Artificial Intelligence Course Project: Use Machine Learning to Classify Gene Chips

Yanjun Fu
Department of Computer Science and Engineering, SJTU

Abstract

Gene sequences determine the features of creatures and gene mutations usually cause serious health problems, so it is important to understand how gene chips can affect human from many aspects. In this course project, we are going to use some techniques of artificial intelligence, especially techniques of machine learning, to classify different gene chips and enchance the understanding of genes. We use some classical machine learning methods like PCA for dimensionality reduction, SVM, logistic regression and k-NN to build classifier. Moreover, we follow the current tide of deep learning to explore the performance of it. In this report, we will first review the theory and the insights of these methods, then discuss our design, and we show the results of experiments and evaluation of our models finally.

Key Words: *Gene chips classification, machine learning, PCA, logistic regression, SVM, k-NN, deep learning.*

1. Introduction

Nowadays, understanding the nature of genes and how they influence human's life is not only the task of biologists, but also a possible task of computer scientists. More and more people hope to use the development of computer science especially artificial intelligence and data mining techniques to understand genes, and apply them into diagnosing disease or providing more reasonable therapeutic regimens.

In this project, we need to use methods of traditional machine learning and methods of deep learning to build a mutli-label classifier for genes with lots of features. Possible methods include using PCA [1] to do dimensionality reduction and using techniques such as logistic regression, support vector machine (SVM) [2], k nearest neighbors (k-NN) [3] or deep neural networks to build an effective classfier.

The rest part of this report mainly contains three parts, which are introduction of the insights and theories of these machine learning methods, experiments and evaluation, and some deeper discussion about this problem.

Our contributions are listed as below:

- using PCA and other methods to pre-process naive data;
- designing and implementing many kinds of machine learning methods to build multi-label classifiers;
- doing experiments to show the better choice of hyper-parameters and the performance of our design;
- comparing deep learning methods with other traditional machine learning methods and proposing some possible future work.

2. Methods

In this part, we will first discuss the data processing method PCA and some classical and traditional machine learning methods to do gene chips classification, including logistic regression, SVM and k-NN. Then we will propose a proper deep neural network to classify the gene chips. We mainly discuss the theories and the insights of these methods.

2.1. Data Processing with PCA

2.1.1 Importance of Data Processing

Usually, when we are talking about pre-processing data, we hope to find a method which makes this problem discussible in Euclidean space and makes the complexity of data is compatible with our problems or our models. Specify the discussion to this project we are working at, we will see that the dataset gives us n gene chips with p features and $p = 22283$ while $n = 5896$, which shows that the number of features is much more than the size of the given data ($p >> n$). What is obvious is that the features are not all independent and the problem is not located in Euclidean space. In this case, using dimensionality reduction to transfer the problem to a lower-dimensional space and make the dimensionality compatible with the data size is necessary to make the problem feasible.

PCA is now a commonly used statistical method to processing the data.

2.1.2 Theory of PCA

PCA, which is the short of Principal Component Analysis, is aimed to orthogonally transfer a set of possibly correlated variables that was located in high-dimensional space to linearly uncorrelated variables located in lower-dimensional space. We hope to find a hyperplane, for which the sum of distance between all the data and the hyperplane is minimized or for which the projection of all data is as discrete as possible, i.e. make the variance of the projection of the data as great as possible. These linearly uncorrelated variables are called **principal components**.

Interestingly, from either the aspect of minimizing the sum of distance or the aspect of maximizing the variance of the projection, we can get the same mathematical representation, which means these two aspects is mathematically equivalent. Here is the mathematical derivation of PCA from the aspect of maximizing the variance of the projection of data like it is shown in Figure 1.

$$\begin{aligned}\frac{1}{m}\sum_{i=1}^{m}\left({\boldsymbol{x}^{(i)}}^T\boldsymbol{u}\right)^2 &= \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{u}^T\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T\boldsymbol{u} \\ &= \boldsymbol{u}^T\left(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T\right)\boldsymbol{u}\end{aligned} \tag{1}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T. \tag{2}$$

Then we have a constrained optimization problem,

$$\begin{aligned}\max \quad & \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} \\ s.t. \quad & \boldsymbol{u}^T\boldsymbol{u} = 1.\end{aligned} \tag{3}$$

Then we use general Lagrangian,

$$L(\boldsymbol{u},\lambda) = \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} - \lambda(\boldsymbol{u}^T\boldsymbol{u} - 1). \tag{4}$$

To find the optimal solution,

$$\begin{aligned}\nabla_{\boldsymbol{u}} &= \boldsymbol{\Sigma}\boldsymbol{u} - \lambda\boldsymbol{u} = 0 \\ \boldsymbol{\Sigma}\boldsymbol{u} &= \lambda\boldsymbol{u}.\end{aligned} \tag{5}$$

Thus, we see that PCA are **the eigenvectors of a covariance matrix**.

2.1.3 PCA Using SVD

Here we show one of the algorithm to solve PCA which uses **Singular Value Decomposition** (SVD).

