Feature Representation Selection from Gene Expression Data

1. Introduction

We live in an age where we're surrounded by data, and are often interested in collecting them [1]. Analyzing these huge portions of data is a must as they lead to important discoveries and patterns that are vital to experts and users and may lead to business decisions, directions or policies.

However, people and enterprises often don't have time to look at data [1]. Even if they did, with the great flow of data regularly coming in, it's almost impossible to observe data and simply derive conclusions about the patterns and structures, or make predictions.

That's why new-found sciences and tools such as Data Science and Machine Learning are essential to any corporation or organization dealing with big data. They are powerful tools that automatically analyze the data, classify and summarize it, and extract trends or anomalies, or discover some hidden pattern or information [1].

One important challenge rises with the rapid use of big data, which is known as the Curse of Dimensionality. In some data, there is a reasonable number of features present, compared to millions of samples or observations collected. This is often desirable in data science, as many machine learning models require millions of samples to actually work correctly and efficiently, such as Deep Neural Networks (DNN). However, in some data the opposite happens, meaning that the number of samples is much smaller than the features.

This can often occur in fields such as bioinformatics, where obtaining data for different samples is a time consuming, expensive procedure, as opposed to a single scientific experiment, where various features can be evaluated for a single sample. An applied example is the concept of measuring gene expression data on several patients or samples in bioinformatics, where one experiment is costly, but many genes (features) can be collected in that single experiment.

The curse of dimensionality is a drawback on many machine learning models like DNNs, which won't work correctly in the absence of many samples, or models like KNNs where similar features aren't detected anymore. That's why the need of reducing the dimensions and reaching a balance between the amounts of features and samples is strong. Dimensionality reduction is widely used in different fields of data science and machine learning, as it helps to avoid the curse of dimensionality, and makes the use of machine learning models possible.

In the following paper, the dimensionality reduction, it's official definition in computer science, and its application in the field of bioinformatics is presented. Overall techniques attempting to solve the problem are evaluated and compared, and the best one is chosen to tackle the corresponding problem in gene expression data in bioinformatics. Relative datasets, their source, and the preprocessing tasks on them is stated. Also, various tools with the potential of implementing the problem are compared and the best one is selected.

The rest of the paper is organized as follows. In section 2 we formally define the problem of dimensionality reduction and determine its input and output. Part 3 is dedicated to the application of dimensionality reduction in bioinformatics and the analysis of gene expression data. Part 4 studies the datasets collected for this paper and the preprocessing tasks performed on them. Part 5 summarizes overall state-of-the-art techniques and algorithms used for solving the dimensionality reduction in data science and machine learning, and picks the best one to use. Part 6 states the tools needed for implementing the problem, and the reason for choosing them.

2. Dimensionality Reduction Problem

In Data Science and Machine Learning, the problem referred to as the Dimensionality Reduction Problem (DRP) is the process of taking a dataset with a large number of features and much smaller number of samples as input, reducing the number of features to a reasonable amount, and presenting the new dataset with lower dimension as an output.

Assume that P and N represent the number of features and samples, respectively. We have $N \ll P$, meaning that the number of samples is much smaller than the number of features. The detailed input and output of the DRP are given below:

- *Input*
  The input of the problem is a data matrix named $X_{P\times N}$, consisting of P features placed on rows and N samples placed on columns. Each cell of X can be defined as follows:
  $$x_{ij} = the\ value\ of\ the\ i^{th} feature\ in\ the\ j^{th}\ sample$$
  where $x_{i.}$ is a vector representing the observation of the i^{th} feature in N samples, and similarly $x_{.j}$ is a vector reporting the value of P features in the j^{th} sample.
- *Output*
  The output of the problem is $\hat{X}_{K\times N}$, where K is a smaller dimension than P, and for the j^{th} sample vector of X we have new feature representations in $\hat{X}$ in the following form:
  $$\hat{x}_{.j} = T(x_{.j})$$

  where T is some transformation operator.

3. Application of dimensionality reduction in bioinformatics

In bioinformatics, understanding and predicting the behavior of genes (gene expression) is an important task. Gene expression refers to the process of transcription of a gene's DNA sequence into the RNA sequence, which is then used to produce proteins [2]. The gene expression level indicates how active a certain gene is in a certain tissue, at a certain time, or in some experimental conditions [2]. Measuring the expression levels represents an overall performance of the genes being studied, for example a virus, or some cancer cells [2].

