Machine Learning Approaches on High Throughput NGS Data to Unveil Mechanisms of Function in Biology and Disease

Pezoulas VC[+], Hazapis O[+], Lagopati N[+], Exarchos TP, Goules AV, Tzioufas AG, Fotiadis DI, Stratis IG, Yannacopoulos AN*, Gorgoulis VG*
+ equally contributed, * corresponding authors

Supplementary Material

1. Other machine learning strategies

1.1 Online learning

Online learning is a widely used machine learning (ML) strategy which aims to update a training model according to a cost function that is constantly updated on newly coming data. The online learning approach uses stochastic optimization methods to update an existing model on upcoming training samples by minimizing a global cost function, where the model can be either updated on an individual upcoming sample or on a series of accumulated samples (known as mini-batch processing) (Saunders et al. 2019). The minimization of the empirical risk is given as:

$$I(f) = E[V(f(x), y)] = \int V\,(f(x), y)dp(x, y) \quad (1)$$

, where usually a training sample of $x_i, y_i$is extracted from a distribution p(x,y). Function f is a loss function where either least square or support vector machines with linear kernels can be used. The online learning will be based on only new incoming data but also use stored information independent of the training dataset.

Hybrid online learning approximation methods can be based on non-linear kernels, as well as use of stochastic gradient descent (SGD) approaches for convex optimization. This aspect can be useful when considering many formulations where online learning is not applicable due to the fact that one needs to store all previous data points. One solution could be to use mini-batches techniques while using permutations over the training points. For such purposes SGD with back propagation can provide a valuable tool.

1.2 Incremental learning

Another interesting approach that shares many similarities with the online learning, is incremental learning (Ganguly et al. 2019), which tries to adapt a continuous data model on constantly upcoming data streams. In contrast to online learning, incremental learning does not always need to be executed in an "online" manner or fashion. More specifically, incremental learning is a batch processing method that is used to train a supervised learning model on an initial batch (data stream), and then adjust the existing model on a series of upcoming batches, due to its ability to solve additive cost problems. This makes incremental learning, ideal in the case where large-scale data need to be processed "offline", i.e., when the data are already stored in distributed databases. According to the literature, existing implementations of incremental learning methods are based on SGD for convex optimization, stochastic gradient boosting (Friedma et al. 2002) based on ensemble classifiers for achieving high performance, and Multinomial Naïve Bayes, among others.

1.3 Meta learning

Meta learning is a rigorous category of ML strategies where individual classification outcomes (metadata) are collected from the training of multiple classifiers on the same data and are finally combined to reduce the computational complexity of the incremental learning or online learning

process offering some kind of parallel execution. Such methods, however, suffer from biases that are introduced during the assembly stage where the classification outcomes from different classifiers are combined.

1.4 Instance based learning

Perhaps the simpler of the two approaches is instance-based learning. It is based on the solutions of previous instances (problems) in order to provide outcomes for new inputs. This is achieved by producing predictions based on the similarity (distance) of a new input to its nearest neighbors in the training dataset. This in turn implies that all known instances are stored in memory for use. In this approach the generalization is explicit, no abstract models are involved in the process. This can lead to more adaptive generalizations, given that the implementation can simply store a previously unknown instance for future use. On the other hand, instance-based learning may lead to increased computational complexity due to the need of storing in, large volumes of instances. A learner of this category may also prove susceptible to data noise and overfitting.

1.5 Model based learning

This approach aims to create internal knowledge representations (abstractions) based on raw inputs. A model-based learner attempts to construct and consequently refine its model of the environment it operates, in order to deduce a set of underlying properties which in turn are to be used for producing predictions when new/unknown data are inserted. In this scenario, direct interactions with the environment (fundamentally represented by the set of inserted raw inputs) is minimized in comparison to instance-based learning. This may lead to faster learning sessions in some cases and may also produce more robust learning paradigms.

2. Preprocessing for improving the quality of biomedical data

2.1 Data curation

2.1.1 Data quality dimensions

Many studies have been launched, highlighting the emerging role of data quality control in different domains, including the biomedical domain. Prior to the assessment of the quality of the data one must first define the data quality metrics, including the : (i) accuracy, (ii) conformity, (iii) completeness, (iv) relevance, (v) clarity, (vi) accessibility, and (vii) validity, among others. For example, accuracy denotes the act of valid robust classification among repetitive regimes, relevance shows the compliance of the data according to the requirements for describing the knowledge of a particular clinical domain under investigation. The increasing size of daily generated medical data has led to the development of automated data quality control strategies that overcome the huge amount of time needed by the clinical and data experts for manual curation, such as, outlier detection, inconsistent data types and formats, missing values, duplicated variables, and any other data recording errors.

2.1.2 Outlier detection

The detection of values that deviate from standard population distribution is of great importance. These values are known as outliers. According to the literature, a variety of both univariate and multivariate outlier detection methods have been proposed. Examples of univariate methods include the z-score and the modified z-score which seek for values whose distance is larger than 3 times the standard deviation from the mean, i.e., values with z-scores larger than 3 or lower than -3, the interquartile range which seeks for values that are higher than the third quartile range or less than

the first quartile range, the Grubb's statistical test (Aslam et al. 2020) which tests for the hypothesis that the data include outliers, the median absolute deviation (MAD) which controls for the biases in the z-score, among many others. Examples of more straightforward methods, i.e., multivariate methods, include the isolation forests (Liu et al. 2008) which serves as a supervised learning method that focuses on the isolation of outliers by generating partitions around a randomly selected data point to form regions that isolate the outliers from the normal data points, and the local outlier factor which estimates the local density for a given data point with its neighboring data points and indicates regions in the data where the density is low or high, where regions with low density correspond to outliers. The isolation forests algorithm can also be trained on data including outliers to detect them on new data streams.

2.1.3 Data imputation

Often characterized as rigorous, data imputation is a computational approach that deals with missing data by filling the missing values to reduce the overall information loss. This attempt is particularly useful in the case of longitudinal data, where the existence of data across multi timepoints might be absent. Widely used data imputation strategies include the imputation with the mean value in the case of missing continuous values (e.g. numerical values) and with the median or the most frequent value in the case of missing discrete values (e.g. categorical values). One such example such as K-nearest neighbor (KNN) (Altman 1992) is an example of neighbour-based imputation which will use the most frequent observation among the k neighboring. Another imputation technique uses interpolation of observed values. Such imputation approaches focus on the development of regression models such as, regression trees and linear/logistic regression models to predict the missing values by training the algorithm on the given set of non-missing values for different variables. Gaussian processes can also be used for fitting an interpolant within a Gaussian data distribution or *Hermitte interpolation, via interpolating the* data points *as a* polynomial function. At this point it is important to note that outlier detection must be performed prior to the application of data imputation to avoid any data contamination.

2.1.4 Data standardization/harmonization

Data standardization refers to the normalization of data values according to a pre-defined set of criteria that include standard data ranges and data types. It is a part of an emerging field in data science known as data harmonization which aims to overcome the barriers that obscure the interlinking of medical databases, such as, structural heterogeneities. To do so, a gold-standard data model is usually defined by the clinical experts on a field of interest (e.g. on breast cancer) which consists of a set of variables (laboratory tests, therapies, demographics, patient history, etc.) which sufficiently describe the domain knowledge of the field. For each one of the variables within the gold-standard data model, a particular data type (e.g. categorical) along with a value range (e.g. [0, 1] where "0" denotes the absence and "1" denotes the presence of a condition) are assigned for data standardization purposes. Data harmonization, however, does not end there. It also focuses on the detection of variables among heterogeneous data structures that present similarities with those from the gold-standard model. To do so, lexical matching and semantic matching methods are applied to detect lexically similar data names (e.g. "gender" and "sex"), as well as variables that are conceptually similar (e.g. variables that describe the same concept).