Formally, the singular-value decomposition of an $m \times n$ real or complex matrix $\boldsymbol{A}$ is a factorization of the form

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times m}\boldsymbol{\Sigma}_{m\times n}\boldsymbol{V}^T_{n\times n}, \tag{6}$$

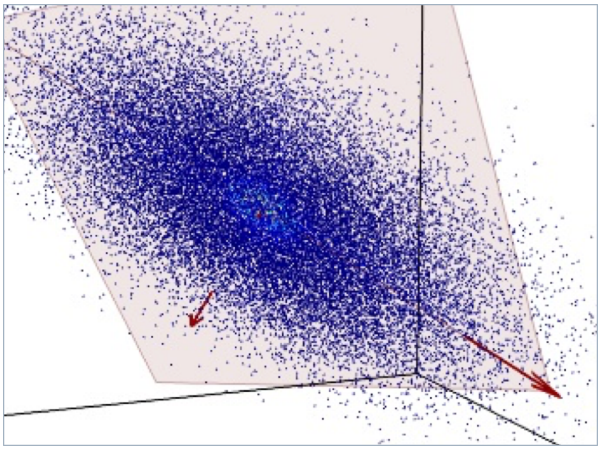

Figure 1. **PCA Feature Selection.** To maximize information representation(variability).

where $\boldsymbol{U}$ is an $m \times m$ real or complex unitary matrix, $\boldsymbol{\Sigma}$ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and $\boldsymbol{V}$ is an $n \times n$ real or complex unitary matrix. The diagonal entries σ_i of $\boldsymbol{\Sigma}$ are known as the singular values of $\boldsymbol{A}$.

Singular value σ_i is arranged incresingly in $\boldsymbol{\Sigma}$, and the declining speed is very fast. Thus, a small part of singular values can take up the most part of the sum of all singular values. Then we can get $\boldsymbol{A}$ approximately,

$$\boldsymbol{A}_{m\times n} \approx \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}\boldsymbol{V}^T_{r\times n} \quad (r << m). \tag{7}$$

Recall the PCA method, we need to transfer features from one higher dimension to a lower dimesion, i.e. to find a $r < m$ satisfying:

$$\boldsymbol{A}_{m\times n}\boldsymbol{P}_{n\times r} \approx \boldsymbol{A}'_{m\times r} \tag{8}$$

Thus we can take advantage of SVD,

$$\begin{aligned}\boldsymbol{A}_{m\times n} &\approx \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}\boldsymbol{V}^T_{r\times n} \\ \boldsymbol{A}_{m\times n}\boldsymbol{V}_{r\times n} &\approx \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}\boldsymbol{V}^T_{r\times n}\boldsymbol{V}_{r\times n}.\end{aligned} \tag{9}$$

We have $\boldsymbol{V}^T\boldsymbol{V} = 1$, then

$$A_{m\times n}\boldsymbol{V}_{r\times n} \approx \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}. \tag{10}$$

We can use $\boldsymbol{V}$ to do the transfer.

2.2. Logistic Regression

Logistic models use logistic function to model a binary dependent variable, logistic regression is to estimate the parameters of a logistic model. The definition of logistic function is as below and its shape is shown in Figure 2.

$$\sigma(t) = \frac{1}{1+e^{-t}} \tag{11}$$

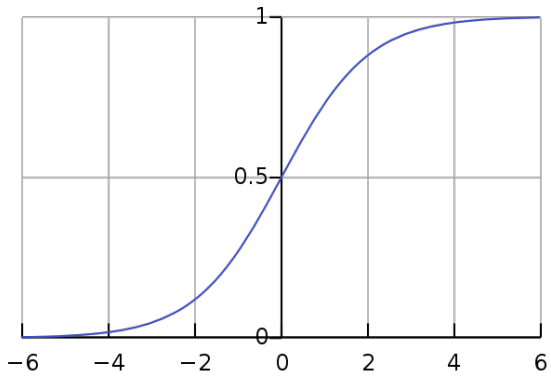

Figure 2. **Standard Logistic Function.** Note that $\sigma(t) \in (0, 1)$ for arbitrary t.

In logistic regression, define $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$. Assume a event obeys Bernoulli distribution, i.e. $P(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $P(y = 0|\boldsymbol{x}, \boldsymbol{\theta}) = 1 - f_{\boldsymbol{\theta}}(\boldsymbol{x})$. Then $p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = (f_{\boldsymbol{\theta}}(\boldsymbol{x}))^y (1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}))^{1-y}$

Use MLE, we can get

$$\begin{aligned} l((\theta)) &= \log L(\boldsymbol{\theta}) \\ &= \log \prod_{i=1}^{m} p(y_i|x_i, \boldsymbol{\theta}) \\ &= \sum_{i=1}^{m} y_i \log f_{\boldsymbol{\theta}}(x_i) + (1 - y_i)(1 - \log f_{\boldsymbol{\theta}}(x_i)) \end{aligned} \tag{12}$$

And the loss function of logisitic regression is:

$$\begin{aligned} L(\boldsymbol{\theta}) = -\left[\frac{1}{m} \sum_{i=1}^{m} y_i \log f_{\boldsymbol{\theta}}(x_i) + (1 - y_i)(1 - \log f_{\boldsymbol{\theta}}(x_i))\right] \\ + \lambda R(\boldsymbol{\theta}) \end{aligned} \tag{13}$$

2.3. Support Vector Machine (SVM)

2.3.1 Introduction

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. The core insight of SVM is to use a hyperplane to divide the data into two categories that are mutual exclusive. The goal of optimization is to maxmize the margin between two different categories. Here is the simple example of SVM in Figure 3.

