CS410 Artificial Intelligence
Project Report

Zhengtian Xu
5140309178
Shanghai Jiao Tong University
xztlook@sjtu.edu.cn

Qingrong Chen
5140309203
Shanghai Jiao Tong University
chenqingrong@sjtu.edu.cn

Abstract

Artificial intelligence has become ubiquitous in our society, especially in computer vision, natural language processing and intelligent speech, with applications in search, image classification and understanding, robot control, medicine and self-driving cars. Typical machine learning methods has shown great performance on some relative tasks. Recent developments in neural network (aka "deep learning") approaches have greatly advanced the performance of these state-of-the-art AI related systems. In this paper, we conduct a thorough examination of traditional, classical approach and deep learning method with regard to a basic but common task. Our primary aim is to understand the core difference and their application scope. We use Human Gene Expression Dataset and approach this by doing several careful hand-analysis experiments using different methods. Extensive experiments carried on this dataset to demonstrate more specialties in classical methods and deep learning based methods.[1]

1. Introduction

Human always has only one human genome sequence, different genes are expressed in many different cell types and tissues, as well as in different developmental stages or diseases [7]. Thus, human gene expression can be used to determine the diseases. Once the hidden semantic meaning behind the gene expression is fully exploited, this will promise to enable a wide array of medical applications such as disease diagnosis and medical AI robot.

Some traditional, classical classification models have demonstrated their ability and robust in supervised learning. Especially in classification tasks which aim to classify examples into given set of categories, classical machine learning like SVM, logistic regression, nearest neighbor can interpret a specific task into statistical classification. All of these methods has their application scopes and can achieve great performance.

Recently, deep neural network, also called deep learning, has achieved great performance on applications like image classification tasks and natural language processing. Neural networks are inspired by our understanding of the biology of our brains all those interconnections between the neurons. But, unlike a biological brain where any neuron can connect to any other neuron within a certain physical distance, these artificial neural networks have discrete layers, connections, and directions of data propagation.

To address this problem, in this work, we show that both classical machine learning methods and deep learning models can be applied into multiple class classification tasks and achieve great accuracy. However, in different experiment setting, these methods will reflect slight difference, which will be discussed in Section. Since the given dataset is a large p small n, where p and n respectively denotes the number of features and samples. We use PCA as our data preprocessing before training. The system is trained on feature vectors annotated with class label of samples from our processed data. We will compare different methods and their performance in a fairly manner.

2. Approach

We formulate the problems as follow: Given a set of samples annotated with class label $Y_{\{1,2,...,n\}} = \{y_1, y_2, ..., y_n\}$, we expect to find exact category of the given data. Each sample has its own feature vector, these feature vectors can interpret the characteristic of each sample. Since the given dataset is not very suitable for training, we employ some data preprocessing methods on them.

2.1. Data Preprocessing

Since the given dataset is a large p small n problem, we employ dimensionality reduction technique PCA [10] to perform a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the

[1] Our code is available at https://github.com/ZhengtianXu/Gene_Chip

low-dimensional representation is maximized. Usually, the number of dimension after PCA is smaller than n. We suppose the data matrix X be of $n \times p$ size, where n is the number of samples and p is the number of features. Some important steps are listed as follows

- Normalize the data and let it be centered.
- Calculate the covariance matrix $C = \frac{X^T X}{n-1}$. It is a symmetric matrix and so it can be diagonalized: $C = VLV^T$, where V is a matrix of eigenvectors and L is a diagonal matrix with eigenvalues λ_i in the decreasing order on the diagonal.
- We choose d rows and the sum of whose variance reach 90% of original variance.
- After data preprocessing, the dimension of feature vectors reduce to 453 and the original dimension is 22283.

2.2. Classical Method

In the experiment setting using classical methods, we use data after data preprocessing, which means the size of data is 5896 × 453

K Nearest Neighbor Classifier [3] K-nearest neighbor classifier is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (*e.g.*, distance functions). Suppose now that we are given all of the training dataset annotated with labels, and we wish to label testing dataset. The nearest neighbor classifier will take a test sample, compare it to every single one of the training samples, and predict the label of the closest training sample. In this method, we will discuss the performance using different distance function. Suppose we are given two data samples and we represent them as feature vectors f_1 and f_2.