Other means of data harmonization can be smoothing techniques such as Averaging Methods or Exponential Smoothing Methods. As an example, the simplest algebraic formula for a simple exponential smoothing can be defined as $I(f) = E[V(f(x), y)] = \int V(f(x), y) dp(x, y)$. X_t is the current observation and α is the smoothing coefficient. A very well-known procedure is the moving average technique where the formula for a simple moving average can be given as:

$$x_i, y_i \quad (2)$$

where y is the variable and t is the current time period of observing the variable while n is the number of time periods in the average.

2.1.5 Data discretization

Data discretization is widely used to discretize continuous variables in order to deal with potential data recording errors during the data collection process. To do so, data discretization is usually based on the assignment of data values into a pre-defined number of bins (i.e., the Equal frequency method) or into approximately equally sized instances (i.e., the Equal Width method). A more straightforward approach for data discretization is the minimum description length algorithm (Bouckaert 1993) which recursively splits the vector values to maximize the information gain in entropy, until the latter is lower than a threshold. In fact, the algorithm evaluates the information gain for all possible splitting values and picks the one that maximizes the information gain (Azhagusundari and Thanamani 2013):

If $y=\{x_1,x_2,x_i,\ldots..x_n\}$ then the information gain is

$$IG(x|y) = H(x) - H(x|y), \quad (3)$$

where:
$H(x)$ is the average rate of existing information in , i.e. the entropy:

$$\overline{y_t} = \frac{y_t + y_{t-1} + y_{t-n-1}}{n} \quad (4)$$

and
$H(x|y)$ is the conditional entropy of given the output vector :

(5)

where is the fraction of the data point x_i across the vector, is the conditional probability of data point given the data point as the fraction of all the data points that have the value .

2.2.1 Feature selection and feature importance

Based on the above ideas when choosing to train a classifier it is important apart from the data normalization techniques to select suitable training data. The goal of feature selection algorithms is to find the best set of features that will allow one to build a well performed classifier. There can be supervised or unsupervised techniques for classifying the data, depending on whether the data are labeled. We can classify such techniques in categories. Filter techniques which enable one to pick up intrinsic properties of each feature via using univariate statistics instead of cross-validation. Such methods include information gain which is the information gain of each variable. Chi-square test which selects the desired number of features with the best chi-square scores. The Fisher score (gradient or the derivative of the log likelihood function) can be used to rank the variables. The use

of correlation for feature selection can be used to select variables that are highly correlated with the target but are uncorrelated among themselves. The variance threshold removes all features whose variance does not meet some threshold such as removal of all zero variance features or includes the removal of features that hold the same variance. In addition, one can use the mean absolute difference from the mean value or the calculation of dispersion ratios.

Other forms of selecting features includes wrapper methods. Wrapper methods search all possible subsets of features by learning a classifier and evaluating on each iteration its performance. These methods can include recursive feature eliminations which selects in each training iteration according to the performance of a classifier (the coefficients of a linear model) less features. Lasso methods which consist of adding a penalty to the different parameters of the ML model thus to reduce the freedom of the model in order to avoid over-fitting. Random Forest Importance or Markov Blankets can rank features by placing them in decision trees or nodes and measure how well can they improve the purity of the nodes.

2.3.1 Class imbalance handling

A common problem during the development of ML models is the existence of class imbalance in the population which has a tremendous impact in the training process. Class imbalance occurs in the case where the population of the study group is significantly lower than the population of the control group. In this case, there are several methods that have been proposed to deal with this issue and are presented below, including: (i) random undersampling of the majority class, where the majority class is under-sampled into a given ratio, and (ii) random resampling of the minority class, where the samples in the minority class are replicated to match the population of the samples in the majority class. Let' s consider a binary classification problem where the number of patients in the control group is significantly larger than the number of patients in the study group. In that case, the term "majority class" refers to the class of the control group (e.g. to the patients that have a "0" value) whereas the term minority class refers to the class of study group (e.g. to the patients that have a "1" value). Of course, the opposite can also occur.

2.3.2 Random under-sampling of the majority class

One strategy for class imbalance handling is to reduce the number of samples in the majority class so that a ratio 1: x is maintained between the minority class and the majority class. The ratio 1: x is known as the under-sampling ratio, where "x" denotes the number of times that the majority class will be under-sampled. For example, in the case where the population group consists of 1000 patients and the number of patients in the control group (majority class) is 900 and the number of patients in the study group (minority class) is 100, an under-sampling ratio of 1: 2 would result to a new dataset with 300 patients, where the number of patients in the study group will be the same and the number of patients in the majority group will be 200. During the under-sampling process, the number of patients in the control group can be selected randomly with or without replacement to match the given ratio. To avoid any biases during the application of a ML algorithm on only one randomly under-sampled population, the ML algorithm can be re-applied on different under-sampled subsets and the performance evaluation results can be averaged. A more straightforward method for under-sampling is to compute the Tomek links (Zeng et al. 2016), i.e. pairs of close instances in the minority and majority class and remove the samples of the majority class on each link to increase the distance between the samples among the two classes. Another attempt is to use a combination of undersampling with ensemble learning (Polikar 2012) which can achieve maximum performance. Such schemes can be visualized above (Figure 1A).

2.3.4 Random over-sampling of the minority class

Another strategy for class imbalance handling is to increase the number of samples in the minority class. In the over-sampling process, the records of the minority class are duplicated in order to match with the number of samples in the majority class. In the previous example, where the

population group consists of 1000 patients and the number of patients in the control group (majority class) is 900 and the number of patients in the study group (minority class) is 100, an over-sampling ratio of 1: 2 would result to a new dataset with 600 patients, where the number of patients in the study group will be duplicated (replicated) two times (yielding 300 patients) and the number of patients in the majority group will be 300 out of 900 to match the population. In the case where the over-sampled dataset included all 900 patients in the majority class the minority class should be replicated 8 times. This, however, introduces critical biases during the development of a classifier since the samples in the minority class are duplicated and the training process is overfitted. A more straightforward and at the same time controversial method for over-sampling is SMOTE (Synthetic Minority Oversampling TEchnique) (Chawla et al. 2002) which generates synthetic patient records by selecting a random sample from the minority class and then uses the k-nearest neighbors' method to generate synthetic data points in order to place them around that sample.

The ADASYN method (He et al. 2008) uses a weighted distribution based on the difficulty in learning, i.e. more synthetic data are generated for a harder learning process. This results in reducing the bias from incorrect imbalancing via shifting appropriately the classification boundaries. Other methods such as Cluster-Centroids under-samples and resizes via clustering and finding centroids where these centroids are used to replace the original samples (Figure 1B).

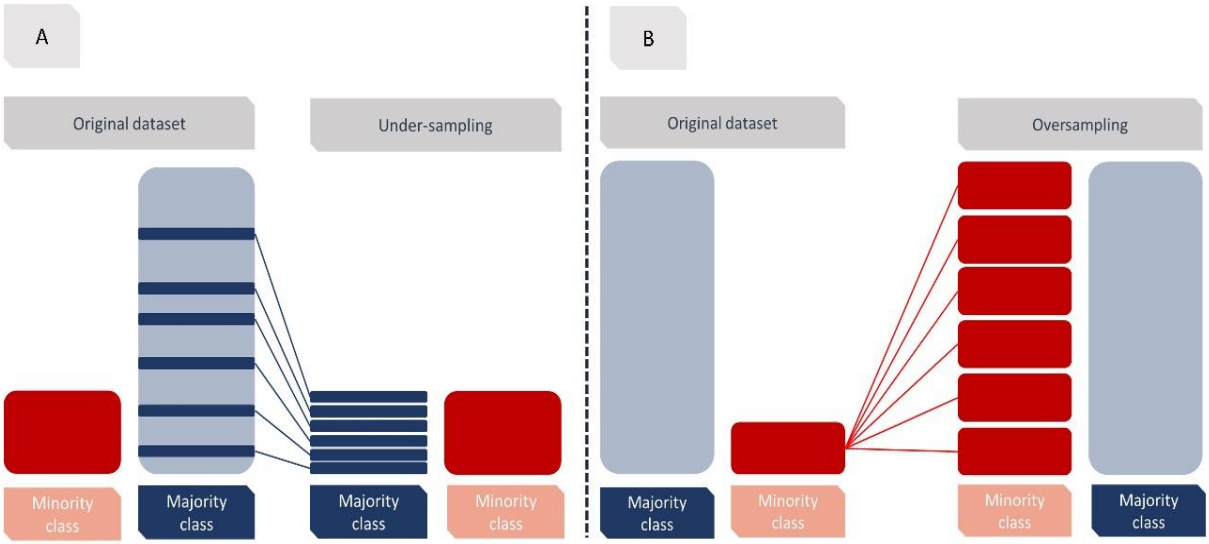


Figure 1. Differences between undersampling (A) and oversampling (B).