Here is the mathmetical representation of SVM,

$$\begin{aligned} \min_{\boldsymbol{\omega}, b} \quad & \frac{1}{2}||\boldsymbol{\omega}||^2 \\ s.t. \quad & y_i(\boldsymbol{\omega}^T \boldsymbol{x_i} + b) \geqslant 1 \quad i = 1, 2, \cdots, m \end{aligned} \tag{14}$$

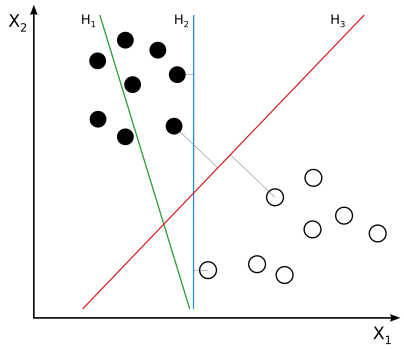

Figure 3. **A Example of SVM.** H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximal margin.

2.3.2 Kernel Trick

Given a problem, it may be not linear separable in current dimensional space, we need to project all of the data to a higher-dimensional space with a function $\Phi(\boldsymbol{x_i})$ used for projection. Fortunately, it can be proved that if the number of dimensions of current space is finite, then we can always find a space that is higher-dimensional, and the projection of given problem in this new space is linear separable.

However, the function $\Phi(\boldsymbol{x_i})$ for projection is hard to find and then kernel trick was proposed to solve this difficulty. The mathmetical representation for kernel function is:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) =< \Phi(\boldsymbol{x_i}), \Phi(\boldsymbol{x_j}) >= \Phi(\boldsymbol{x_i})^T \Phi(\boldsymbol{x_j}). \tag{15}$$

Kernel function directly calculates the product of vectors and avoid the difficulty to find $\Phi(\boldsymbol{x_i})$, but we still need to make appropriate assumption of a kernel. Here are some common kernel functions:

1. linear kernel:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i}^T \boldsymbol{x_j}; \tag{16}$$

2. polynomial kernel:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \left(\gamma \boldsymbol{x_i}^T \boldsymbol{x_j} + b\right)^d, \quad b > 1; \tag{17}$$

3. RBF(Gaussian) kernel:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-\gamma||\boldsymbol{x_I} - \boldsymbol{x_j}||^2), \gamma > 0; \tag{18}$$

4. sigmoid kernel:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = tanh(\gamma \boldsymbol{x_i}^T \boldsymbol{x_j} + b), \gamma > 0, b < 0. \tag{19}$$

2.3.3 Soft Margin

Sometimes current problem is not linear separable in its space, though we can use kernel trick to project it into a higher-dimensional space and make it separable, we still do not know if overfitting has occurred. Thus we allow some data do not fullfill the strict margin (called hard margin) of SVM, then we have the concept of soft margin. Here is the mathmetical representation of soft margin:

$$\begin{aligned} \underset{\boldsymbol{\omega},b,\boldsymbol{\xi}}{min} \quad & \frac{1}{2}||\boldsymbol{\omega}||^2 + C\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y_i(\boldsymbol{\omega}^T\boldsymbol{x_i} + b) \geqslant 1 - \xi_i \quad i = 1, 2, \cdots, m \\ & \xi_i \geqslant 0 \quad i = 1, 2, \cdots, m \end{aligned} \tag{20}$$

2.4. k-Nearest Neighbors (k-NN)

The core idea of k-NN is very simple, which is to find k nearest points of one point based on certain measurement and predict information for this point based on the information of the k neighbors. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

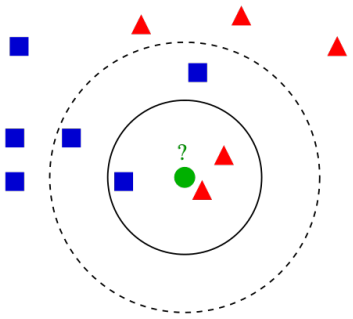

Figure 4. **An Example of k-NN.** The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$(solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

2.5. Deep Learning

2.5.1 Fully Connected Neural Network

Since the features do not have any locality or direct relationship, no convolutions are necessary. In this case, fully connected neural network is the most approriate neural network structure. We first discuss a simple fully connected layer as shown in Figure 5.

In this example,

$$\begin{aligned} y_1 &= f(\boldsymbol{W}_{11}x_1 + \boldsymbol{W}_{21}x_2 + \boldsymbol{W}_{31}x_3 + \boldsymbol{b}_1); \\ y_2 &= f(\boldsymbol{W}_{12}x_1 + \boldsymbol{W}_{22}x_2 + \boldsymbol{W}_{32}x_3 + \boldsymbol{b}_2); \\ y_3 &= f(\boldsymbol{W}_{13}x_1 + \boldsymbol{W}_{23}x_2 + \boldsymbol{W}_{33}x_3 + \boldsymbol{b}_3), \end{aligned} \tag{21}$$

where $f(\cdot)$ is the activation function, $\boldsymbol{W}$ is the matrix of parameters and $\boldsymbol{b}$ is the bias. $\boldsymbol{W}$ and $\boldsymbol{b}$ are learnable parameters.

