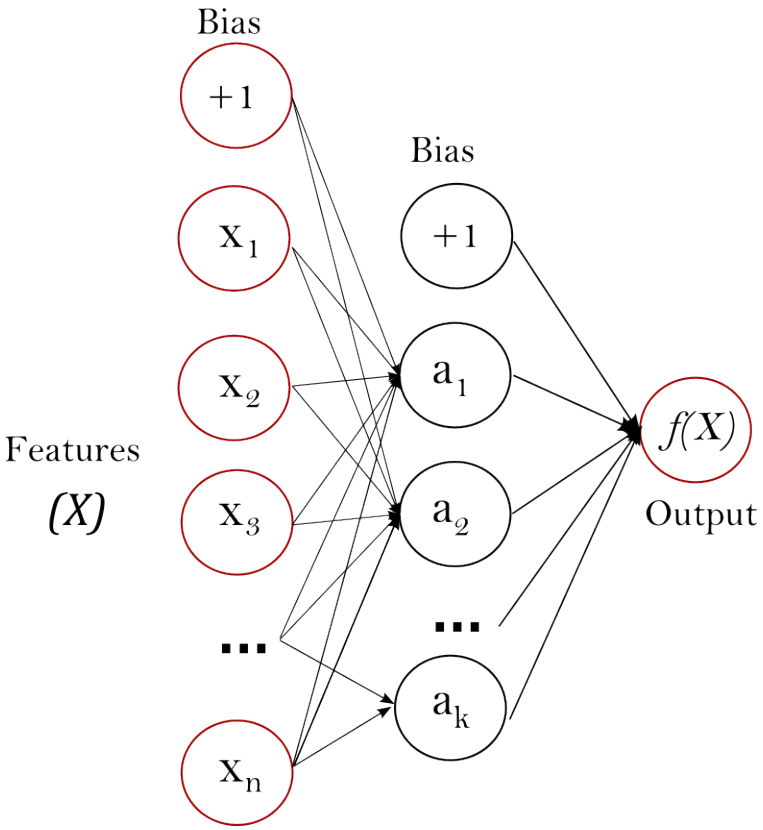

Based on the chosen distance metric, the KNN algorithm finds the *k* samples in the training dataset that are closest (most similar) to the point that we want to classify. The class label of the new data point is then determined by a majority vote among its *k* nearest neighbors (7).

**Multi-Layer Perceptron (Neural Networks) Algorithm:**

A perceptron is a single neuron model that was a precursor to larger neural networks. Multi-layer Perceptron is the most popular feedforward neural network model. They are currently one of the most active research topics in the machine learning field. The power of neural networks come from their ability to learn the representation in a training data and how to best relate it to the output variable that we want to predict. In this sense, neural networks learn a mapping. Mathematically,

they are capable of learning any mapping function and have been proven to be a universal approximation algorithm.

Multi-layer Perceptron is a supervised learning algorithm that learns a function by training on a dataset, with a number of dimensions for input and a number of dimensions for output. Given a set of features and a target, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.




The predictive capability of neural networks comes from the hierarchical or multi-layered structure of the networks. The data structure can pick out (learn to represent) features at different scales or resolutions and combine them into higher-order features. For example, from lines to collections of lines to shapes (8).

## Methods:

The analysis is carried out using Python version 2.7 with all the machine learning tools being imported from the scikit-learn library (http://scikit-learn.org/stable/), this library uses SciPy, NumPy, Biopython, and matplotlib libraries to implement training algorithms and data analysis tools. Furthermore, the scikit-learn library has many convenient functions to pre-process data and to fine-tune and evaluate different models.

The data used for the analysis are stored in two separate files:

- GSE60361C13005Expression.txt. This file contains the RNA-seq raw data, arranged in columns (cells) and rows (gene expression levels). It covers approximately 3000 cells with 20,000 genes.
- expressionmRNAAnnotations.txt. This file contains the annotation data, such as molecule count and cell type classification.

The following programs were used to carry out the analysis:

- main.py: This file processes the command line arguments supplied and runs the program using the selected classifier. Based on a user input, the program decides whether to down sample, cross-validate, and which classifier to use. This file defines the four classifiers supplied to the user for classification: Support Vector Machine using Radial Basis Function Kernel, Multi-Layer Perceptron (Neural Network), K-Nearest Neighbor, and Random Forest. After calling the pre-process code and running the classification, this class sends the results to an analysis file that performs evaluations and writes the results to file.
- RNASeqData.py This class object represents the RNA-Seq Data. It holds the raw data, the annotations, and provides methods for partitioning the data. The partitions (for both down sampling and non-down sampling and cross-validation and no cross-validation) randomly

make partitions of the data for both training and testing, while simultaneously holding the annotations for the randomly selected testing data. The class also provides accessor methods for all data, annotations, training data, testing data, and training data target values to evaluate performance.

- preprocess.py This file does all of the preprocessing work before running classification. This file loads raw data and annotations into memory. Next, this file is used to down sample by both cluster size and molecule count.
- analysis.py After classification, this file is used to evaluate the performance of the classifier and write the results to an output file. First, this file creates a confusion matrix and then computes the accuracy, sensitivity, specificity, MCC, and F1 Score at both the class level and the global level across the supplied number of folds (10 folds for cross validation and 1 fold for non-cross validation). This class also uses a basic metric of merely counting the number of correct classifications that was used initially to check the performance of the classifiers. After evaluating, the results are written to a file.
- knn_RNASeq.py This file defines the K Nearest Neighbor classifier. It allows the user to specify the number of neighbors used and then fits the training data and the samples to the classifier. Then, it takes training data and makes predictions, returning the results of the predictions.
- neuralNetwork_RNASeq.py This file defines the Multi-Layer Perceptron (Neural Network). It fits the training data and the samples to the classifier. Then, it takes training data and makes predictions, returning the results of the predictions.

- randomForest_RNASeq.py This file defines the Random Forest Classifier. It fits the training data and the samples to the classifier. Then, it takes training data and makes predictions, returning the results of the predictions.
- rbfSVC_RNASeq.py This file defines the Support Vector Machine using a Radial Basis Function Kernel. It fits the training data and the samples to the classifier. Then, it takes training data and makes predictions, returning the results of the predictions.

## References:


1. Zeisel, A. *et al*. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science* **347**, 1138-1142 (2015).
2. Sandberg, R. Entering the era of single-cell transcriptomics in biology and medicine. *Nature Methods* **11**, 22–24 (2014).
3. Maxwell W. Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics* **16**, 321–332 (2015).
4. Vapnik, V. The Nature of Statistical Learning Theory. Springer (1995).
5. Scikit-learn documentations. http://scikit-learn.org/stable/.
6. Leo Breiman. Random forests. Machine Learning, 45:5–32 (2001). Retrieved from https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf.
7. Raschka, S. Python Machine Learning. Packt publishing (2015).
8. Trevor, H. et al. The Elements of Statistical Learning, 2nd edition. Springer (2009).