2.3.5 Heuristic rules for data selection

Other known methods for handling imbalanced datasets include Near-Miss algorithms (Taylor et al. 2014). The Near-Miss algorithms apply heuristic rules in order to select samples. There are 3 main categories that Near-Miss algorithms use. NearMiss-1 uses data from the majority class with an average distance k which is the nearest distance to the minority class samples. Similarly, NearMiss-2 includes the samples from the majority class where the average distance to the farthest samples of the minority class is the smallest. NearMiss-3 where at first this algorithm uses the nearest neighbors of the minority samples, then the majority samples are those where the average distance from the m neighbors is the largest.

Condensed Nearest Neighbour is another technique which uses an 1-NN to iteratively decide if a sample should be kept in a dataset or not. This method however is sensitive to noise due to the fact that noisy samples might be preserved in a class. One Sided Selection also uses the 1-NN but also uses Tomek Links to remove the samples which might be considered as noisy. In addition, the

neighborhood cleaning rule uses an edited nearest neighbor to remove some samples. Additionally, they use 3 nearest neighbors to remove samples which do not agree with this rule.

3. Association rules

3.1 Association rule mining

Association rule mining is widely adopted towards the discovery of hidden associations within clinical and genetic data which are described in the form of mathematical rules or sets of frequent items. The outcome of the association analysis can be treated as a matrix, where each row corresponds to a transaction and each column to an item. An association rule can be described using the form , where and are disjoint item sets. The item sets and B are called antecedent and consequent, respectively. The significance of an association rule can be quantified using two metrics, namely the support and the confidence. The support metric determines how often a rule is applicable whereas the confidence metric determines how frequently items in appear in transactions that contain . Support is an important metric because a rule that has a very low support may occur simply by chance. On the other hand, confidence measures the reliability of the inference made by a rule. The higher the confidence the more likely it is for to be present in transactions that contain . In addition, confidence provides an estimation related to the conditional probability of given A. The association rule mining problem can be generalized as the problem of finding all the rules having support ≥ sup and confidence ≥ conf, where sup and conf are the thresholds of support and confidence, respectively. Using large support and confidence values we ensure that only significant rules are extracted to avoid any random occurrence with increased conditional dependence among the items.

$$\text{Support}=\frac{Numberoftransactionwithboth A\cap B}{Totalnumberoftransanctions}=P(A\cap B) \tag{6}$$

$$\text{Confidence}=\frac{Numberoftransactionswithboth A\cap B}{P(A)}=\frac{P(A\cap B)}{P(A)} \tag{7}$$

$$\text{Lift}=\frac{P(A\cap B)}{P(A)xP(B)} \tag{8}$$

A common pipeline for association rule mining involves the application of two steps: (i) *the detection of* item sets that satisfy the *sup*-threshold which are referred to as frequent item sets, and (ii) *the extraction of* the high-confidence rules from the frequent item sets. Assuming a set of items in a data set, say $I=\{i_1,i_2,\ldots,i_n\}$, and a set of transactions, say $T=\{t_1,t_2,\ldots,t_m\}$, a transaction, $t_j, j=1,2,\ldots,m$, contains the item set I. If an item set contains items, it is called a k-itemset. An example of an association rule in the form of a 2-itemset is the following:

$$\{Cryoglobulinemicvasculitis_{presence}\}\rightarrow Cryoglobulinemia_{presence}, \tag{9}$$

which implies that a patient with cryoglobulinemic vasculitis has also cryoglobulinemia. A widely used algorithm for association rule mining is the Apriori algorithm (Hegland et al. 2007) which uses support-based pruning to crease the exponential growth of candidate item sets thus, generating reliable frequent item sets and rules based on a level-wise approach, where each level corresponds to the number of items that belong to the rule consequent. These rules enable the detection of hidden patterns underlying disease onset and progression. In contradiction, the FP-Growth algorithm (Borgelt et al. 2005), which is an alternative of the A-priori algorithm, does not require

candidate generation but uses a frequent pattern tree (FP-tree) without the need to generate candidate sets making FP-Growth particularly useful towards the detection of associations in big data.

4. Acceding the performance of the classifier

4.1 Train/test splitting

Perhaps one of the simplest and at the same time biased methods for validating the performance of a classifier. In the train/test splitting process, a ratio x/y is pre-defined to split the dataset into one training and one testing subset. For example, in a 0.8 ratio, 80% of the samples will be assigned in the training subset and 20% in the testing subset. Then, the classified is trained on the training subset and evaluated on the testing subset. It is obvious that this method introduces biases since the splitting process is randomly executed and only for one time.

4.2 Clustering metrics

As it has already been mentioned in unsupervised learning, any prior knowledge regarding the class (i.e., the true set of labels) is unknown. To evaluate the appropriateness of a pre-defined number of clusters (or partitions), assume ^{B}, which yield the optimal set of clusters (or partitions) for a particular unsupervised learning problem, three popular indices that quantify the overall clustering density have been proposed, namely: (i) the silhouette (Rousseeuw 1987), (ii) the Davies-Bouldin (Davies and Bouldin 1979), and (iii) the Calinski-Harabasz (Wang et al. 2019).

For a given set of clusters, say  , the silhouette index (SI), ^{B} , is defined as in (10):

$$silh(K) = \frac{a - b}{max(a, b)}, \tag{10}$$

where, a is the between-the-cluster distance, i.e., the average distance between an observation that belongs to cluster ^{B} with all its neighboring observations in K_j, and b is the within-the-cluster distance, i.e., the average distance between the same observation, x_i, with its neighboring observations from the next nearest cluster. The silhouette value is evaluated on different c values and the clustering number with the highest silhouette score is the one that yields optimal clusters with small between-the-cluster distance and large within-the-cluster distance. A silhouette value 1 denotes a well-separated cluster, whereas a value -1 denotes otherwise.
Another widely used clustering evaluation index is the Davies-Bouldin (DB) index, which quantifies the clustering similarity of K_j by taking into consideration the average distance between the centroids of K_j with its most similar one from the set of clusters K, as in:

$$DB(K_{i,j}) = \frac{1}{c}\sum_{i=1}^{c} max\,(R_{lj}), \tag{11}$$

where ^{A} is the number of clusters, and R_{ij} is the similarity between cluster K_i and cluster ^{B} , ^{A}:

$$R_{lj} = \frac{s_i + s_j}{s_{ij}}. \quad (12)$$

In (12), ^{B} is the average distance between the samples in ^{A} from its centroid, and  is the distance between the centroids in $^{P(A\cap B)}$ and  . The cluster number that achieves the lowest DB score (ideally close to 0) is the one that yields well-separated clusters since a large distance between the clustering centroids, $\frac{P(A\cap B)}{P(A)}$ denotes better separation.