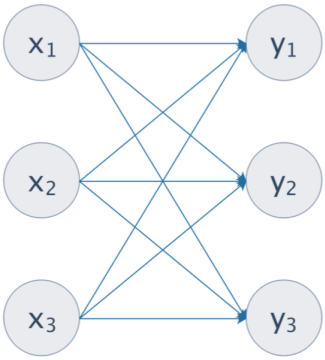

Figure 5. **A Simple Example of Fully Connected (FC) Layer**

2.5.2 Our design

Based on the view that fully connected neural network is proper and the encoding-decoding strcture is useful, our design has **three hidden layers**, where the hidden layer 1 is of dimension 256, the hidden layer 2 is of dimension 512 and the hidden layer 3 is of dimension 256. We have also tried some simpler or more complex models and we will prove this three-hidden-layer model's performance is the best later. The dimension of the input layer is 453 based the result of dimensionality reduction based on PCA. The dimension of output layer is flexible and it is determined by the number of categories we need to classify. Figure 6 shows the brief view of the structure of our design.

In our model, activation function is $\text{ReLU}(x) = \max(0, x)$, the optimization method is **Stochastic Gradient Descent** (SGD), the loss function is **Cross Entropy Loss** [4], and its mathematical representation is:

$$L = \sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}). \tag{22}$$

What's more, we use **batch normalization** [5].

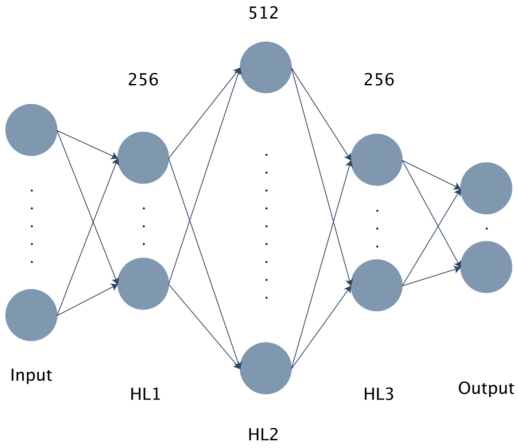

Figure 6. **The Structure of Our Design**

2.5.3 Regularization

Here we use two kinds of regularization, **L2 norm** and **Dropout** [6] to test these two methods' performance.

- **L2 norm** L2 norm's mathematical representation is $||\boldsymbol{\theta}||_2 = \sqrt{\sum_{i=1}^{n} |\theta_i|^2}$.
- **Dropout** Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. For dropout, we have a probability of dropping a neuron and this probability obeys **Bernoulli distribution**.

3. Experiments and Evaluation

In this section, we will discuss some of the details of our model implementation, experiments and evaluation, including how to choose hyper-parameters, how to determine the metric of evaluation, how to choose and divide datasets and some obervations from experiment results, etc.

3.1. Data Selection and Preparation

In this project, we have 5896 gene chips and they have 22283 features. For all of the gene chips, they have more than 10 labels or characteristics.

Among more than 10 labels or characteristics, we choose *"Sex"*, *"CellLine"* and *"DiseaseState"* as the goal of learning of our models.

For any of the three selected labels, it makes our learning problem a multi-classification task. Before we start to run our model, we need to get the data well prepared. We can not ignore that lots of gene chips are not valid data when we talk about one of these three labels, because they do not labeled when we do classification. For these invalid gene chips, we need to drop them from our train sets and test sets

Another thing that needs our attention is that the amount of data does not uniformly distribute among all the categories, especially for label *"DiseaseState"*. There are 193 categories under label *"DiseaseState"* and the amount of data for most categories is not enough to train a promising model, so unfortunately, we need to drop most categories. The strategy to address the issue of lacking data is:

- Merging categories that are very close, for instance, the different stages of one kind of disease will be merged to one category.
- Set a threshold after merging close categories, if the amount of data of one category is lower than this threshold, then this category will be dropped . We set the threshold with two values that are 10 and 50, then we get two datasets called *"Diseasestate16"* with threshold of 50 and *"DiseaseState70"* with threshold of 10. They have 16 categories and 70 categories as mentioned in their names.

Table 1 shows the number of categories and the amount of samples of each separated dataset:

Table 1. **Number of Categories and Amount of Data for Different Separated Dataset**

Dataset	# Categories	# Samples
Sex	7	1675
CellLine	97	1142
DiseaseStare16	16	2452
DiseaseState70	70	3358

3.2. Method and Metric for Evaluation

3.2.1 Cross Validation

We use cross validation to get more convincing evaluation results. Cross validation is a better way to evaluate a model, for it can avoid the possible locality and sequentiality of a data sequence. It combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance.

Since the numbers of samples are not uniformly distributed, we need to divide samples with the same category to five equal parts for every single category. 80% of samples of each category are used to train, and the others are used to test. Finally, we combine the train or test data of each category to get the full train or test data.