There are many ways to model this behavior. One way of doing this is to use microarrays, or so-called gene expression data matrices. These matrices analyze huge amounts of genes' expression levels in different samples [2]. The rows of the matrix correspond to the genes being studied in a particular tissue, and represent the features. The columns of the matrix represent either different time fragments, or different samples (patients) being analyzed. Each data cell corresponds the particular gene's expression level, at the corresponding time, or in the patient being studied [2].

There are many machine learning models developed for classifying the microarrays. One application of this in bioinformatics is to classify different patients based on their gene expression levels and determine if they are healthy or cancerous [3]. Another application is to classify different cancer patients' responses to a particular drug, and determine if a certain person is resistant to a particular drug, or sensitive [3].

Developing classification models for gene expression data faces a serious challenge. The challenge is that gene expression data matrices usually have a huge number of features (genes) and a small number of samples [3][4]. If we represent the genes with P and the samples with N, this can be represented as $N \ll P$. In other words, $X_{P \times N}$ has thousands of rows, and a few hundred columns.

The $N \ll P$ property prevents the use of classification models based on deep learning methods, as the necessity there is the $P < N$ property, meaning that we should have lots of samples and smaller amounts of features [3][4].

This challenge strongly resembles the curse of dimensionality in computer science, and thus adjustments can be made to the dimensionality reduction problem to make it beneficial in bioinformatics, so that the presented challenge in gene expression matrices can be solved.

In this paper, we will analyze the DRP on the gene expression matrix $X_{P \times N}$, where P and N show the number of genes and samples, respectively. Each cell of X, namely x_{ij} represents the expression value of the i^{th} gene in the j^{th} sample.

4. Datasets

Two real time gene expression datasets will be used in this project, both downloaded from the GEO data bank.

The first one is called GSE99095, and is a single-cell RNA sequence dataset from bone marrow cells [3]. It has 17,258 rows (genes) and 979 columns (samples) in total. 391 of the samples belong to healthy control cells, and 588 cells are collected from 5 patients with bone-marrow failure [3]. The normalized gene expression matrix has been downloaded from the data bank, and is mainly used to classify patients based on their bone marrow cell status (healthy/diseased) [3]. Based on the GSE99095 data, the respective data matrix $X_{17,258\times979}$ includes the gene expression values of 17,258 genes of 979 cells. We define the set $Y = \{y_1, \dots, y_{979}\}$ as a label set of X as follows:

$$y_i = \begin{cases} 0, & if\ i\ is\ known\ to\ be\ an\ unhealthy\ cell, \\ 1, & else\ . \end{cases}$$

The next dataset is called GSE106291, and is an RNA sequence gene expression matrix collected from patients with acute leukemia under treatment [3]. There are 23,368 gene profiles (rows) and 250 patients (columns). 78 patients are labeled as resistant to treatment, while 172 patients are labeled sensitive [3]. Therefore, this dataset is used to classify patients as resistant or sensitive to treatment [3].

There are some preprocessing tasks needed for the GSE106291 dataset. Rows with 10% or more zero measurements need to be removed from the data, which leaves us with 11,068 rows [3]. Also, in every corresponding gene row, the expression values are normalized with the Z-score method, meaning that each value is subtracted by the mean of the row and divided by the standard deviation [3]. Based on the preprocessed GSE106291 data, the corresponding data matrix $X_{11,068\times250}$ includes the gene expression of 11,068 genes of 250 patients. We define the set $Y = \{y_1, \dots, y_{250}\}$ as a label set of X as follows:

$$y_i = \begin{cases} 0, & if\ i\ is\ known\ to\ be\ a\ patient\ resistant\ to\ treatment, \\ 1, & else\ . \end{cases}$$

5. Background of the problem

The techniques for solving DLP are divided into two main groups:

1. Feature Selection – where only the most important and useful features from the main data are kept [5][18].
2. Feature Extraction (also called Feature Representation or Dimensionality Reduction) – where a smaller subset of features is formed from the original dataset, where each feature is the combination of the original features, and yet this smaller set of new features contains the most important information from the previous dataset [18].

There are several techniques available in Data Science for both groups, a few of which have been summarized above.

5.1. Feature Selection

5.1.1. Filter Methods

Filter methods are based on using feature ranking algorithms as the principle criteria for selecting the features by ordering [5]. They are simple and practical. A suitable ranking criterion scores the features and then a specified threshold removes those above the threshold [5]. The reason why they're called filter methods is that they filter unimportant features and keep the most relevant ones. Correlation Coefficient (criteria) [6], Mutual Information (Information Gain) [7], Chi-Square Test [8], and Fisher Score [9] are commonly used filter methods.