Finally, the Calinski-Harabasz (CH) index measures the between the cluster variance (BCV) and the within the cluster variance (WCV) to quantify the clustering density of each cluster in $\frac{P(A\cap B)}{P(A)P(B)}$, as in:

$$I = \{i_1, i_2, \ldots, i_n\} \quad (13)$$

where the BCV is defined as in:

$$T = \{t_1, t_2, \ldots, t_m\} \quad (14)$$

and the WCV is defined as in:

$$t_j, j = 1,2, \ldots, m \quad (15)$$

In (15), ^{I} denotes the number of samples in  ,  is the clustering centroid of  ,  is the overall mean of the samples. In (40) the term  is the Euclidean distance between the input samples ^{k} and the centroid  of cluster K_i. In practice, high CH scores denote well-separated clusters.

5. Applications of AI with Python.

This section provides some basic examples of how to build Deep Neural architectures using libraries in python from Keras (Gulli and Sujit 2017), Tensorflow (Abadi et al. 2016) and Pytorch (Paszke et al. 2019). In addition, examples for evaluating the performance of classifiers are demonstrated. The example code for the CNN is based on DeepRipe (Ghanbari and Ohler 2020) architecture for deciphering the binding properties of RBPs.

5.1 Code examples in Python for Building a CNN model:

Import of libraries:

```
import sys
import os
import numpy as np
import h5py
import scipy.io
import tensorflow as tf
import pdb

from keras.utils import plot_model
from keras.models import Model, load_model
from keras.layers import Input
from keras.layers import Dense, Flatten, Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.pooling import MaxPooling1D
from keras.layers.merge import concatenate
from keras.optimizers import SGD, RMSprop, Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from keras.layers.normalization import BatchNormalization
from keras.layers.recurrent import GRU
from keras.layers.wrappers import Bidirectional
from keras.layers import LSTM
import keras.backend as K

import matplotlib as mpl

import matplotlib.pyplot as plt
from sklearn.metrics import f1_score

import math

from keras.layers import Input,Conv2D,MaxPooling2D,UpSampling2D,UpSampling1D
from keras.models import Model
from keras.optimizers import RMSprop
```

Built convolutional neural networks:

```
CONVOLUTIONAL NEURAL NETWORK
```

```
    input_1 = Input(shape=(nxm),name="input_1")


    conv1 = Conv1D(filters=n,kernel_size=m, padding='x',activation="x",name="input1_conv1")(input_1)
    pool1 = MaxPooling1D(pool_size=n, strides=m,name="pool1")(conv1)
    drop1 = Dropout(n,name="drop1")(pool1)

############ Built fully connected network ###########################################################3

    hidden1 = Dense(250, activation='relu',name="hidden1")(drop)
    output = Dense(num_task, activation='sigmoid',name="output")(hidden1)
    model = Model(inputs=[input_1,], outputs=output)
    return model
```

5.2 Deep autoencoder model

```
#input = 28 x 28 x 1 (wide and thin)
conv1 = Conv1D(filters=128, kernel_size=7, activation='relu', padding='same')(input) #28 x 28 x 32
pool1 = MaxPooling1D(pool_size=(4),strides=2)(conv1) #14 x 14 x 32
233conv2 = Conv1D(filters=64,kernel_size=7, activation='relu', padding='same')(pool1) #14 x 14 x 64
pool2 = MaxPooling1D(pool_size=(4),strides=2)(conv2) #7 x 7 x 64
conv3 = Conv1D(filters=32,kernel_size=7, activation='relu', padding='same')(pool2) #7 x 7 x 128
```

```
conv4 = Conv1D(filters=32,kernel_size=7, activation='relu', padding='same')(conv3) #7 x 7 x 128
up1 = UpSampling1D((2))(conv4) # 14 x 14 x 128
conv5 = Conv1D(filters=64,kernel_size=7, activation='relu', padding='same')(up1) # 14 x 14 x 64
up2 = UpSampling1D((2))(conv5) # 28 x 28 x 64
decoded = Conv1D(filters=128,kernel_size=7, activation='relu', padding='same')(up2) # 28 x 28 x 1
```

5.3 Basic design of Encoder / Decoders

```
#fully-connected neural encoder-decoder layer using as input an image
###examples obtained from https://blog.keras.io/building-autoencoders-in-keras.html
import keras
from keras import layers
#####################################Simple encoder -decoder  ##########
# size of encoded representations
encod_dim = 64
# input image
input_img = keras.Input(shape=(1568,))
# "encoded"
encoded = layers.Dense(encod_dim, activation='relu')(input_img)
# "decoded"
decoded = layers.Dense(1568, activation='sigmoid')(encoded)

# Create a model which Maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
#############################################################################################################
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

#load mnist data set # picture digits from Mnist
from keras.datasets import mnist
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data()
#########################################################normalise mnist images between 0 and 1
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))


# Encode and decode digit images from mnist
encoded = encoder.predict(x_test)
decoded = decoder.predict(encoded)
```

```
########################################plot original and reconstructed images
import matplotlib.pyplot as plt

n = 5  #first 5 mnist digits
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

5.4 Basic design of Logistic regression & random forest

```
LOGISTIC REGRESSION
```

```
from sklearn.calibration import calibration_curve
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticD
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKF
from sklearn.metrics import roc_curve, auc, roc_auc_score, make_scorer
from sklearn.svm import LinearSVC, NuSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from skopt import BayesSearchCV
from skopt.plots import plot_evaluations, plot_objective
from skopt.space import Real, Categorical, Integer

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
model.fit(x, y)
model.predict_proba(x)
model.score(x, y)
confusion_matrix(y, model.predict(x))
```

5.5 Example of Random Forests

```
RANDOM FOREST
```

```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor

# Ensmble a  model with x decision trees

rf = RandomForestRegressor(n_estimators = x, random_state = m)

# Train the model on training data

rf.fit(train_features, train_labels);
# Testing
predictions = rf.predict(test_features)

# Calculate the absolute errors

errors = abs(predictions - test_labels)

# Print the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Testing
predictions = rf.predict(test_features)

# Calculate the absolute errors

errors = abs(predictions - test_labels)

# Print the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

6. Testing functions in python for accessing the performance of classifiers.

The performance of classification can be assessed via several methods such as with the aid of confusion matrices or via determining and demonstrating the scores on what motifs or values the classifier has learned on while using the integrated gradient method (Sundararajan et al. 2017).

6.1 Examples in R for machine learning and testing the performance of a classifier while learning from gene expression data.

The following examples demonstrate a learning procedure based on gene expression data while using Random forest, glm and glmnet. A glmnet model (Friedman et al., 2010) fits generalized linear models via penalizing the maximum likelihood.

```
In [45]: library(randomForest)
```

##Load data

In [121]:
```
rawCountTable <- read.table("~/Desktop/lessons/senesence_project/genes2/3dOn_OFFnorm.csv", header=TRUE, sep=",", ro
sampleInfo <- read.table("~/Desktop/lessons/senesence_project/genes2/3dOn_OFFmeta.txt", header=TRUE, sep="\t", row.
```

##Create training and test data

In [138]:
```
set.seed(3031) # set the random number seed for reproducibility

# # seperate test and training setse data set
intrain <- createDataPartition(y = tgexp[,1], p= 0.5)[[1]]

training <- tgexp[intrain,]
testing <- tgexp[-intrain,]
```

In [139]:
```
head(training)
```

A data.frame: 4 × 3081

	condition	ENSG00000188290	ENSG00000188157	ENSG00000207730	ENSG00000207607	ENSG00000198976	ENSG00000078808	ENS
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
d3ON_1Aligned.out.bam	3dON	2.000000	2.833333	61.08333	2279.833	1918.833	2.000000	
d3ON_2Aligned.out.bam	3dON	3.833333	2.000000	49.66667	2131.083	1884.500	1.000000	
OFF_1Aligned.out.bam	OFF	1.000000	2.666667	28.91667	2769.500	1918.833	4.500000	
OFF_3Aligned.out.bam	OFF	4.333333	6.500000	47.16667	1282.417	1282.417	5.333333	