3.2.2 Metric

Since all of the four datasets we select shown in Table 1 have mutiple categories, we use accuracy which is mesured

by correctly predicted samples divide all of the test samples, i.e. $accuracy = acc\_num/total \times 100\%$.

3.3. Tools for Implementation

All of our work are implemented based on Python 3.x. For classical machine learning methods like PCA, logistic regression, SVM and k-NN, we use the library *scikit-learn* [7] to implement. For our deep neural network, we use the very famous deep learning framework *PyTorch* [8] to implement. All of the images that show the results of experiments and evaluation is based on the *Matplotlib* library.

3.4. Dimensionality Reduction by PCA

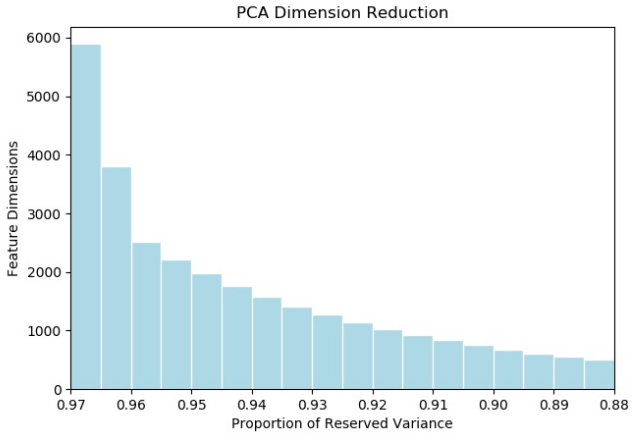

Figure 7. **Changes of Dimensions While Reserved Variance Changing**

There are 5896 ($n = 5986$) gene chips and a gene chip has 22283 ($p = 22283$) features so that the number of dimensions of much larger than the number of samples ($p >> n$), which means we need to reduct dimensionality to reduce the redundance of fetures and reduct the dependency between features.

We try different proportion of reserved variance to do PCA and observe how the number of rest features changes, and the result is shown in Figure 7. We can see that the number of reserved features reducts significantly when the reserved variance just change a little at first (a jump happened at first). This phenomenon indicts that most part of useful information is mantained by only a small part of features after PCA in this project. Through test, we find that only 453 features (only about 0.02% of original features) can maintain 90% of information.

3.5. Logistic Regression

Before doing logistic regression, we use PCA to reduct the dimension of features to 1000. As discussed in section 3.1, we use 80% of data to train the model and use the rest 20% to evaluate the performance of our model. The parameters of logistic regression is set as default at first.

We record the intermediate results of our logistic regression as shown in Figure 8. We can see that the speed of convergence is fast for all of these 3 datasets and the final accuracy are all over 95%. The detailed accuracy for these three datasets are listed in Table 2.

Table 2. **LR Accuracy on All the Three Datasets.**

Dataset	Sex	DiseaseState70	CellLine
Accuracy	95.02%	95.88%	97.37%

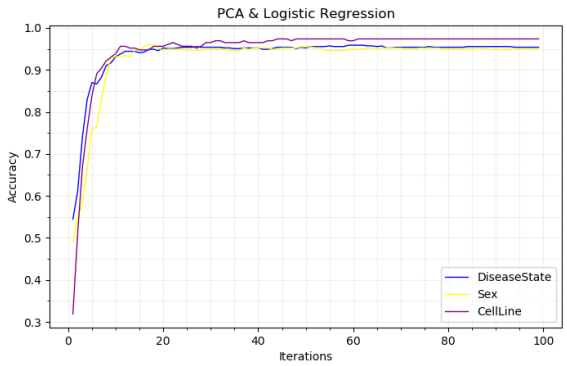

Figure 8. **The Accuracy-Iteration Curve of LR**

3.6. Support Vector Machine

Before doing SVM, we use PCA to reduct the dimension of features to 1000. As discussed in section 3.1, we use 80% of data to train the model and use the rest 20% to evaluate the performance of our model.

We use **soft margin** which allows part of the data not to follow the limitations strictly. We use linear kernel and RBF kernel to evaluate the result.

3.6.1 Kernel Selection

- **Linear kernel.** We set the penalty factor C to 0.9. We train the model for 100 iterations and the intermediate results is recorded and is shown in Figure 9. We can see that the accuracy converged after a relative small number of iterations and the accuracy of all of the three labels are over 94%.

- **RBF kernel.** We try different penalty factor C and hyper-parameter γ for RBF kernel and the observation is that RBF kernel is very sensitive to the selection of C and γ. We explore the influence of different parameter selections and list the results of our experiments in

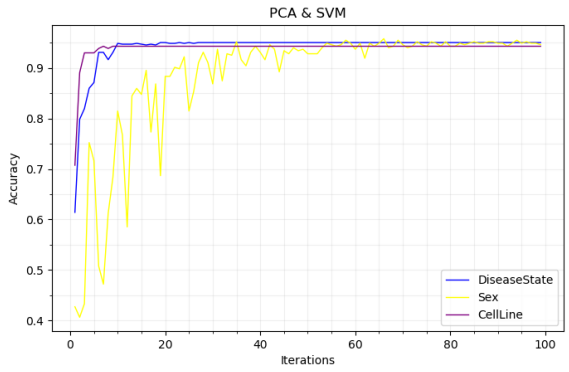

Figure 9. **The Accuracy-Iteration Curve of SVM with Linear Kernel**

Tabel 3 and we visualize it using "Accuracy-Iteration" curve as shown in Figure 10.

