Project Report for the AI (CS410 2017 Spring)

***Abstract*—this report is a solution of the classification problem under the dataset *Gene_Chip_Data*. As a basic problem in machine learning field, there are plenty of mature methods to solve such problem. This study proposed the LR (Logistic Regression) method and the SVM (Support Vector Machine) method to train a classifier for multiclass and binary classification problem. The model also adopted the preprocessing of PCA (principal component analysis) to reduce the dimension of original data. Besides, the study tried to implement a deep learning model called DBN (Deep Belief Network) to do the same work as LR and SVM. By solving the same problem with different methods, we can have simple discussion and comparison between those methods.**

***Keywords*—Binary Classification, Multiclass Classification, Logistic Regression, Support Vector Machine, Deep Belief Network**

I. METHODS

This section reviews the usage of the mentioned methods, namely classification and preprocessing techniques.

A. Principal Component Analysis

PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate(called the first principal component), the second greatest variance on the second coordinate, and so on[1].

Consider a data matrix $\boldsymbol{X}$, where each of the n rows represents a different repetition of the experiment, and each of the p columns gives a particular kind of feature.

Mathematically, the transformation is defined by a set of p-dimensional vectors of weights or loadings $\boldsymbol{W_{(k)}} = (\boldsymbol{w_1}, \dots, \boldsymbol{w_p})_{(k)}$ that map each row vector $\boldsymbol{x_{(i)}}$ of $\boldsymbol{X}$ to a new vector of principal component scores $\boldsymbol{t_{(i)}} = (\boldsymbol{t_1}, \dots, \boldsymbol{t_m})_{(i)}$, given by

$$\boldsymbol{t_{k(i)}} = \boldsymbol{X_{(i)}} \cdot \boldsymbol{W_{(k)}} \quad \text{for} \quad \boldsymbol{i} = 1,\dots,n \quad \boldsymbol{k} = 1,\dots,m$$

In such a way that the individual variables of t considered over the data set successively inherit the maximum possible variance from x, with each loading vector w constrained to be a unit vector.

In order to maximize variance, the first loading vector $\boldsymbol{w_{(1)}}$ thus has to satisfy

$$\boldsymbol{w_{(1)}} = arg \max_{||\boldsymbol{w}||=1} \left\{ \sum_{\boldsymbol{i}} (\boldsymbol{t_1})_{(i)}^2 \right\} = arg \max_{||\boldsymbol{w}||=1} \left\{ \sum_{\boldsymbol{i}} (\boldsymbol{x_{(i)}} \cdot \boldsymbol{w})^2 \right\}$$

Equivalently, writing this in matrix form gives

$$\boldsymbol{w_{(1)}} = arg \max_{||\boldsymbol{w}||=1} \{||\boldsymbol{Xw}||^2\} = \max_{||\boldsymbol{w}||=1} \{\boldsymbol{w^T X^T X w}\}$$

Since $\boldsymbol{w_{(1)}}$ has been defined to be a unit vector, it equivalently also satisfies

$$\boldsymbol{w_{(1)}} = \arg max \left\{ \frac{\boldsymbol{w^T X^T X w}}{\boldsymbol{w^T w}} \right\}$$

A standard result for a symmetric matrix such as $\boldsymbol{X^T X}$ is that the quotient's maximum possible value is the largest eigenvalue of the matrix, which occurs when $\boldsymbol{w}$ is the corresponding eigenvector.

The *k-th* component can be found by subtracting the first *k-1* principal components from $\boldsymbol{X}$:

$$\boldsymbol{\hat{X}_k} = \boldsymbol{X} - \sum_{\boldsymbol{s}=1}^{\boldsymbol{k}-1} \boldsymbol{X w_{(s)} w_{(s)}^T}$$

And then finding the loading vector which extracts the maximum variance from this new data matrix

$$\boldsymbol{w_{(k)}} = arg \max_{||\boldsymbol{w}||=1} \{||\boldsymbol{\hat{X}_k w}||^2\} = \arg max \left\{ \frac{\boldsymbol{w^T \hat{X}_k^T \hat{X}_k w}}{\boldsymbol{w^T w}} \right\}$$

It turns out that this gives the remaining eigenvectors of $\boldsymbol{X^T X}$, with the maximum values for the quantity in brackets given by their corresponding eigenvalues. Thus the loading vectors are eigenvectors of $\boldsymbol{X^T X}$. $\boldsymbol{X^T X}$ itself can be recognized as proportional to the empirical sample covariance matrix of the dataset $\boldsymbol{X}$.