In [140]:
```
head(testing)
```

A data.frame: 2 × 3081

	condition	ENSG00000188290	ENSG00000188157	ENSG00000207730	ENSG00000207607	ENSG00000198976	ENSG00000078808	ENS
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
d3ON_3Aligned.out.bam	3dON	2.833333	2.833333	37.91667	1436.333	1273.667	2.833333	
OFF_2Aligned.out.bam	OFF	1.916667	4.500000	39.33333	1884.500	1655.667	4.500000	

In [136]:
```
tgexp=merge(sampleInfo,tgexp,by="row.names")

rownames(tgexp)=tgexp[,1]
tgexp=tgexp[,-1]
```

In [137]:
```
(tgexp)
```

A data.frame: 6 × 3081

	condition	ENSG00000188290	ENSG00000188157	ENSG00000207730	ENSG00000207607	ENSG00000198976	ENSG00000078808	ENS
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
d3ON_1Aligned.out.bam	3dON	2.000000	2.833333	61.08333	2279.833	1918.833	2.000000	
d3ON_2Aligned.out.bam	3dON	3.833333	2.000000	49.66667	2131.083	1884.500	1.000000	

#Determine labels

In [28]:
```
gexpnoNA=missing_tgexp[ , colSums(is.na(missing_tgexp)) == 0]
```

In [29]:
```
tgexp=merge(patient,tgexp,by="row.names")

# push sample ids back to the row names
rownames(tgexp)=tgexp[,1]
tgexp=tgexp[,-1]
```

In [31]:
```
install.packages("caret")
library(caret)
```

Random Forest

In [46]:
```
set.seed(17)
options(expressions = 5e5)
trctrl <- trainControl(method = "none")

# train random forest model
rfFit <- train(subtype~.,
               data = training,
               method = "ranger",
               trControl=trctrl,
               importance="permutation", # calculate importance
               tuneGrid = data.frame(mtry=100,
                                     min.node.size = 1,
                                     splitrule="gini")
               )

rfFit$finalModel$prediction.error
```

GLM model

In [145]:
```
lrFit = train(condition~.,
              data = training, trControl=trainControl("none"),
              method="glm", family="binomial")
```

In [147]:
```
plot(varImp(lrFit),top=10)
```

GLMNET model

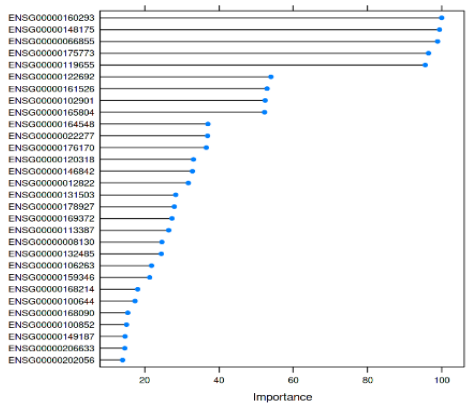
In [150]:
```
trctrl <- trainControl(method = "cv",number=10)

# we will now train elastic net model
# it will try
enetFit <- train(condition~., data = training,
                 method = "glmnet",
                 #trControl=trctrl,
                 # alpha and lambda paramters to try
                 tuneGrid = data.frame(alpha=0.5,
                                       lambda=seq(0.1,0.7,0.05)))

# best alpha and lambda values by cross-validation accuracy
enetFit$bestTune
```

Output feature importance as genes that contribute the most to the classification procedure

In [153]:
```
plot(varImp(enetFit),top=30)
```



7. Additional clustering algorithms

7.1 Spectral clustering

Spectral clustering (Von Luxburg 2007) is an alternative approach for data clustering which adopts mathematical concepts from spectral graph theory to cluster the data using a different embedding than in the Euclidean space. This embedding is based on a similarity (possibly weighted) graph whose vertices are the data points providing a set of affinity relations between the data, which may be more general than the ones that can be expressed by standard distances as for

example the ones mentioned above; in this viewpoint data points whose distance (in any of the above sense) is far, may still be related by some affinity condition. These affinity relations are used to create a graph of connectivities between the data, and this graph is the essential object in spectral clustering. There are different ways of constructing this graph, such as the ε-neighborhood graph where all points whose pairwise distances as smaller than ε are connected, the k- nearest neighbor graphs in which vertex (data point) i is connected with vertex (data point) j if j is among the k nearest neighbors of i. These graphs can be represented by an affinity matrix with non-zero entries between the connected vertices (data points). An interesting variant is the case of fully connected graphs where such affinity relations can be weighted in terms of a measure of affinity, with these measures for every pair of data point collected in an affinity matrix of the real data, i.e., an $N \times N$ square matrix, where a cell (i, j) represents the degree of similarity between a data point i and a data point j and n is the number of features. Typical similarity measures for computing the affinity matrix W include the Gaussian similarity measure $s_{ij} = exp\left(\frac{-1}{\varepsilon}||x_i - x_j||^2\right)$, where $|||$ can be the Euclidean distance, among others, and $\epsilon > 0$, is a suitable parameter. The similarity matrix is defined as [illegible], since the data points with small distance refer to highly similar ones. The resulting graph (weighted or not) carries important information on the connectivity and affinity of the data points, for instance if the graph is not fully connected but consists of say 2 disconnected subgraphs this is a clear indication of the existence of two distinctive clusters of qualitatively similar data. A fruitful way of revealing information concerning the connectivity properties of the data is through the matrix representation of the data affinity graph, and in particular, through a related matrix called the graph Laplacian and its spectral representation. There are various versions of the graph Laplacian, with different interpretations, the unnormalized graph Laplacian $L = D - W$ (where D is the degree matrix of the graph) or normalized versions such as:

$L_{sym} = \left(D^{\frac{1}{2}}\right)^{-1} L \left(D^{\frac{1}{2}}\right)^{-1}$ or $K=\{K_1, K_2, .., K_c\}$ which is related to random walks on the graph.The spectral properties, i.e., the eigenvalues and eigenvectors of these matrices reveal important information concerning the affinity graph, for example the multiplicity of the 0 eigenvector of either version of L gives the number of connected components of the graph with the eigenvectors delineating the content of the components. This allows partitioning of the Laplacian into block form, with the submatrices corresponding to the Laplacian of each of the connected components, with their corresponding spectra revealing the substructure of the components. Knowledge of the full spectral decomposition of the Laplacian (i.e., all eigenvalues $sinh(K)$ and eigenvectors) allows full reconstruction of the Laplacian matrix with major information concerning the matrix encoded in the eigenvectors corresponding to the largest eigenvalues. Spectral clustering methods use the full information of the spectral content on the Laplacian, i.e., slightly abusing terminology a representation of the data in "spectral space" rather than in standard Euclidean space, in combination with standard clustering schemes such as the k-means to reconstruct the most salient features of the affinity graph, its connected components and their microstructure. Great applications of such mathematical approach allow testing the affinity of the cells from scRNA-seq while performing spectral clustering on gene expression per cell (Park and Zhao 2018).