Table 3. **The Influence of Hyperparameters on SVM with RBF Kernel.**

No.	C	γ	Accuracy
1.	0.1	5e-5	56.33%
2.	1	5e-5	90.83%
3.	10	5e-5	**94.32%**
4.	0.1	1e-4	48.03%
5.	1	1e-4	88.21%
6.	10	1e-4	92.14%

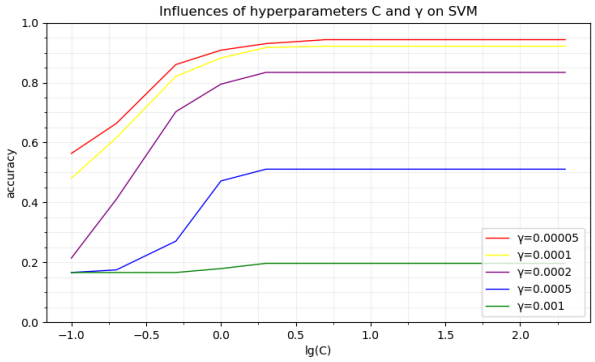

Figure 10. **The Accuracy-Iteration Curve of SVM with RBF Kernel and Different C and γ**

3.6.2 Final Results

Here we summarize all the available results of our SVM classifier and we show the best accuracy we get in Table 4 and The accuracy of all three datasets on our SVM model is over 94%.

Table 4. **SVM Best Accuracy on All of the Three Datasets.**

Datasets	Sex	DiseaseState70	CellLine
Accuracy	95.82%	95.06%	94.32%

3.7. k-Nearest Neighbors

Before doing k-NN, we use PCA to reduct the dimension of features to 1000. As discussed in section 3.1, we use 80% of data to train the model and use the rest 20% to evaluate the performance of our model.

The accuracy of k-NN for all of the three datasets are shown in Table 5

Table 5. **k-NN Accuracy on All of the Three Datasets.**

Dataset	Sex	DiseaseState70	CellLine
Accuracy	75.22%	94.05%	93.88%

Analysis of The Reason of Bad Accuracy for Dataset Sex We note that the accuracy of dataset *"Sex"* when we use k-NN is very low compared with logistic regression and SVM and we want to know the reason. At first, we thought that the reason is that dataset *"Sex"* is not linear separable in current space until we notice that SVM with linear kernel show promising result of accuracy and it does not project data to a higher-dimension space, which means dataset *"Sex"* is linear separable actually. Now we guess that the locality of k-NN algorithm leads to the bad accuracy, and maybe the feature determines dataset *"Sex"* is not strong enough as other features so that its influence to "distance" calculation in k-NN is weak, which makes the classfication of dataset *"Sex"* not effective and promising.

3.8. Deep Neural Network

Different with our previous models, we use PCA to maintain the reserved variance as 90% and reduct the dimension of features to 453. As discussed in section 3.1, we use 80% of data to train the model and use the rest 20% to evaluate the performance of our model. During training, we use the batch techique and the batch size is set to 64 for all datasets.

3.8.1 Choice of Model Complexity

Model complexity is determined by the number of hidden layers and the number of neurons of each hidden layer. Here we use a n-dimension tuple to denote the complexity of a fully connected neural network with n hidden layers. For example, our model has three hiddern layers with 256, 512, 256 neurons, and its complexity can be denoted as $(256, 512, 256)$.

We do experiments to test the performance of three neural works with different complexity (256), $(256, 512, 256)$, $(256, 512, 1024, 512, 256)$. Surprisingly, we found that all

of the three models were effective and could show promising accuracy for dataset *"DiseaseState70"*, maybe because the problem is not very difficult and it is compatible with many different models, but we still found that the performance of neural network with comlexity $(256, 512, 256)$ had the best performance. The "Loss-Epoch" curve and "Accuracy-Epoch" curve for *"DiseaseState70"* is shown in Figure 11 and Figure 12.

We can see that neural network with complexity $(256, 512, 256)$ can reach the best accuracy, which denoted by red line in Figure 12.

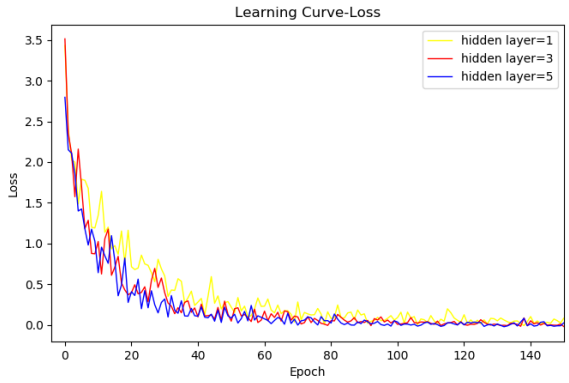

Figure 11. **The Loss-Epoch Curve of Neural Network with Different Model Comlexity of "DiseaseState70".** Batch size is 64, learning rate is 0.001, trained on Nvidia 1080-Ti.