5.1.2. Wrapper Methods

Wrapper methods see the classifier (predictor) as a black box for which its performance is the objective function, which we can use to evaluate the feature subset [5]. If we were to consider all possible combinations of features, 2^N different sets would've been available. Evaluating this many feature sets in an NP-hard problem, thus suboptimal feature sets can be obtained using search algorithms that work heuristically [5]. In wrapper methods, some search algorithms can be used to find a suboptimal subset of features which maximizes the objective function mentioned above, that is, the predictor performance [5]. Sequential Selection algorithms [10][11], Heuristic Search algorithms [12][13][14], and Recursive Feature Elimination algorithms [10][11] are all classified as wrapper methods.

5.1.3. Embedded Methods

Embedded methods combine the perks of filter and wrapper methods. The main goal of these methods is to reduce computational time needed for reclassifying the different sets of feature subsets, as done in wrapper methods [5]. They do this by incorporating the feature selection task as part of the training [5]. In other words, each learning algorithm has a feature selection process built-in. L1 (LASSO) Regularization [15][16], and Random Forests (RF) [17] using Decision Trees are embedded methods.

5.2. Feature Extraction

5.2.1. Methods based on Statistics and Information Theory

The methods categorized in this subsection reduce the dimension of the input data according to some statistical or information theory criteria [18]. They can capture non-linear relationships between variables, and have the ability of handling both categorial and interval features [18]. Vector Quantization (K-means) [19] and Mixture models [20], Principal Component Analysis [21], Manifolds [22], Generative

Topographic Mapping [23], Elastic Maps and Nets [24][25], Kernel PCA [26] and Multidimensional Scaling [27], and Factor Analysis [28-33] are some of the methods grouped in this category.

5.2.2. *Methods based on Dictionaries*

This family of techniques are based on matrix decompositions of the input data, where the input data matrix is transformed to a new data matrix with different feature representations [18]. This is simply done by a linear change of basis between the two sets of old and new feature matrices, where the algebraic basis of the matrices evolves [18]. The matrix representing the change of basis is called a Dictionary, and the elements are called Atoms [18]. Algorithms like Non-negative Matrix Factorization [34][35], Principal Tensor Analysis [36] and Non-negative Tensor Factorization [37], and Generalized SVD [38] into this sub-section.

5.2.3. *Methods based on projections*

The third category of algorithms view the dimensionality reduction problem as a projection problem [18]. That is, the input data is projected into a new subspace where it has some interesting characteristics [18]. Projection onto Interesting Directions [39][40], and Projection onto Manifolds [41][42] are two of the main techniques in this section.

Overall, different methods, models and theories have been proposed for the challenges we face in dimensionality reduction, some using feature selection and others using feature extraction, and the main goal of all these methods are learning meaningful feature representations with a reduced dimension, either by selecting some important features or by creating new ones.

With all being said, a comparison is need to be made so that a suitable technique can be chosen for our purposes.

Between feature selection and extraction methods, both are equally useful techniques for dimensionality reduction, and their difference mainly lies in the fact that feature selection keeps the original features while feature extraction combines and creates new ones [18]. For the purpose of our project, we wish to keep the original format of our genes and only select a subset of relevant genes, but combining different genes and creating new ones sounds a bit meaningless. So, we'll narrow down our chosen method to feature selection.

Now that the main subgroup is chosen, a comparison is again needed on strengths and weaknesses of different methods in feature selection. One advantage of filer methods is that they are fairly simple methods, without too many computations, and they avoid overfitting problems very well [5]. However, they suffer from the drawback that the selected feature subset may still be redundant [5]. Also, an ideal method for choosing the dimension of the newly selected feature space doesn't exist [5]. Wrapper methods have the advantage of choosing the optimal subset of features heuristically, and also the new dimensions are selected automatically. One main drawback of these methods is that they are computationally expensive, since for each subset evaluation, a new model is created and the predictor performance is tested [5]. For large samples, this is not efficient at all. Embedded methods on the other hand, have the advantages of both filter and wrapper methods since they are a combination of both, while trying to solve the problems in them such as computational complexity. With the comparison made here, clearly embedded methods are the winner.