A reasonable choice for comparing them is L1 distance, which is defined as follows:

$$d_1(f_1, f_2) = \sum_p |f_1^p - f_2^p| \tag{1}$$

where p denotes the dimension of feature vector and the sum is taken over all dimensions.

Another common choice could be to instead use the L2 distance, which has the geometric interpretation of computing the euclidean distance between two vectors. The distance takes the form:

$$d_2(f_1, f_2) = \sqrt{\sum_p |f_1^p - f_2^p|^2} \tag{2}$$

Later we will consider the differences between the two metrics. In particular, the L2 distance is much more unforgiving than the L1 distance when it comes to differences between two vectors. That is, the L2 distance prefers many medium disagreements to one big one.

Linear Classifier Since the data given is a large p small n problem, we start out with arguably the simplest possible function, a linear mapping:

$$f(x_i, W, b) = Wx_i + b \tag{3}$$

In Eq 3, we let x_i flatten out to a single column vector of shape $[D \times 1]$. The size of matrix W is $[K \times D]$, and the size of vector b is $[K \times 1]$. Both W and b are the parameters of the function.

In linear classifier, we employ two different loss function to compare the differences between them.

- **Multiclass Support Vector Machine [9]**

We first develop the multiclass SVM loss, which can be also called Hinge Loss. The intuition of Hinge Loss is that it wants the correct class for each sample to a have a score higher than the incorrect classes by some fixed margin Δ. The loss function is defined as follows

$$L_i = \sum_{j \neq y_i} \max\{0, (w_j^T x_i - w_i^T x_i + \Delta)\} \tag{4}$$

where w_j is the j-th row of W reshaped as a column. However, we extend the loss function with a regularization penalty $R(W)$ to avoid the set of W is not necessarily unique. We use the most common regularization penalty L2 norm, which can discourage large weights through the square sum of all the elements in matrix

$$R(W) = \sum_i \sum_j W_{ij}^2 \tag{5}$$

Thus, the Hinge Loss is made up of Eq 4 and Eq 5

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) \tag{6}$$

where W is the normal vector to the hyperplane. The parameter λ balances the trade-off between increasing the margin-size and ensuring that the x_i lie on the correct side of the margin.

- **Softmax Classifier**

We then employ softmax loss. We stay the linear interpreting unchanged and replace the Hinge loss with a cross-entropy loss after doing the unnormalized log probabilities for each class. The loss function is defined as follows

$$L_i = -f_{y_i} + \log \sum_j e^{f_j} \tag{7}$$

where the notation f_j means the j-th element of the vector of class scores f calculated in Eq 3. For regularization and the overall loss function, it is similar to the Eq 5 and Eq 6.

2.3. Deep Learning [6] based Method

In the experiment setting using deep learning methods, we use data after data preprocessing, which means the size of data is 5896×453

Deep Neural Network Based on the experiment setting of using deep learning, we use neural network to train a classifier. Since the features do not have any locality or direct relationship, we don't use convolutions. We employ a method based on encoding-coding with a dully-connected multilayer structure. We use L2 normalization in the input layer to ensure the data smoothing, and use L1 to control the model complexity.

- **Weight Initialization**

For weight initialization, we initialize each neurons weight vector as: $w = \frac{randn(n)}{\sqrt{n}}$, where n is the number of its inputs and $randn$ means that we sample from a zero mean, unit standard deviation gaussian. With this formulation, every neurons weight vector is initialized as a random vector sampled from a multi-dimensional gaussian, so the neurons point in random direction in the input space. The reason why we use $\sqrt{n}$ is that this ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

For bias initialization, we simply use 0 bias initialization.

- **Regularization**

We use L2 regularization in each fullt-connected layer. It can be implemented by penalizing the squared magnitude of all parameters directly in the objective. For every weight w in the network, we add the term $\frac{1}{2}\lambda w^2$ to the objective, where λ is the regularization strength. This can help control the capacity of neural networks to prevent overfitting

Furthermore, we employ Dropout[8] to prevent overfitting. While training, dropout is implemented by only keeping a neuron active with some probability p, or setting it to zero otherwise.

- **Loss Function**

Since this is a multiple class classification task, we employ cross-entropy loss after doing unnormalized log probabilities for each class. You can refer Eq 7 for more information about this.