The following is a detailed description of PCA using the covariance method as opposed to the correlation method [2]. And our study also use this method. But note that it is better to use the singular value decomposition (using standard software). The goal is to transform a given data set **X** of dimension *p* to an alternative data set **Y** of smaller dimension *L*:

$$\mathbf{Y} = PCA(\mathbf{X})$$

1) Calculate the empirical mean and the deviations from the mean

Find the empirical mean along each column $j = 1, \dots, p$. Place the calculate mean values into an empirical mean vector u of dimensions $p \times 1$

$$\mathbf{u}[\boldsymbol{j}] = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{X}[i,j]$$

Mean subtraction is an integral part of the solution towards finding a principal component basis that minimizes the mean square error of approximating the data. Hence we should center the data by subtract the empirical mean vector **u** from each row of the data matrix **X** and store the result in the $n \times p$ matrix **B**

$$\mathbf{B} = \mathbf{X} - \mathbf{h}\boldsymbol{u^T}$$

Where **h** is an $n \times 1$ column vector of all 1s

2) Find the covariance matrix

Find the $p \times p$ empirical covariance matrix C from the product of matrix B with its transpose:

$$\mathbf{C} = \frac{\mathbf{1}}{\boldsymbol{n}} \boldsymbol{B} \cdot \boldsymbol{B}^{\boldsymbol{T}}$$

3) Find the eigenvectors and eigenvalues of the covariance matrix

Compute the matrix V of eigenvectors which diagonalizes the covariance matrix C:

$$\boldsymbol{V}^{-1}\boldsymbol{C}\boldsymbol{V} = \boldsymbol{D}$$

Matrix **D** will take the form of an $p \times p$ diagonal matrix, where

$$\mathbf{D}[\boldsymbol{k}, \boldsymbol{l}] = \begin{cases} \boldsymbol{\lambda_k}, & \mathbf{k} = \mathbf{l} \\ \mathbf{0}, & \mathbf{k} \neq \mathbf{l} \end{cases}$$

λ_k is the *k-th* eigenvalue of the covariance matrix **C**, and matrix **V**, also of dimension $p \times p$, contains p column vectors, each of length p, which represent the p eigenvectors of the covariance matrix **C**

4) Rearrange the eigenvectors and eigenvalues

Sort the columns of the eigenvector matrix **V** and eigenvalue matrix **D** in order of decreasing eigenvalue. Make sure to maintain the correct pairings between the columns in each matrix.

5) Compute the cumulative energy content for each eigenvector

The eigenvalues represent the distribution of the source data's energy among each of the eigenvectors, where the eigenvectors form a basis for the data. The cumulative energy content g for the j-th eigenvector is the sum of the energy content across all of the eigenvalues from 1 through j:

$$\mathbf{g[j]} = \sum_{\boldsymbol{k}=1}^{\boldsymbol{j}} \boldsymbol{D}[\boldsymbol{k}, \boldsymbol{k}]$$

6) Select a subset of the eigenvectors as basis vectors

Save the first L columns of V as the $p \times L$ matrix W. Use the vector g as a guide in choosing an appropriate value for L. The goal is to choose a value of L as small as possible while achieving a reasonably high value of g on a percentage basis. For example, you may want to choose L so that the cumulative energy g is above a certain threshold, like 90 percent. In this case, choose the smallest value of L such that

$$\frac{\boldsymbol{g}[\boldsymbol{L}]}{\boldsymbol{g}[\boldsymbol{p}]} \geq 0.9$$

7) Project the data onto the new basis

$$\mathbf{Y} = \mathbf{X} \cdot \mathbf{W}$$

B. Logistic Regression

In statistics, logistic regression is a regression model where the dependent variable is categorical—that is, where it can take only two values, 0 and 1. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor variables (features). It allows one to say that the presence of a risk factor increases the probability of a given outcome by a specific percentage. In order to fit multiclass need, logistic regression can be binomial, ordinal or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types, "0" and "1", as this leads to the most straightforward interpretation [3]. If a particular observed outcome for the dependent variable is the noteworthy possible outcome (referred to as a "success" or a "case") it is usually coded as "1" and the contrary outcome (referred to as a "failure" or a "noncase") as "0". Logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a noncase.