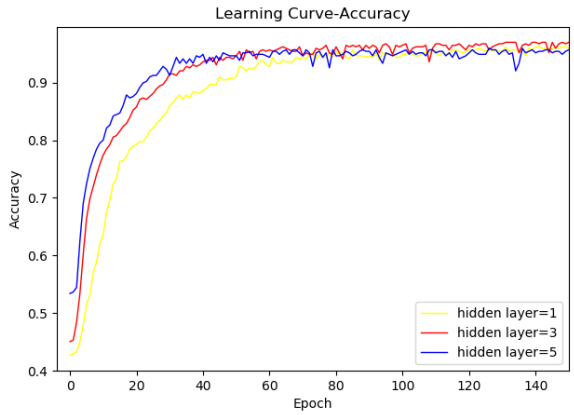

Figure 12. **The Accuracy-Epoch Curve of Neural Network with Different Model Comlexity of "DiseaseState70"**. Batch size is 64, learning rate is 0.001, trained on Nvidia 1080-Ti.

3.8.2 Choice of Learning Rate

Learning rate is an important hyper-parameter which can influence the performance of neural networks siginificantly. We test the change of loss and the change of accuracy on test dataset when changing learning rate for train on dataset *"DiseaseState70"*. The "Loss-Epoch" curve and "Accuracy-Epoch" curve for *"DiseaseState70"* with different learning rate for training is shown in Figure 13 and Figure 14.

Here is our analysis:

- **Choice of learning rate is accosiated with batch size.** For **Batch** technique, larger batch size tends to have larger learning rate. We can see the performance of choosing learing rate of 0.0001 is poor, which show that for such a big batch size of 64 while training, such a small learning rate of 0.0001 is not appropriate and will lead to bad performance. Since the batch size is large enough, such a large learning rate of 0.5 can work well.

- **Learning rate will influence the speed of convergence.** We can see that the speed of convergence is increasing while increasing learning rate from 0.0001 to 0.5. We can see that the speed of convergence for learning rate of 0.0001 is too slow and can not reach good result though the number of epochs is large enough. As we have known before, such a big learning rate of 0.5 will make convergence harder and the loss and accuracy will continue jumping, but the big batch size makes learning rate of 0.5 work well as shown in figures.

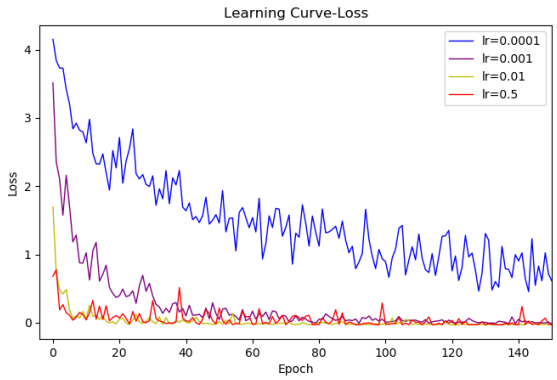

Figure 13. **The Loss-Epoch Curve of Neural Network with Different Learning Rate of "DiseaseState70".** Batch size is 64, learning rates are 0.0001, 0.001, 0.01 and 0.5, trained on Nvidia 1080-Ti.

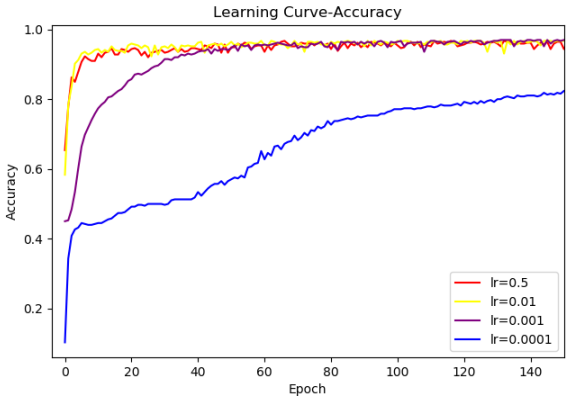

Figure 14. **The Accuracy-Epoch Curve of Neural Network with Different Model Learning Rate of "DiseaseState70"**. Batch size is 64, learning rates are 0.0001, 0.001, 0.01 and 0.5, trained on Nvidia 1080-Ti.

3.8.3 Choice of Regularization

Besides commonly used L2 norm, we explore the performance of the state-of-art regularization technique **Dropout** [6] which has already shown to be very powerful in extremely deep neural nerworks such as AlexNet [9] and ResNet [10]. We find that for such a powerful regularization method, it performs poorly on shallow and simple neural network. **For a simple network like the neural network with 3 hidden layers, randomly drop neurons with rate of 0.5 will cause serious underfitting.** The "Accuracy-Epoch" curve for *"DiseaseState70"* with dropout or not is shown in Figure 15.