7.2 Hierarchical clustering

Hierarchical clustering (Sasirekha and Baby 2013) is another widely used method for data clustering which does not pre-specify the number of required clusters, but rather chooses the number of clusters in terms of a hierarchical process, in which – based upon a predetermined dissimilarity measure between disjoint groups of observations – the clusters within each level of the hierarchy are created by merging of the clusters determined on the previous or next level of the hierarchy (depending on the choice of scheme). It focuses on the construction of hierarchical data models (known as dendrograms) which can be constructed in two different ways either through a top-down (divisive – starting at the top and splitting the existing clusters in a pair of new clusters at

the next level) or a bottom-up (agglomerative – at each level merging a selected pair of clusters to a single cluster on the next level) approach. The dendrograms share a similar structure with the decision trees in supervised learning, including nodes and leaves, as well, where each node represents a cluster, and the number of leaves is equal to the number of pre-defined clusters. Thus, the depth of the dendrogram depends on the number of pre-defined clusters. Typical distance functions (used to model dissimilarity measures) that are deployed towards the partitioning or merging of clusters with similar properties in both divisive and agglomerative hierarchical clustering approaches, which respectively, include the Mahalanobis distance (McLachlan 1999), the Euclidean distance (Mohibullah et al. 2015), and the Manhattan distance (Mohibullah et al. 2015) among others. These distance functions are also referred to as linkage functions since they link similar clusters. Furthermore, the Mahalanobis distance can be used to detect outliers from a distribution, is motivated by the multidimensional normal distribution of zero mean and with covariance matrix S and in effect is a weighted Euclidean distance using the inverse covariance matrix (or precision matrix) i.e.,

$$d_M(x,y) = (x \cdot S^{-1} y)^{\frac{1}{2}} \tag{16}$$

Different distances endow the n-dimensional space that the data reside in with different geometries and this geometrical structure may play an important role on the clustering procedure.

7.3 Force directed graphs and its applications in biology

The recent advance in scRNA-seq technology allows to determine the cellular state of each cell subpopulation. ML applications have paved the way to predict cell trajectories and determine the cellular differentiation and so the evolution of any cell system. To tackle such task, force directed graph algorithms have been implemented. These algorithms consider edges as applying forces (repulsive or attractive) to nodes thus, edge-weights are applied to define a relation between any node and thus to determine cluster structures. A great example of such an approach is GraphDDP (Costa et al 2021) . The main idea is first to create an unweighted instant graph G=(V,E) having V to be a set of vertices and E the set of edges. A length is assigned to each edge based upon a desired distance between the points of the edge. Borrowing the fundamental principles of dynamic particle physics where each particle is described with v_i=x_i,y_i coordinates connected by springs of strength k_{ij}. The layout that will minimise the total energy of the system will be the optimal chosen. The total energy of the system is considered as:

$$\text{E=} \quad silh(K) = \frac{a-b}{max(a,b)}, \tag{17}$$

where d_{ij} is the desired distance between i and j points. The spring strength is defined as $k_{ij} = K/d_{ij}^2$with constant K. The solution to derive the minimum energy can be given as $\frac{\theta x}{\theta E} = \frac{\theta y}{\theta E} = 0$for $1 \leq m \leq n$. To solve this, a two dimensional Newton-Raphson method (Dence, T. 1997) is applied which includes only one particle moving from one point to the other while the others particles are stable. Optimal-Transport Analysis (Schiebinger et al., 2021) is another method which enables to learn a relationship between ancestor cells and descendant cells at specific time points. It uses stochastic couplings to determine the transitions between time points without strict structural constraints. More precisely the optimal-transport calculation assumes that the cell's future state depends only on it's current position irrelevant of the previous history. This problem can be described as : Let X_0 and X_1 be the one dimensional Gaussian random variable describing the location of a particle at time points 0 and 1. Assuming that the particles or cells can not move very far (change the cell state fast), a γ coupling estimate between X_0 and X_1 is defined with Eq. 18 as

$$\gamma \Leftarrow argmin_{\pi} E_{\pi} \|X_0 - X_1\|^2$$

(18)

In general, If the cost for a mass to be transferred from point x to y is set as c(x,y) then transporting an amount of mass from x to y according to a transport plan $\pi\ (x,y)$ will result to a total cost that is defined via Eq. 19

$$\iint c\ (x,y)\pi(x,y)d_x d_y \text{where} \quad c(x,y) = \text{a}$$

(19)

The minimisation of Eq 19 will be the optimal transport principle.

8. Application examples for dimensionality reduction techniques and clustering using real data

8.1 UMAP and clustering applications in sc-RNAseq analysis

As it has been examined the advances of scRNA-seq can provide us information of the characteristic gene expression per cell type which thus can provide us with specific biomarkers of differentiation. A plethora of methods and analysis exist taking advantage of several dimensionality reduction techniques such as PCA, t-SNE and UMAP. Spatial transcriptomics in organoids or tissues in combination with scRNA-seq (Saviano et al. 2020) can be efficiently used to provide a map resolution of the transcriptome. ML techniques can be applied to learn the state of the cells in developmental biology (Karaiskos et al. 2017) or senescence from gene expression data for the identified cell clusters. Methods such as Viterbi or HMM can be applied to predict the cell state fate (McGarvey et al. 2020). In Figure 2 a UMAP clustering of data from a 10x library (https://www.10xgenomics.com/resources/library/) for sc-RNA-seq analysis is provided as done with Seurat sc-RNAseq analysis suite (Hao et al. 2020).

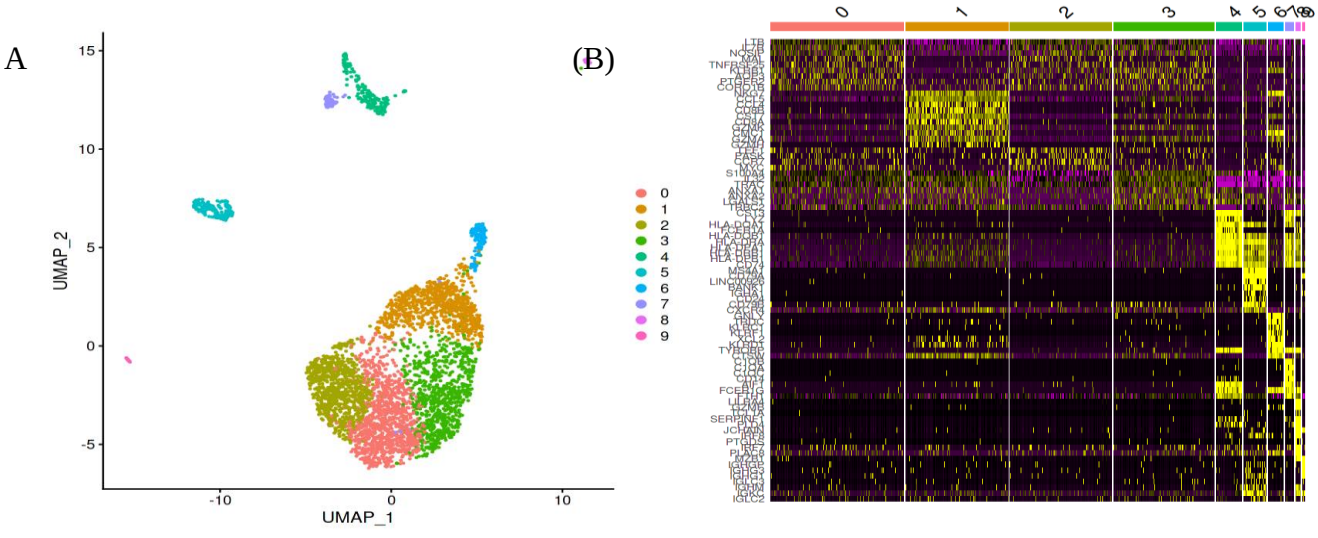


Figure 2: Classification of cells from sc-RNAseq on cerebrospinal fluid leukocytes in multiple sclerosis (Schafflick et al. 2020). In (A) we can define 9 clusters or 9 subtypes and in (B) the clustering of the genes that compose the sub clusters.