- **Parameter Updates**

We employ RMSprop[4] as our updating strategy. It is a very effective adaptive learning rate method and adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate.

- **Batch Normalization [5]**

Batch normalization potentially helps in two ways: faster learning and higher overall accuracy. We can use a higher learning rate, potentially providing another boost in speed.

Normalization (shifting inputs to zero-mean and unit variance) is often used as a pre-processing step to make the data comparable across features. As the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again. By normalizing the data in each mini-batch, this problem is largely avoided. We will prove it validity in our ablation experiment and more in discussion.

3. Experiment

In the section of experiment, we mainly describe the experiment setting including some parameter settings and experimental process. Since we have 5896 samples, and 1875 of them have no label. Thus we choose the other 4021 samples. Then we can find that the other dataset has 179 class. Since each class has imbalance number of samples and the data distribution is not very satisfied. We select classes whose data are more than ten times and split them to two set: training dataset and testing dataset. The training class is set to 79 because 79 classes have more than ten samples. The ratio is about $8 : 2$. To solve the imbalance data distribution, the ratio is calculated by each class instead of the whole data. It means for each class, the ratio between training data and testing data is $8 : 2$. Thus the number of training data is 2470 and the number of testing data is 1059. Furthermore, for 1875 samples without labels, we will use our best model to predict the class label for them. Since we ignore classes whose samples are less than ten times, we should have the probability that the samples without labels will not have prediction, this probability we set is 0.1. That means we don't have any prediction with the probability of 0.1.

3.1. Data Preprocessing

We split all effective data into training data and testing data. The ratio is about $8 : 2$. We ignore class with samples less than 10 times. Thus, this is a 79 class classification task. We employ PCA as the technique of dimensionality reduction. The feature dimension we reduce still remain 90% variance of the original.

3.2. Classical Method

In the KNN experiment, we compare the performance using different distance function. In the linear classifier, the SVM and softmax classifier need learning rate and regularization strength. We change these two parameters to choose the best performance we have ever achieved. This will be discussed in Section . To solve the imbalance data distribution, we oversample the minority class and undersample the majority class.

3.3. Deep Learning based Method

We use Tensorflow [1] to implement deep learning model. The learning rate is set to 0.001. The weight decay is set to 1×10^{-6} and the batch size is set to 200. We use four hidden layers and the neural units for these layers are $256, 512, 512, 256$. Then a classification fully-connected layer containing 79 neural units is added after hidden layers. Finally, a cross-entropy loss is employed. For weight initialization and regularization of each layer, you can refer to Section 2.3 for more information. To solve the imbalance data distribution, we oversample the minority class and undersample the majority class.

3.4. Oversampling and Undersampling

This approach to solve imbalanced data distribution change some processing steps in training process. It simply involve adjusting the example sets until they are balanced. Oversampling randomly replicates minority instances to increase their population. Undersampling randomly downsamples the majority class. We made a slight change in the original undersampling. The original undersampling throws away data but we think this lose information about this class. We just randomly select samples from majority class and enable the classifier train fair samples from each class in one epoch.

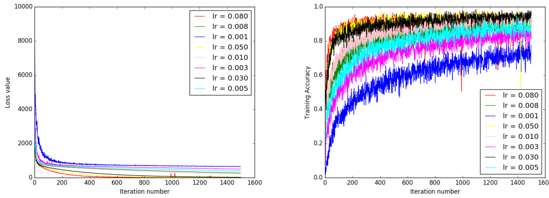
3.5. Evaluation Metric

Since the data distribution is imbalance, it is not a good choice to use scalar accuracy evaluation metric. Because scalar accuracy metric will not reflect prediction performance of smaller classes and it is shadowed by performance of any much bigger class. For example, if we have 950 positive samples and only 50 negative samples. When the classifier all predict positive label, the accuracy achieve 0.95% but actually this classifier is definitely bad. Actually, the decision made by the classifier can be represented as a 2×2 confusion matrix. The matrix has four categories: True positives (TP); False positives (FP); True negatives (TN); False negatives (FN) . Using these categories we can derive two performance metrics: the precision $P = \frac{TP}{TP+FP}$ and the recall $R = \frac{TP}{TP+FN}$ values of the classifier. Precision is the fraction of recognized instances that are relevant, while recall is the fraction of relevant instances that are retrieved. We use F_1-score, which can be interpreted as a weighted average of the precision and recall values:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{8}$$

K Nearest Neighbor		
	L1 distance	L2 distance
F_1-score	0.864	0.839

Table 1: Performance with KNN



(a) Loss Value while Training (b) Accuracy Value while Training

Figure 1: SVM Training Parameters Update

4. Result

This section we will discuss and analyze the result of our classification experiment on the given dataset.