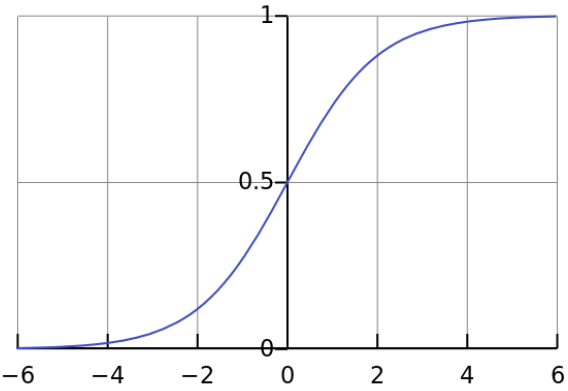

Figure 1. The standard logistic function σ(t)

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is useful because it can take any real input $\boldsymbol{t}$ ($\mathbf{t} \in \mathbf{R}$), whereas the output always takes values between 0 and 1 and hence is interpretable as a probability. The logistic function $\sigma(\boldsymbol{t})$ is defined as follows:

$$\sigma\,(\boldsymbol{t}) = \frac{1}{1 + e^{-\mathbf{t}}}$$

A graph of the logistic function of the t-interval (-6, 6) is shown in Figure 1. Let us assume that t is a linear function of a single explanatory variable x. We can then express t as follows:

$$\mathbf{t} = \boldsymbol{w_0} + \boldsymbol{w_1 x_1} + \cdots + \boldsymbol{w_n x_n} = \boldsymbol{W x}$$

And the logistic function can now be written as:

$$\mathrm{F(x)} = \frac{1}{1 + e^{-\boldsymbol{W x}}}$$

C. Support Vector Machine

Support vector machines are used for clustering data into two categories according to maximum boundary geometry. SVMs are supervised learning models associated with learning algorithms and used to analyze data and recognize patterns. In the SVM training algorithm, new examples are assigned to one category or another as either nonlinear or linear binary classifiers obtained from a set of training examples. Generally, SVMs provide a suitable means of clustering data for small data sets especially, because the classification is based on support vectors, and data dimension-size ratio has no effect of model

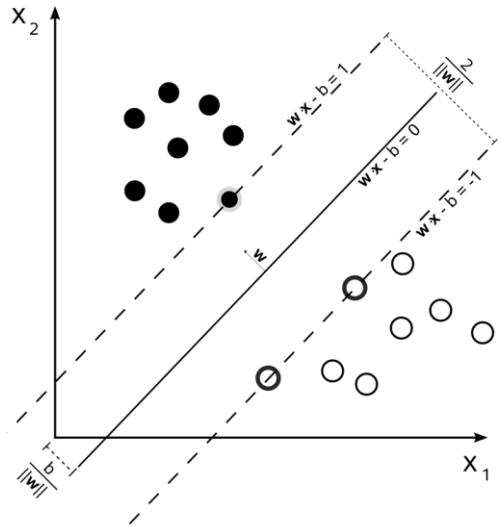

Figure 2. Support vector machine

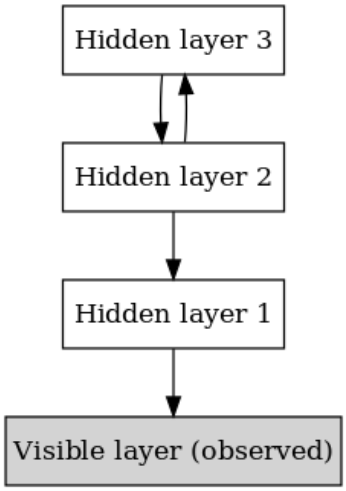

Figure 3. Schematic overview of a DBN

complexity.

A graph of the SVM is shown in figure 2. Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Given a training dataset $\boldsymbol{D(x_i, y_i)}$, where x_i denotes n observations of malware signatures, $\boldsymbol{x_i} \in \boldsymbol{R^N}$, $\boldsymbol{i} = 1,\ldots,N$; and $\boldsymbol{y_i}$ is the corresponding class label whose value is either 1 or -1(i.e., malicious or benign), indicating the class to which the point $\boldsymbol{x_i}$ belongs, $\boldsymbol{y_i} \in \{1, -1\}$, assigned to each observation $\boldsymbol{x_i}$. Each behavioural signature $\boldsymbol{x_i}$ is of dimension $\boldsymbol{d}$ corresponding to the number of propositional variables.