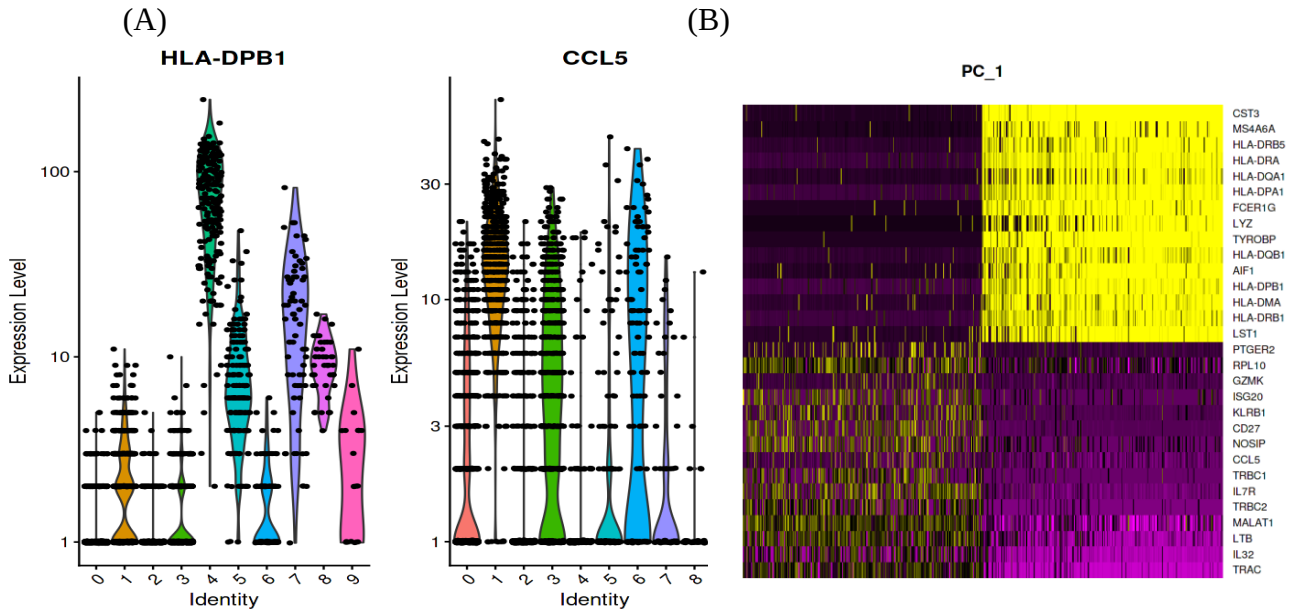


Figure 3: Expression of identified marker genes in (A) and in (B) the clustering per principal components.

Examples for clustering gene expression data using Python:

The data processing is based on a multiple sclerosis study from GSE60424 (Linsley et al. 2014).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap

import sklearn.cluster as cluster
from sklearn.metrics import adjusted_rand_score, adjusted_mutual_info_score

#Creating dataframe for data.
data = pd.read_csv('~/MS/transposematrixImmune.csv', delimiter='\,',index_col=0)
nRow, nCol = data.shape
print(f'There are {nRow} rows and {nCol} columns in data.')


labels = pd.read_csv('~/MS/immuneclass.txt', delimiter='\t')
nRow, nCol = labels.shape
print(f'There are {nRow} rows and {nCol} columns in labels dataframe.')
```

```
<ipython-input-24-e74bff4e092f>:14: ParserWarning: Falling back to the 'python' engine because the 'c' engine does
not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoi
d this warning by specifying engine='python'.
  data = pd.read_csv('~/Desktop/intelligencia_MS/MS/transposematrixImmune.csv', delimiter='\,',index_col=0)
```

```
There are 134 rows and 19420 columns in data.
There are 88 rows and 2 columns in labels dataframe.
```

Loading of data:

Construction of a PCA plot:


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap

import sklearn.cluster as cluster
from sklearn.metrics import adjusted_rand_score, adjusted_mutual_info_score

#Creating dataframe for data.
data = pd.read_csv('~/MS/transposematrixImmune.csv', delimiter='\,',index_col=0)
nRow, nCol = data.shape
print(f'There are {nRow} rows and {nCol} columns in data.')


labels = pd.read_csv('~/MS/immuneclass.txt', delimiter='\t')
nRow, nCol = labels.shape
print(f'There are {nRow} rows and {nCol} columns in labels dataframe.')
```

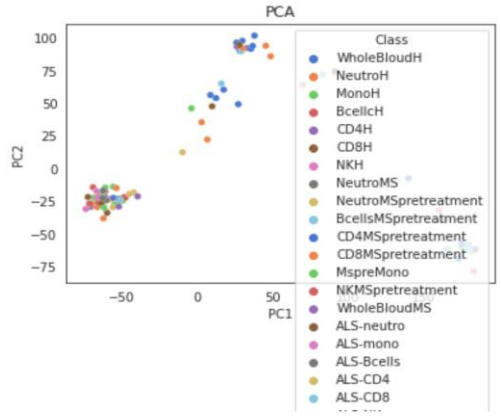
```
<ipython-input-24-e74bff4e092f>:14: ParserWarning: Falling back to the 'python' engine because the 'c' engine does
not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoi
d this warning by specifying engine='python'.
  data = pd.read_csv('~/Desktop/intelligencia_MS/MS/transposematrixImmune.csv', delimiter='\,',index_col=0)
```

```
There are 134 rows and 19420 columns in data.
There are 88 rows and 2 columns in labels dataframe.
```

```
X = data.values
X_std = StandardScaler().fit_transform(X)
print("Principal Component Analysis (PCA)")
pca = PCA(n_components = 2).fit_transform(X_std)
pca_df = pd.DataFrame(data=pca, columns=['PC1','PC2']).join(labels)
palette = sns.color_palette("muted", n_colors=32)
sns.set_style("white")
sns.scatterplot(x='PC1',y='PC2',hue='Class',data=pca_df, palette=palette, linewidth=0.2, s=30, alpha=1).set_title('
```

```
Principal Component Analysis (PCA)

Text(0.5, 1.0, 'PCA')
```

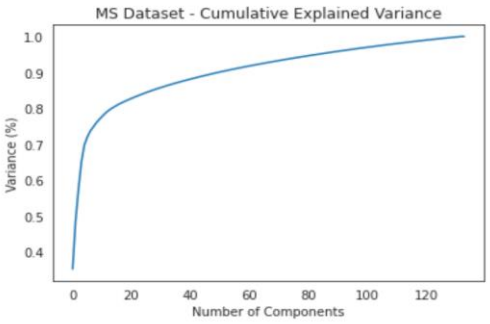


Examining the variance:

```
#Fitting PCA on Data
print("Explained Variance of PCA components")
pca_std = PCA().fit(X_std)
percent_variance=pca_std.explained_variance_ratio_*100

#Plotting Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca_std.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('MS Dataset - Cumulative Explained Variance')
plt.show()
```

```
Explained Variance of PCA components
```

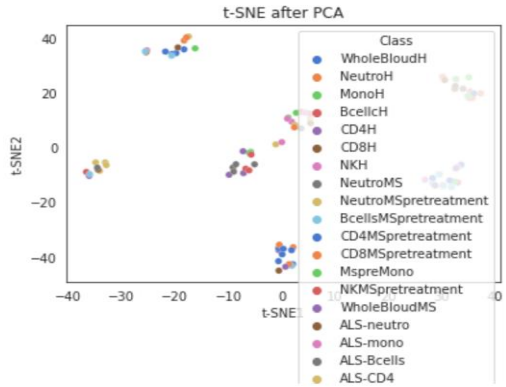


Construction of tSNE plots:

```
#Visualize data using t-SNE after PCA.
print("t-Distributed Stochastic Neighbor Embedding (tSNE) on PCA")
X_reduced = PCA(n_components =5 ).fit_transform(X_std)
model = TSNE(learning_rate = 50, n_components = 2, random_state = 123, perplexity = 7)
tsne_pca = model.fit_transform(X_reduced)
tsne_pca_df = pd.DataFrame(data=tsne_pca, columns=['t-SNE1','t-SNE2']).join(labels)
palette = sns.color_palette("muted", n_colors=32)
sns.set_style("white")
sns.scatterplot(x='t-SNE1',y='t-SNE2',hue='Class',data=tsne_pca_df, palette=palette, linewidth=0.2, s=30, alpha=1).
```

```
t-Distributed Stochastic Neighbor Embedding (tSNE) on PCA
```

```
Text(0.5, 1.0, 't-SNE after PCA')
```