From all embedded methods available, RF with decision trees will be used for our purposes. The reason of choosing RF is that it's a powerful ensemble technique obtaining prediction results from every single base tree, and not just from a single predicted score [3]. This aspect is very promising for selecting useful feature subsets. Also, we can easily evaluate how important the features are in each base tree [3].

6. Methods

The fDNN method for solving the DRP [3], consists of using Random Forests (RF) and DNNs. In the first step, RF is used as a primary feature extractor, which finds sparse and efficient feature representations from the raw input based on the prediction result of the trees in the forest. The forest consists of independent decision trees, and each tree makes an independent prediction. Therefore, we have an ensemble method.

In the second part, we feed the newly extracted features to a DNN and it makes predictions based on the sparse feature representations. A simple sketch of the fDNN architecture is given in Figure 1.

Training for the RF and the DNN are done separately, as follows in the next two sections.

6.1. Random Forest

A forest in the RF model is defined as a set of M independent trees:

$$F(\theta) = \{J_m(\theta_m)\}, \quad m = 1, \dots, M,$$

where $\theta = \{\theta_1, \dots, \theta_M\}$ represents the forest parameters. As stated above, we fit the raw data input $X_{N\times P}$ with N samples and P genes (features), and the corresponding data classification labels Y_N to the forest F, and train it. Afterwards, the entire samples (including both the train and test samples) are given to each tree in the forest, and for each sample, every tree makes a 0/1 label prediction. In other words, for each sample $x_i, i = 1, \dots, N$, a prediction is made by the forest F:

$$f(x_i, \theta) = f_i = \left(T_1(x_i, \theta_1), \dots, T_M(x_i, \theta_M)\right)^T$$

where $T_m(x_i, \theta_m) = \hat{y}_{im}$ is a binary prediction of the sample x_i made by J_m. In conclusion, f_i is a binary vector of length M predicted by individual trees in the forest F, given each instance x_i. The set of predictions for all samples are considered as extracted features in the form of a $F_{N\times M}$ matrix, later given to the DNN for the second training phase.

6.2. DNN

Next, before training the DNN, another $N \times M$ matrix is constructed similar to $F_{N\times M}$, with the difference that the 0 and 1 values are flipped with each other. In other words, the input of the DNN is defined a vector named I ($|I| = M$) as follows:

$$I[i] = \begin{cases} 10, & \textit{if the sample is predicted as positive by the ith tree} \\ 01, & \textit{else} \end{cases}$$

Thus, based on the decision of the trees for the i-th sample, vector f_i is generated, and then converted to vector i_i, and then fed to the DNN. So, the final feature representation is a $N \times M \times 2$ tensor. The same process described is also done for the entire label set Y, therefore the corresponding label set is in the form of a $N \times 2$ tensor. Before moving on to the DNN part, the newly constructed feature and label tensors are again split to train and test set tensors.

The output of the DNN is therefore defined as a vector named O ($|O| = 2$) as follows:

$$O[i] = \begin{cases} 10, & \textit{if the sample is predicted as positive by the DNN} \\ 01, & \textit{else} \end{cases}$$

The DNN network is a standard deep fully-connected network. Details on the architecture and hyper parameters of the DNN are given in section 7.

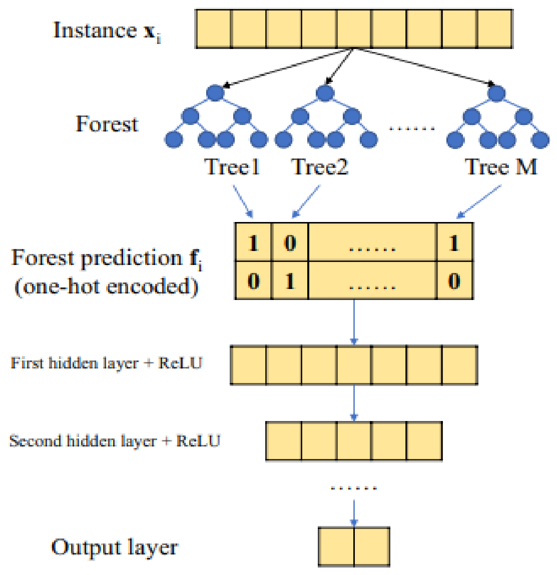

Figure 1-Visualization of the architecture of the fDNN model

Next, we implement an additional method, where the feature extraction is done differently, by using a stacked auto-encoder, and then the final prediction is done by the DNN described in the previous section. More details are given in the sub-section below.