$$D = \left\{(\boldsymbol{x_i}, \boldsymbol{y_i}) | \boldsymbol{x_i} \in \boldsymbol{R^N}, \boldsymbol{y_i} \in \{1, -1\}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{N}}$$

A typical clustering problem is identifying the maximum margin of a hyperplane that divides the points exhibiting $\boldsymbol{y_i} = 1$ from those exhibiting $\boldsymbol{y_i} = -1$. Any hyperplane can be written as the set of points, x, satisfying the following formula:

$$\mathbf{w} \cdot \boldsymbol{x_i} + \boldsymbol{b} = 0 \qquad \forall \boldsymbol{i}$$

where · denotes the inner product and W denotes the normal vector of the hyperplane. The parameter $\frac{\boldsymbol{b}}{||\boldsymbol{w}||}$ determines the offset of a hyperplane from the origin along the normal vector W. Generally, a decision function D($\boldsymbol{x}$) is defined for clustering as D($\boldsymbol{x}$) = $\mathbf{w} \bullet \boldsymbol{x} + \mathbf{b}$.

The basic idea is to find a set of support vectors which define the widest linear margin between two classes. A traditional linear SVM has a key drawback, i.e, assuming that the training data are linearly separable, it cannot hold true when applied to practical real cases. Bernhard et al.[4] suggested a novel approach to generating nonlinear classifiers by applying a kernel function to maximum-margin hyperplanes. The nonlinear classification algorithm is formally similar to the linear SVM, except that each inner product ($\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$, i.e) is replaced by a kernel function. This enables the algorithm to fit the maximum margin hyperplane in a transformed feature space.

SVM can also be used in multiclass classification problems by modifying the dimension of output value. If the dataset has $\boldsymbol{n}$ different labels, then we need to modify the output to hold $\boldsymbol{n}$ dimensions, making the corresponding dimension value be 1 and others be -1.

D. Deep Belief Network

Deep belief network is a generative graphical model, or alternatively a type of deep neural network, composed of multiple layers of latent variables (hidden units), with connections between the layers but not between units within each layer. DBN can be viewed as a composition of simple, unsupervised networks of restricted Boltzmann machines (RBMs)， where each sub-network's hidden layer serves as the visible layer for the next. This also leads to a fast, layer-by-layer unsupervised training procedure.

A scheme of a DBN is shown in figure 3, and the arrows represent directed connections in the graphical model that the net represents. The training algorithm for DBNs proceeds as follows [5]. Let *X* be a matrix of inputs, regarded as a set of feature vectors.

1) Train a RBM on $\boldsymbol{X}$ to obtain its weight matrix $\boldsymbol{W}$. Use this as the weight matrix between the lower two layers of the network.

2) Transforms $\boldsymbol{X}$ by the RBM to produce new data $\boldsymbol{X'}$, either by sampling or by computing the mean activation of the hidden units.

3) Repeat this procedure with $\boldsymbol{X} \leftarrow \boldsymbol{X'}$ for the next pair of layers, until the top two layers of the network are reached.

4) Fine-tune all the parameters of this deep architecture with respect to a proxy for the DBN log-likelihood, or with respect to a supervised training criterion (after adding extra learning machinery to convert the learned representation into supervised predictions, e.g. the logistic regression classifier).

II. RESULTS

In this section, the implementations and the results of the models above will be represented. The experiment environment is shown in the table 1:

Table 1: Experiment environment

Operating System	Microsoft Windows 10 Professional
RAM	8GB
Graphics Device	NVIDIA GeForce GT 755M
CPU	Inter i5-4200M 2.50GHz
Platform	Python 3.5 + tensorflow 1.0.1

In the *Gene_Chip_Data* dataset, file *microarray.oroginal.txt* is the data file, which saves 22283 rows and 5894 columns matrix data. Every row shows the feature and every column shows the observation. In another word, there are 5894 examples and every example has 22283 features. Another file *E-TBM-185.sdrf.txt* saves the labels for examples from *[material]* to *[disease status]*, etc. Our experiment uses the *[material]* labels as the binary classification standard and uses the *[disease status]* labels as the multiclass classification standard. There are 2 different label, *organism_part* & *cell_line* under the *[material]* and 194 kinds of disease status labels under *[disease status]*. And in order to test the performance of multiclass classifier and binary classifier, we conducted two kind of experiment for every methods. Multiclass test requires classifiers to tell all 194 kinds of status, and binary class test only requires classifiers to tell whether the observations comes from organism or cell.