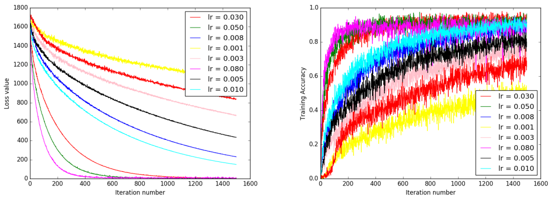
4.1. K Nearest Neighbor

In this experiment, we change the distance function to compare the performance of L1 distance function and L2 distance function. The performance we achieve is shown in Table 1.

Through the table, we can find that KNN using L1 distance is slightly better than that using L2 distance. The final performance seems better on the whole, which may be the dataset given is linear separable. Obviously, KNN is not suitable for non-linear separable data and KNN is slow when we have a lot of observations, since it does not generalize over data in advance, it scans historical database each time a prediction is needed.

4.2. Linear Classifier with SVM

In the experiment, we try different learning rate and regularization strength and compare which combination can achieve best performance on the given dataset. In the experiment, we find that when the regularization strength is 0.06, we can get best result. Thus, we show the loss value and training accuracy with different learning rate in Figure 1a and Figure 1b. In this experiment, we set the iterative number is 1500. Finally, we can get the best F_1 score on the testing dataset is 0.869 and the F_1 score on the training dataset is 0.943. The learning rate is 0.05 and the average time to train is 4.67 seconds.



(a) Loss Value while Training (b) Accuracy Value while Training

Figure 2: Softmax Training Parameters Update

Linear Classifier		
	SVM	Softmax
F_1-score	0.869	0.888
Time	4.67	13.42

Table 2: Performance of SVM and Softmax

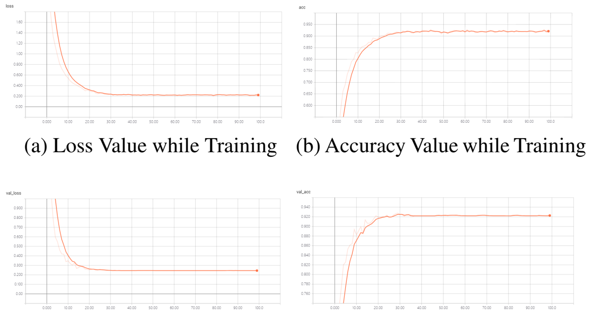
4.3. Linear Classifier with Softmax

In the experiment, we still try different learning rate and regularization strength and compare which combination can achieve best performance on the given dataset. In the experiment, we find that when the regularization strength is 0.07, we can get best result. Thus, we show the loss value and training accuracy with different learning rate in Figure 2a and Figure 2b. In this experiment, we set the iterative number is 1500 to make a fair comparison with SVM. Finally, we can get the best F_1 score on the testing dataset is 0.888 and the F_1 score on the training dataset is 0.928. The learning rate is 0.05 and the regularization strength is 0.07 and the average time to train is 13.42 seconds.

Thus, we can make a comparison in Table 2. Softmax has a better performance than SVM while SVM can provide stable results and train faster than softmax.

4.4. Deep Learning based Method

In the experiment using deep neural network, we use Tensorflow with Keras[2] to implement it and use Tensorboard to visualize the training process and testing process. We use four hidden layer and each layer with L2 regularization to avoid overfitting and make data smoothness. Dropout parameter is set to 0.5. Especially, we use batch normalization in each layer and it greatly improves the performance. We set the batch size to 200 and the epoch to train is 100. The learning rate is set to 2.5×10^{-3}. Finally, the best F_1-score achieve 0.931 at training dataset and 0.924 at testing dataset. The training, testing loss and accuracy are shown in Figure 3.