6.3. *Stacked Auto-encoder + DNN*

This method is also composed of two parts, a feature extraction part and a training part.

For feature extraction, we use a stacked auto-encoder architecture described in [43], and extract efficient low-dimensional feature vectors from the original data. Stacked auto-encoders have proven to be an excellent tool for feature extraction, by mapping the original feature space into a low-dimensional sub-space containing all the important information in a small sized vector.

The stacked auto-encoder used in this project is simply stacked layers of simple fully-connected auto-encoders stacked on top of each other, where the output of each layer is fed to the next layer as an input. Each layer is actually a batch normalization layer, followed by a dense layer. A sketch on the architecture of the auto-encoder is given in Figures 2 and 3.

The data is partitioned into train and test sets, and we discard the labels as we are only interested in coding and decoding the samples, in an unsupervised manner. In other words, each sample $x_i, i = 1, \dots, N$ is given to the network as an input. The encoder maps the features to a low-dimensional vector using some hidden layers, and afterwards tries to reconstruct x_i using the decoder layers. For doing so, a loss function is used to compute the error between x_i and the reconstructed $\widehat{x_i}$. Afterwards, the test set is used to evaluate the reconstruction accuracy of the auto-encoder.

Here, we are interested in the so-called bottleneck layers, or the last layer of the encoder containing the minimum low-dimensional feature vector, which will be our new features. After training the network, we make a prediction using the whole data (both train and test) and obtain the feature vector in the bottleneck layer, as our new features.

In conclusion, we now have a feature matrix called $F_{N \times L}$, where $L \ll P$ is the dimension of the extracted feature vector from the auto-encoder, and its corresponding label vector Y, as we had before, and are ready to make final predictions.

We do a train-test split again, on the new features and labels, train the DNN described in the previous section, and make final predictions using the test set. The only difference lies in the fact that we don't need to flip the 0s and 1s anymore, so the input of the DNN is not a tensor anymore.

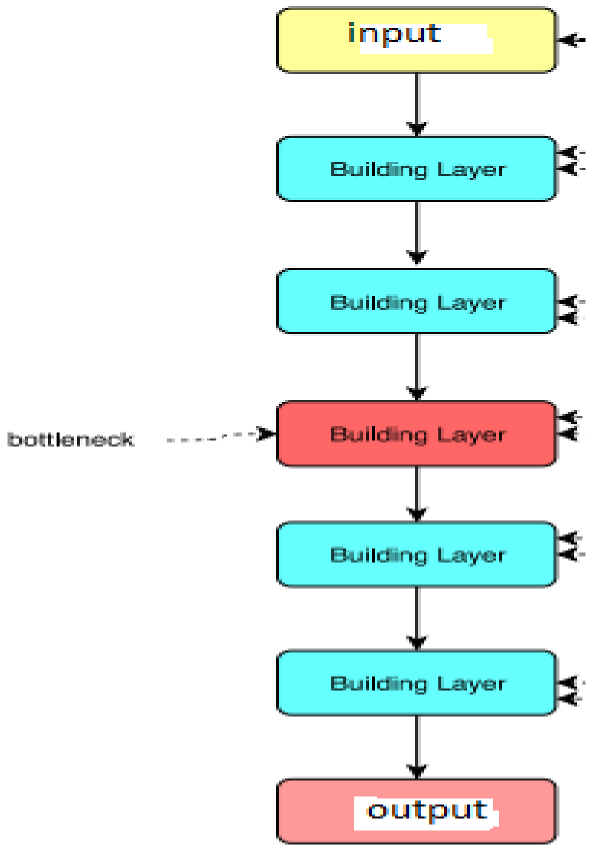

Figure 2-Visualization of the architecture of the stacked auto-encoder

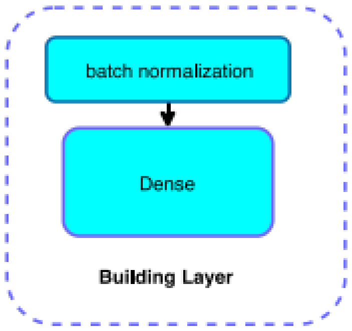

Figure 3-architecture of each layer in the auto-encoder

7. Results and Discussion

The topics of this section are about the train/test split portion on the datasets, tools used for implementing the models, the evaluation metric used to test the performances, results obtained by implementing every model, and finally a brief comparison discussion and between the models based on their results.