As for the testing, we use the **k-fold cross-validation** method to test our models. The original data is divided into *k* groups and we make *k-1* groups be the training set and 1 group left be the test set. Then change the training set and test set in turn for k times so that every group has the chance to be tested. Finally we will calculate the average accuracy of k times cross-validations.

A. PCA & LR

Since the number of features in the dataset is more than 20000, we should use PCA first to reduce the dimension. We set the contribution ratio of PCA as 0.9 and we finally get 112 main features. Then we could use the logistic regression model to fit the reduced data. Besides, it's important to transfer the label to vectors. We used one-hot code to represent different kind of labels. For example, we could use [1, 0] to represent *organism_part* and use [0, 1] to represent *cell_line*. Figure 4 is the result of multiclass classification and binary classification. We trained the model for 100 iterations and every iteration has 100 batches.

B. PCA & SVM

In this experiment, we used soft margin SVM with C = 0.9 rather than hard margin SVM. Same as PCA & LR, we first used PCA to reduce the dimension and then used SVM model to fit the data. However, instead of using original one-hot code to represent labels, we use the methods that similar to one-hot code, which uses -1 to replace 0 in one-hot code. For example, we use [1, -1] to represent organism_part and use [-1, 1] to represent cell_line. Figure 5 is the result of multiclass and binary classification. We trained the model for 100 iterations and every iteration has 100 batches.

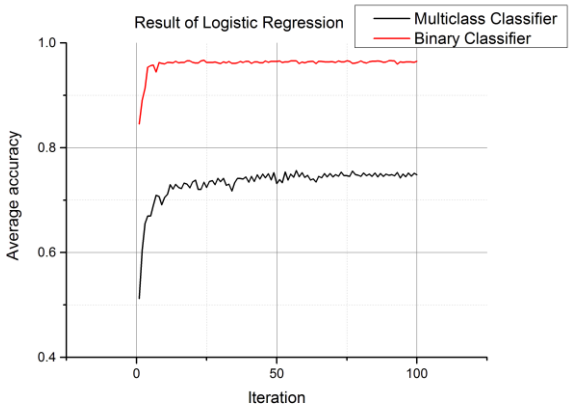

Figure 4. Result of LR

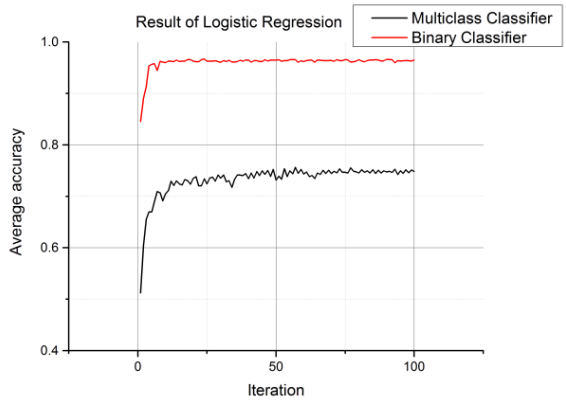

Figure 5. Result of SVM

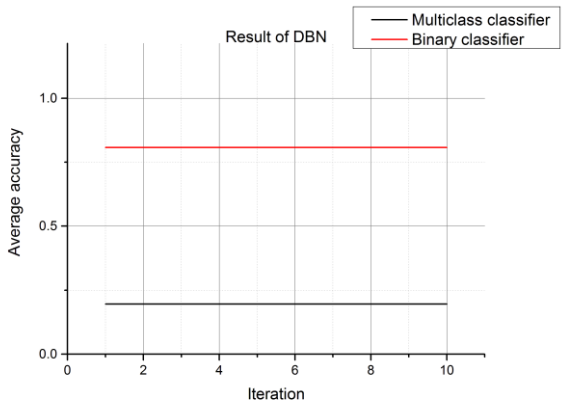

Figure 6. Result of DBN

C. DBN

When we focus on Figure 6, we can find that DBN model has so perfect convergence that the accuracy of every epoch is almost the same. We think the reason is pre-training of deep learning. However, because the number of observations under every label is much too insufficient that it can't satisfy the standard of deep learning, it performs poor in classification problems. The accuracy of multiclass classification problem is only about 0.2 and the accuracy of binary classification

problem is about 0.8, which is far away from LR and SVM. It may do well if there are enough data.