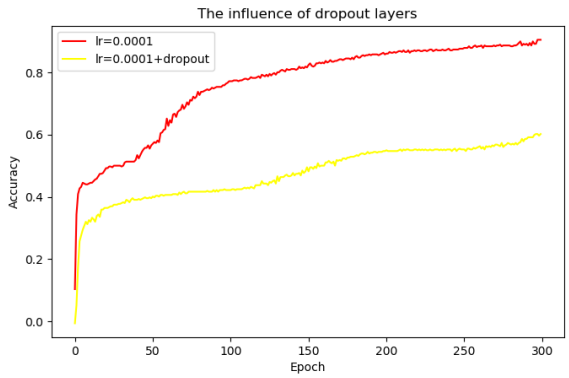

Figure 15. **The Accuracy-Epoch Curve of Neural Network with Dropout or Not of "DiseaseState70"**. Batch size is 64, learning rates are 0.0001, dropout is used or not , trained on Nvidia 1080-Ti.

3.8.4 Choice of Batch Norm

Batch normalization [5] is a technique for improving the performance and stability of artificial neural networks. It is a technique to provide any layer in a neural network with inputs that are zero mean/unit variance. We can see that batch the optimization process more stable and smooth. Train with batch norm can achieve higher accuracy as shown in Figure 17 where red line denotes the accuracy of train with batch norm. The "Loss-Epoch" curve and "Accuracy-Epoch" curve for *"DiseaseState70"* with batch norm or not for training is shown in Figure 16 and Figure 17.

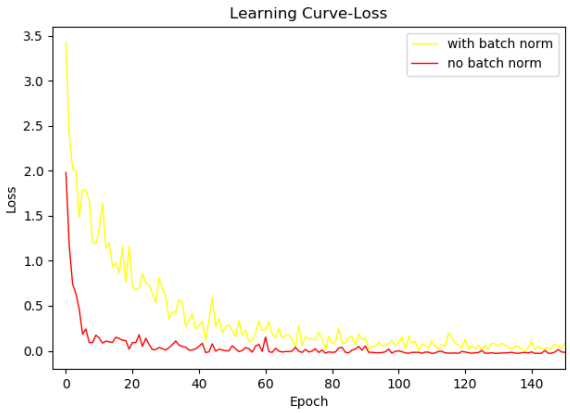

Figure 16. **The Loss-Epoch Curve of Neural Network with batch norm or not of "DiseaseState70"**. Batch size is 64, learning rates are 0.001, batch norm is used or not , trained on Nvidia 1080-Ti.

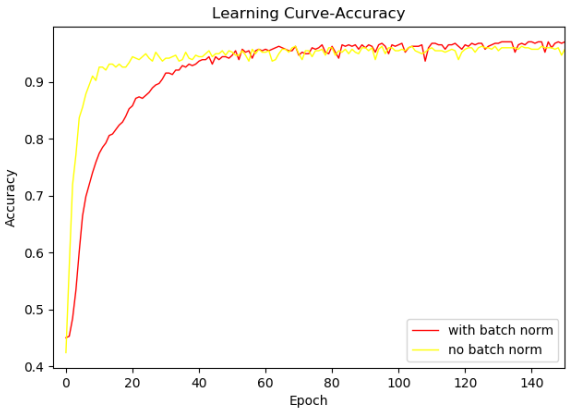

Figure 17. **The Accuracy-Epoch Curve of Neural Network with batch norm or not of "DiseaseState70"**. Batch size is 64, learning rates are 0.001, batch norm is used or not , trained on Nvidia 1080-Ti.

3.8.5 Final Results

After exploring some factors or hyper-parameters that will influence the performance of our deep learning model, our model finally selects a fully connected neural network with model complexity of $(256, 512, 256)$, we trained this neural network with batch normalization. As for the selection of hyper-paramters, we set learing rate as 0.001, batch size as 64 and we choose the best accuracy of test dataset among those epochs.

Final results on datasets *"Sex"*, *"CellLine"*, *"DiseaseState16"* and *"DiseaseState70"* are shown in Table 6

Table 6. **Deep Learning Accuracy on All of the Three Datasets.**

Dataset	Accuracy
Sex	98.81%
CellLine	98.67%
DiseaseState16	96.44%
DiseaseState70	97.65%

4. Discussion

4.1. Comparison Between Traditional Machine Learning Method and Deep Learning

Table 7 shows the comparison between the accuracy of traditional machine learning methods and deep learning (DL in the table) method on different datasets. We can see that the deep learning's accuracy of each dataset is better than other traditional methods.

Though deep learning is the most powerful method for this project, we can still see that traditional methods show comparable and promising performance. Additionally, traditional machine learning methods require less computional overhead than deep learning and they are more time-efficient. For sequence understanding problems, traditional machine learning methods are still effective.

Table 7. **Accuracy of Different Datasets on Different Models.**

Dataset	LR	SVM	k-NN	DL
Sex	95.02%	95.82	75.22%	**98.81%**
CellLine	97.37%	94.32%	93.88%	**98.67%**
DiseaseState70	95.88%	95.06%	94.05%	**97.65%**

4.2. Future Work

Future work includes using more reasonable evaluation metric like **F1 score**, using some new traditional machine learning methods like **deep forest** and **random forest**, using **unsupervised learning** to explore the nature of data distribution.

5. Acknowledgment

My partner of this course project is Bohong Wu. Special thanks to TAs and Prof. Bo Yuan for their guidence.

References

[1] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[3] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, sep 2006.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.