7.1. Train/Test split

As mentioned above, the data matrix has genes (features) placed on rows and samples placed on columns. For implementation purposes, the data matrix is transposed at the beginning so that samples are placed on rows and genes are placed on columns.
We then separate the train and test data randomly, with sizes mentioned in Table 1:

Table 1-Train and Test set sizes for GSE99095 and GSE106291

Dataset	Train/Test Split
GSE99095	700/279
GSE106291	200/50

7.2. Tools for implementation

The methods and implementations are coded in python and executed in Google Colab, while the data preprocessing part is conducted in R and by using Rstudio. The fDNN method is implemented using sklearn.tree and sklearn.ensemble for the RF part, and the DNN is coded using tensorflow. The SVM models described later in this section are also implemented by sklearn.svm.

7.3. Evaluation Metric

The performance and evaluation metric chosen for this project is the AUC out of ROC curve. The ROC (Receiver Operating Characteristic) curve shows the performance of any classification model at all thresholds by plotting the True Positive Rate (TPR) vs. the False Positive Rate (FPR). AUC is defined as the Area Under the Curve in the ROC plot, and is used as the test performance metric. One way of interpreting AUC is as the probability that the model ranks a random positive sample more highly than a random negative sample. It ranges in value from 0 to 1, and if a model has an AUC of 0 it means that its predictions are 100% wrong. Similarly, a model whose predictions are 100% correct has an AUC of 1. So, the higher the AUC, the better the model is at distinguishing between different classes.

As GSE106291 is quite imbalanced, typical accuracy, precision, and recall metrics may not show the real performance. The AUC works better on imbalanced datasets, and gives more accurate results, therefore it is suitable for this problem.

7.4. Results

Details about different model architectures, and their performance results are given in the following sections.

7.4.1. fDNN

As mentioned before, the fDNN method consists of two parts: RF and DNN. The following two sections discuss the results and architectures regarding these two parts, respectively.

7.4.1.1. RF

To test the performance of the RF in terms of feature extraction, the test tensor described in section 6.2. is given to the RF, and it makes a final prediction as a whole ensemble (and not single predictions per tree). The performance results, and the optimal number of trees in the forest are summarized in Table 2:

Table 2 -Feature Extraction performance results for different RF architectures.

Dataset	RF Architecture	AUC test score
GSE99095	400	0.99
GSE99095	1000	0.95
GSE106291	500	0.74
GSE106291	1000	0.66

To test whether the number of trees was optimal or not, the procedure was repeated with 1000 trees as well, for both datasets, with results present in the table.

To further confirm that flipping values and making new feature tensors was appropriate, training and testing the DNN was repeated, this time without constructing tensors and flipping values, and the input of the DNN was in its original matrix form instead of tensors. By doing so, the final performance of the entire fDNN model decreased critically.

No hyper-parameter was set for individual trees, and the default values of the sklearn package were used.

7.4.1.2. DNN

The architecture of the DNN is as follows in Table 3:

Table 3-DNN Architecture

DNN	
Architecture	256,64,16
Activation Function	ReLU, Softmax
Optimizer	Adam
Batch Strategy	Mini-Batch
Loss Function	Cross-Entropy
Backpropagation	SGD

The number of hidden layers and units in each layer are the same for both datasets used in the study. These hyper-parameter values are obtained using cross-validation.

The final results of the complete fDNN model are summarized in Table 7:

Table 4-AUC test score for fDNN

Dataset	AUC test score
GSE99095	0.99
GSE106291	0.74

7.4.2. Nonlinear SVM with Sigmoid kernel

Table 5-AUC test score for Nonlinear SVM with Sigmoid kernel

Dataset	AUC test score
GSE99095	0.97
GSE106291	0.67

The Nonlinear SVM model using the Sigmoid kernel was also implemented on both datasets. The data was again split into train/test sets, and the model was trained and tested using the sets. No feature selection/extraction was done and all genes were present in the experiment. Results are displayed in Table 4.

7.4.3. Linear SVM with full set of features

Table 6-AUC test score for Linear SVM with full set of features

Dataset	AUC test score
GSE99095	0.97
GSE106291	0.61

A simple Linear SVM model using all features was implemented using the same procedure described in the previous part. Again, no feature extraction was done in this part. Results are displayed in Table 5. Results for feature extraction paired with the same Linear SVM model is described in the next part.