III. Discussions

In this section, we decide to discuss the results shown in last section and make some comparisons between different classifiers. At first we want to claim the reason why we used the *[material]* label as the standard of binary classification problem rather than continuously using *[disease status]*. In the dataset, there are 4752 observations with *organism_part* label and 1142 observations with *cell_line* label. The ratio is about 4:1. We would have decided to use healthy and unhealthy as the standard. However, there are only 85 healthy observations and 4917 of all observations are unhealthy, where the ratio is about 58:1. The incommensurable ratio will cause big error in statistics. So we changed the standard.

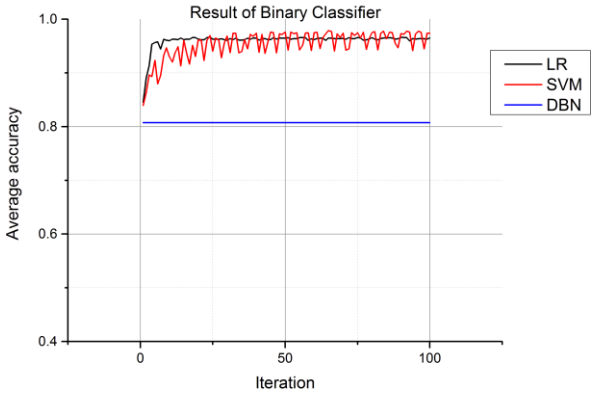

Figure 7. Result of Binary Classifier

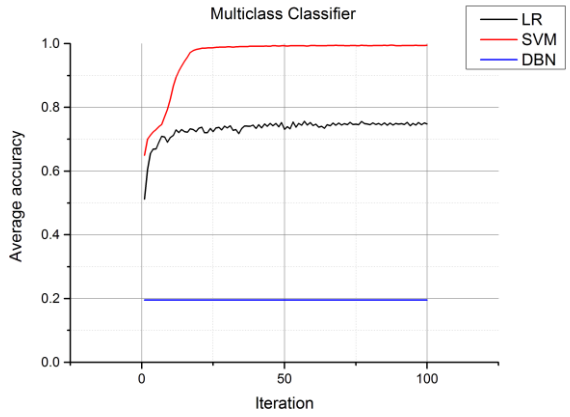

Figure 8. Result of Multiclass Classifier

A. LR

When we focus on Figure 4, we can find that LR model did well in binary classification problems and the accuracy reached 0.96. It also had an excellent performance in convergence. However, it didn't have a high accuracy in multiclass classification problems. And it had little fluctuation in the graph.

B. SVM

When we focus on Figure 5, it's obvious that VM models perfectly fitted multiclass classification problem that its convergence is excellent and can reach 0.99. However, in the aspect of binary classification problem, its bad convergence is also obvious. It fluctuates drastically around the accuracy of 0.95. But it's undeniable that SVM models have higher accuracy than LR.

C. DBN

When we focus on Figure 6, we can find that DBN model has so perfect convergence that the accuracy of every epoch is almost the same. We think the reason is pre-training of deep learning. However, because the number of observations under every label is much too insufficient that it can't satisfy the standard of deep learning, it performs poor in classification problems. The accuracy of multiclass classification problem is only about 0.2 and the accuracy of binary classification problem is about 0.8, which is far away from LR and SVM. It may do well if there are enough data.

D. Comparison

According to the comparison in figure 7 and figure 8, we can draw the conclusion that LR performs better when there are fewer target labels while SVM performs better when there are more target labels, and DBN has absolute advantages in convergence while it doesn't have high accuracy because of the restriction of data quantities.

References

[1] Jolliffe I.T. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4

[2] "Engineering Statistics Handbook Section 6.5.5.2". Retrieved 19 January 2015.

[3] Hosmer, David W.; Lemeshow, Stanley (2000). Applied Logistic Regression (2nd ed.). Wiley. ISBN 0-471-35632-8.

[4] E.B. Bernhard, M.G. Isabelle, V. Vapnik, and N. Vladimir (1992). A training algorithm for optimal margin classifiers. In Haussler, David (editor); 5th Annual ACM Workshop on COLT, pages 144–152, Pittsburgh, PA, ACM Press.

[5] Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets" (PDF). Neural Computation. 18 (7): 1527–1554. doi:10.1162/neco.2006.18.7.1527. PMID 16764513.