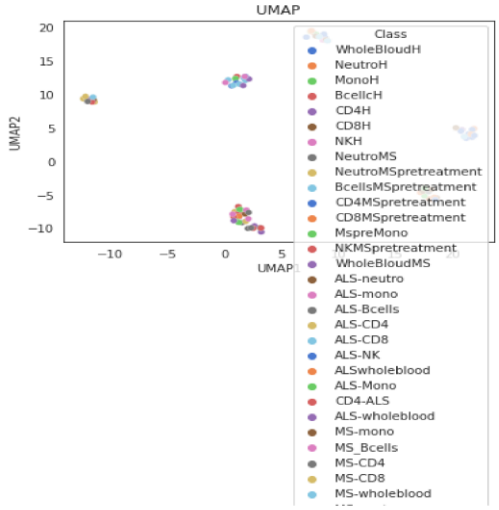
UMAP example:

```
#Visualize data using UMAP.
#import umap_learn
import umap.umap_ as umap
print("Uniform Manifold Approximation and Projection (UMAP)")
model = umap.UMAP(n_neighbors = 7, min_dist = 0.4, n_components = 2)
umap = model.fit_transform(X_std)
umap_df = pd.DataFrame(data=umap, columns=['UMAP1','UMAP2']).join(labels)
palette = sns.color_palette("muted", n_colors=32)
sns.set_style("white")
sns.scatterplot(x='UMAP1',y='UMAP2',hue='Class',data=umap_df, linewidth=0.2, s=30, alpha=1, palette=palette).set_ti
```

```
Uniform Manifold Approximation and Projection (UMAP)
```

```
/home/orsalia/miniconda3/lib/python3.8/site-packages/sklearn/manifold/_spectral_embedding.py:236: UserWarning: Gra
ph is not fully connected, spectral embedding may not work as expected.
  warnings.warn("Graph is not fully connected, spectral embedding"
```

```
Text(0.5, 1.0, 'UMAP')
```



References:

Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., Kudlur M., Levenberg J., Monga R., Moore S., Murray D. G., Steiner B., Tucker P., Vasudevan V., Warden P., Wicke M., Yu Y., Zheng X. (2016). Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 265-283.

Altman N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician 46(3): 175-185.

Aslam M. (2020). Introducing Grubbs's test for detecting outliers under neutrosophic statistics–An application to medical data. Journal of King Saud University-Science 32 (6): 2696-2700.

Azhagusundari B. and Thanamani A. S. (2013). Feature selection based on information gain. International Journal of Innovative Technology and Exploring Engineering (IJITEE) 2 (2): 18-21.

Borgelt C. (2005). An Implementation of the FP-growth Algorithm. In Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, 1-5.

Bouckaert R R. (1993). Probabilistic network construction using the minimum description length principle. In European conference on symbolic and quantitative approaches to reasoning and uncertainty, 41-48. Springer, Berlin, Heidelberg.

Chawla N.V., Bowyer K.W., Hall L.O. Kegelmeyer W.P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16: 321-357.

Davies D L. and Bouldin D. W. (1979). A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence 2: 224-227.

Friedman J H. (2002). Stochastic gradient boosting. Computational statistics & data analysis 38 (4): 367-378.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. "Regularization paths for generalized linear models via coordinate descent." *Journal of statistical software* 33.1 (2010):

Ganguly S, Chatterjee A., Bhoumik D., Majumdar R. (2019). An Empirical Study of Incremental Learning in Neural Network with Noisy Training Set. In International Conference on Computers and Devices for Communication, 72-77. Springer, Singapore.

Ghanbari M. and Ohler U. (2020). Deep neural networks for interpreting RNA-binding protein target preferences. Genome research 30 (2): 214-226.

Gulli A. and Pal. S. (2017). Deep learning with Keras. Packt Publishing Ltd.

Hao Y., Hao S., Andersen-Nissen E., Mauck W. M., Zheng S., Butler A., Lee M. J., Wilk A.J., Darby C., Zagar M., Hoffman P., Stoeckius M., Papalexi E., Mimitou E.P., Jain J., Srivastava A., Stuart T., Fleming L.B., Yeung B., Rogers A.J., McElrath J.M., Blish C.A., Gottardo R., Smibert P., Satija R. (2020). Integrated analysis of multimodal single-cell data. bioRxiv.

He H., Bai Y., Garcia E.A., Li S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence), 1322-1328, IEEE.

Hegland M. (2007). The apriori algorithm–a tutorial. Mathematics and computation in imaging science and information processing: 209-262.

Karaiskos N., Wahle P., Alles J., Boltengagen A., Ayoub S., Kipar C., Kocks C., Rajewsky N., Zinzen R. P. (2017). The Drosophila embryo at single-cell transcriptome resolution. Science 358 (6360): 194-199.

Linsley P. S., Speake C., Whalen E., Chaussabel D. (2014). Copy number loss of the interferon gene cluster in melanomas is linked to reduced T cell infiltrate and poor patient prognosis. *PloS one* 9 (10): e109760.

Liu F. T., Ming Ting K., Zhou Z.-H. (2008). Isolation forest. In 2008 eighth ieee international conference on data mining, 413-422. IEEE.

McGarvey A. C., Kopp W., Vučićević D., Kempfer R., Mattonet K., Hirsekorn A., Bilić I., Trinks A., Merks A. M., Panáková D., Pombo A., Akalin A., Junker J. P., Stainier D.Y.R., Garfield D., Ohler U., Lacadie S. A. (2020). Single-cell-resolved dynamics of chromatin architecture delineate cell and regulatory states in wildtype and cloche/npas4l mutant zebrafish embryos. bioRxiv.

Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Köpf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S. (2019). Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703.

Polikar R. (2012). Ensemble learning. In Ensemble machine learning, 1-34. Springer, Boston, MA.

Rousseeuw P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20: 53-65.

Saunders D. J. Sigrist C., Chaney K., Kozma R., Siegelmann H. T. (2019). Minibatch processing in spiking neural networks. arXiv preprint arXiv:1909.02549.

Saviano A., Henderson N. C., Baumert T. F. (2020). Single-cell genomics and spatial transcriptomics: discovery of novel cell states and cellular interactions in liver physiology and disease biology. Journal of hepatology.

Schafflick D., Xu C. A., Hartlehnert M., Cole M., Schulte-Mecklenbeck A., Lautwein T., Wolbert J., Heming M., Meuth S. G., Kuhlmann T., Gross C. C., Wiendl H., Yosef N., Meyer zu Horste G. (2020). Integrated single cell analysis of blood and cerebrospinal fluid leukocytes in multiple sclerosis. *Nature communications* 11 (1): 1-14.

Sundararajan M., Taly A., Yan Q. (2017). Axiomatic attribution for deep networks." In International Conference on Machine Learning, 3319-3328. PMLR.

Taylor J. A., Lacovara A. V., Smith G. S., Pandian R., Lehto M. (2014). Near-miss narratives from the fire service: a Bayesian analysis. Accident analysis & prevention 62: 119-129.

Wang X. and Xu Y. (2019). An improved index for clustering validation based on silhouette index and Calinski-Harabasz index. In IOP Conference Series: Materials Science and Engineering, 569 (5): 052024. IOP Publishing.

Zeng M., Zou B., Wei F., Liu X., Wang L. (2016). Effective prediction of three common diseases by combining SMOTE with Tomek links technique for imbalanced medical data. In 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), 225-228. IEEE.