7.4.4. Linear SVM with feature extraction

Table 7-AUC test score for Linear SVM with feature extraction

Dataset	Number of important features	AUC test score
GSE99095	400	0.49
GSE106291	500	0.57

For the sake of trying feature extraction paired with SVM, after training the Linear model from the previous part, the most important features were detected using their coefficients values in both models, and only those features were kept. Then, the linear SVM was again trained and tested using these selected features, as seen in Table 6.

7.4.5. Stacked Auto-encoder + DNN

The architecture of the stacked auto-encoder is as follows in Table 8:

Table 8-Stacked Auto-encoder architecture

Stacked Auto-encoder		
Architecture	**GSE99095**	8192,1024,128,2048
	GSE106291	8192,1024,64,2048
Activation Function	Softplus	
Optimizer	Nadam	
Learning Rate	0.0001	
Batch Strategy	Mini-Batch	
Loss Function	Mean-Squared Error	
Backpropagation	SGD	

The DNN architecture is as described in section 7.4.1.2. The train/test split portions are also the same as before.

The final results of the stacked auto-encoder + DNN is summarized in Table 9:

Table 9-AUC test score for stacked auto-encoder+DNN

Dataset	AUC test score
GSE99095	0.94
GSE106291	0.74

7.5. Conclusion

Regarding the RF feature extraction described in section 7.4.1.1, two important conclusions are drawn. First, the experiment showed that 400 and 500 were indeed the right number of trees for capturing the most important features, compared to 1000 trees, and increasing the numbers led to less important features selected, therefore worse results.

Second, flipping values of the extracted f_is and constructing i_is, were necessary for good performance in the DNN part. As the final layer in the DNN consisted of a softmax, it eventually gave binary outputs. When the conversion wasn't done, the DNN ignored 0 values and mostly predicted 1s, despite the fact that both labels are just numerical representations. That’s why the regular input DNN results were worse than feature tensor DNN.

By implementing fDNN on GSE99095, it gave the best prediction results amongst all models, and the outcomes were slightly worse for GSE106291, but still the highest score achieved compared to other models. As the sample size was quite small and we had imbalanced labels, this result was expected. If more samples were present, the fDNN model would perform even better on GSE106291.

Nonlinear SVM gave the second-best results on both datasets. The AUC results were only a bit lower than fDNN, which shows that nonlinear SVM with sigmoid kernel is still a very powerful classification model, specially, considering the fact that no feature extraction was done beforehand, in contrast with the fDNN model. Thus, this model is worth trying and experimenting before moving on to more complicated DNN and RF models.

Linear SVM with full features gave a similar test score with nonlinear SVM on GSE99095, but performed worse than nonlinear SVM on GSE106291. This shows that in the case where all features are present, nonlinear kernels generalize better and make more accurate predictions.

Regarding Linear SVM with feature selection, results were somehow disappointing, as it had the lowest scores on both datasets compared to other models. The score for GSE99095 was even less than 0.50, which would be obtained by simply guessing the labels. Similarly, GSE106291 gave performed only a little better than random guessing. So, it seems that feature extraction using linear SVM doesn’t give the desirable results, as opposed to nonlinear SVM or even linear SVM without feature extraction.

Finally, the auto-encoder + DNN model gave a test score of 0.94 on GSE99095, which was lower than fDNN, nonlinear SVM and full linear SVM, but better than SVM with feature extraction. It gave a test score of 0.74 on GSE106291, similar to fDNN, and was better than other models. It seems that feature extraction in this manner is as efficient as random forests, on imbalance datasets. The feature mapping is done automatically by finding the right weights and biases and minimizing the loss, and there is no need for creativity and novel solutions, as we saw in random forests. It is easier to train and there is no need to flip the values and make tensors for prediction, therefore it is worth consideration. Also, it is more compatible with the DNN, since the auto-encoder is a deep network architecture itself.