(a) Loss Value while Training (b) Accuracy Value while Training

(c) Loss Value while Testing (d) Accuracy Value while Testing

Figure 3: Network Training and Testing Parameters Update

5. Discussion

SVM vs. Softmax The SVM classifier just try to interpret correct class scores and change the weight for margins but the softmax classifier can interpret the probabilities for all classes. An SVM with margin $\Delta = 1$ just make the correct class has a score higher than the margin compared to the other classes, and it does not care about individual probability. However, the probabilities softmax classifier give is mainly depend on the regularization λ because it determines weights. These two classifier is comparable and I find SVM can provide stable results and trains faster while softmax might be slower because it may be bogged down by the calculations if we have a complex training data with many labels.

Linear classifier vs. Kernel classifier From KNN experiment, we know that the given dataset is linear separable. Thus, we mainly focus on linear classifier to make the model simple and low complexity instead using kernel based classifier.

L1 distance vs. L2 distance In KNN experiment, we conduct two sub-experiment to compare the differences between L1 distance and L2 distance. The result is that the performance resulted by L1 distance is slightly better than L2 distance. From the principle of L1 norm and L2 norm, the L1 norm can be said to be less sensitive to outliers and more sensitive to small scale behavior than the L2 norm. For training class with less samples, L1 distance consider them as outliers and be less sensitive to them. That's why L1 distance can perform slightly better than L2 distance in KNN classifier.

Batch Normalization vs. no Batch Normalization We compare two situation with batch normalization or not. We find that with batch normalization, the F_1-score can achieve 0.924. But without batch normalization, the F_1-score can only achieve 0.857. That proves the validity of batch normalization and with batch normalization, the new weights initialized at the consecutive layers accelerates the training process by ensuring a high learning rate while keep the network learns. It normalize and whiten the inputs to each layer: Zero means, unit variances and or not decorrelated, resulting in the flexibility on mean and variance value for every dimension in every hidden layer provides better learning, hence accuracy of the network.

Classical method vs. Deep learning Actually, I think our neural network based method is still called classical method. Exactly, we call it BP neural network or multilayer perceptrons. Deep learning really started with professor Hinton's convolutional neural networks, and has sparked a revival in other variants of neural network architectures such as recurrent neural network. For normal data feature, it is not suitable to use CNN or RNN. That's why we just use 4 hidden fully-connected layers. The given dataset cannot use very deep neural network because it is obviously linear separable. Deep learning can learn features hidden in training samples automatically according to the supervised learning and it tries to learn high level features from the given data. The given dataset is not like images or languages which have high level semantic features. We cannot employ classical method such as SVM or softmax classifier to solve image relative tasks because the feature classical methods extract involves a lot of handcrafted feature extraction while deep learning do it by itself with the direction of supervision. Feature extraction in classical methods requires people to tell the computer what kinds of things it should be looking for that will be formative in making a decision, which can be a time-consuming process. This also results in classical methods having decreased accuracy due to the element of human error during the programming process.

Furthermore, deep learning almost solve the selectivity and invariance problem, because it ignore some features that seems irrelevant to the target. Some noise is filtered by pooling layer and it captures the most significant features. Through iteratively updating the weight for each neural unit, it can reach the optimal status. In addition, it almost solve the speciality and generalizablity problem because it learns high level semantic features instead low-level features from data. It learns semantic meaning so it can capture the most essential features about data or images. Thus as long as the essence of data remain unchanged, deep learning model can still recognize them and exactly label them.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 4

[2] F. Chollet et al. Keras, 2015. 5

[3] P. Cunningham and S. J. Delany. k-nearest neighbour classifiers. *Multiple Classifier Systems*, 34:1–17, 2007. 2

[4] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini–batch gradient descent. 2012. 3

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3

[6] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015. 3

[7] M. Lukk, M. Kapushesky, J. Nikkilä, H. Parkinson, A. Goncalves, W. Huber, E. Ukkonen, and A. Brazma. A global map of human gene expression. *Nature biotechnology*, 28(4):322–324, 2010. 1

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 3

[9] J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999. 2

[10] M. E. Wall, A. Rechtsteiner, and L. M. Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003. 1