References

1. Han, Jiawei, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

2. Xu, Min. "Global Gene Expression Analysis Using Machine Learning Methods." *arXiv preprint arXiv:1506.02087* (2015).
3. Kong, Yunchuan, and Tianwei Yu. "A deep neural network model using random forest to extract feature representation for gene expression data classification." *Scientific reports* 8.1 (2018): 1-9.
4. Daoud, Maisa, and Michael Mayo. "A survey of neural network-based cancer prediction models from microarray data." *Artificial intelligence in medicine* (2019).
5. Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16-28.
6. Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, *3*(Mar), 1157-1182.
7. Battiti, R. (1994). Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, *5*(4), 537-550.
8. Pearson, K. (1900). X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *50*(302), 157-175.
9. Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.
10. Pudil, P., Novovičová, J., & Kittler, J. (1994). Floating search methods in feature selection. *Pattern recognition letters*, *15*(11), 1119-1125.
11. Reunanen, J. (2003). Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, *3*(Mar), 1371-1382.
12. Sastry, K., Goldberg, D., & Kendall, G. (2005). Genetic algorithms. In *Search methodologies* (pp. 97-125). Springer, Boston, MA.
13. Eshelman, L. J. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of genetic algorithms* (Vol. 1, pp. 265-283). Elsevier.
14. Cordón, O., Damas, S., & Santamaría, J. (2006). Feature-based image registration by means of the CHC evolutionary algorithm. *Image and Vision Computing*, *24*(5), 525-533.
15. Santosa, F., & Symes, W. W. (1986). Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, *7*(4), 1307-1330.
16. Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267-288.
17. Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.
18. Sorzano, Carlos Oscar Sánchez, Javier Vargas, and A. Pascual Montano. "A survey of dimensionality reduction techniques." *arXiv preprint arXiv:1403.2877* (2014).
19. Gray, R. (1984). Vector quantization. *IEEE Assp Magazine*, *1*(2), 4-29.
20. Bailey, T. L., & Elkan, C. (1994). Fitting a mixture model by expectation maximization to discover motifs in bipolymers.
21. Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37-52.
22. Smola, A. J., Williamson, R. C., Mika, S., & Schölkopf, B. (1999, March). Regularized principal manifolds. In *European Conference on Computational Learning Theory* (pp. 214-229). Springer, Berlin, Heidelberg.
23. Bishop, C. M., Svensén, M., & Williams, C. K. (1998). GTM: The generative topographic mapping. *Neural computation*, *10*(1), 215-234.
24. Gorban, A. N., Karlin, I. V., & Zinovyev, A. Y. (2004). Constructive methods of invariant manifolds for kinetic problems. *Physics Reports*, *396*(4-6), 197-403.
25. Gorban, A. N., Kégl, B., Wunsch, D. C., & Zinovyev, A. Y. (Eds.). (2008). *Principal manifolds for data visualization and dimension reduction* (Vol. 58, pp. 96-130). Berlin: Springer.

26. Schölkopf, B., Smola, A., & Müller, K. R. (1997, October). Kernel principal component analysis. In *International conference on artificial neural networks* (pp. 583-588). Springer, Berlin, Heidelberg.
27. Cox, M. A., & Cox, T. F. (2008). Multidimensional scaling. In *Handbook of data visualization* (pp. 315-347). Springer, Berlin, Heidelberg.
28. Spearman, C. (1961). " General Intelligence" Objectively Determined and Measured.
29. Thurstone, L. L. (1947). Multiple-factor analysis; a development and expansion of The Vectors of Mind.
30. Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and psychological measurement*, *20*(1), 141-151.
31. Lawley, D. N., & Maxwell, A. E. (1971). *Factor analysis as statistical method* (No. 519.5 L3 1971).
32. Mulaik, S. A. (2009). *Foundations of factor analysis*. CRC press.
33. Harman, H. H. (1976). *Modern factor analysis*. University of Chicago press.
34. Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788-791.
35. Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems* (pp. 556-562).
36. Cichocki, A., Zdunek, R., Phan, A. H., & Amari, S. I. (2009). *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons.
37. Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis.
38. Paige, C. C., & Saunders, M. A. (1981). Towards a generalized singular value decomposition. *SIAM Journal on Numerical Analysis*, *18*(3), 398-405.
39. Watson, A. B. (1993, September). DCT quantization matrices visually optimized for individual images. In *Human vision, visual processing, and digital display IV* (Vol. 1913, pp. 202-216). International Society for Optics and Photonics.
40. Kaski, S. (1998, May). Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)* (Vol. 1, pp. 413-418). IEEE.
41. Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, *100*(5), 401-409.
42. Kruskal, J. B. (1978). *Multidimensional scaling* (No. 11). Sage.
43. Moridi, M., Ghadirinia, M., Sharifi-Zarchi, A., & Zare-Mirakabad, F. (2019). The assessment of efficient representation of drug features using deep learning for drug repositioning. *BMC bioinformatics*, *20*(1), 577.