Digit Classification and Model Analysis

Project One of CS385 Machine Learning, 2021 Spring, SJTU

Aofan Jiang
518030910275

I. Introduction

In this project, I implement digit classification using different methods. These methods include (kernel) logistic regression with loss, SVM, LDA and neural networks. Besides, I use GAN to generate images successfully.

II. Dataset

We use SVHN dataset for digit classification. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits). And SVHN is obtained from house numbers in Google Street View images.

A. Dataset Details

In this experiment, I use the cropped version digit images with fixed shape [32,32,3]. The original dataset includes 73257 pictures for training and 26032 pictures for testing. The size of training and testing depends on classification methods. The size for each methods will be provided in the following sections.

B. Dataset Pre-process

SVHN is a little different from MNIST dataset. It's RGB color picture instead of gray type. Besides, the digit picture in SVHN includes environment noise due to crop. Therefore, if we just reshape the three-dimension picture to one-dimension vector, much information will be lost and the classification performance is extremely poor.

We first extract the histograms of oriented gradients (HOG) feature from images. Then we train and test based on such HOG feature (except neural network). This is taught on class and an example[1] in lecture note.

Since this project focus on classification instead of data pre-process, I use opencv to extract HOG feature and save the processed data in numpy file for convenience. After the process, the training data changes from (32, 32, 3, 73257) to (1764, 73257), same for testing data.

III. Logistic Regression with Different Losses

In my program, I use gradient descent to optimize the logistic regression. The original equation of the problem is

$$Y^* = \beta X + \epsilon$$

where X is in shape of (dimension, samples), β is in shape of (1, dimension) and initialized all value with one.

We use ten different β to represent ten different categories. In train stage, we train these ten models (β) in sequence. For one specific model, we regard all training data as binary label. If the original category equals to the category represented by the model, this data will be set as label one. Else the data will be set as label zero. This process can be written as
For i in category from 0 to 9:

$$\beta_i = \beta_i - \gamma * (Sigmoid(\beta_i X) - H_i)X$$

Where for $h_j \in H_i$, $h_j = 1$ if X_j's label is i; γ is learning rate

This is the training process, ten different β are the learnt model. In testing process, we choose the index with maximum predicted value from ten model as the predicted category. Finally we compare the predicted category with ground-truth and count the correct prediction to calculate accuracy. This can be written as

$$\hat{Y} = argmax_i \beta_i X \ , \qquad Acc = \frac{|\hat{Y} = Y|}{|Y|}$$

A. Ridge Loss

For Ridge loss, it adds $\lambda||\beta||^2$ as a penalty or regularization term. Therefore, when using gradient descent, we need to consider another additional term $2\lambda\beta$. The updating process of model becomes
For i in category from 0 to 9:

$$\beta_i = \beta_i - \gamma * (Sigmoid(\beta_i X) - H_i)X + \lambda'\beta$$

Where λ' is a positive tuning parameter.

B. LASSO Loss

For Lasso loss, it adds $\lambda||\beta||_{l1}$ as a penalty or regularization term. Therefore, for the gradient descent method, we need to consider this additional term. If $\beta > 0$, the gradient is 1. Or the gradient will be -1. We use $sign(\beta)$ to represent this operation for short. The updating process of model becomes
For i in category from 0 to 9:

$$\beta_i = \beta_i - \gamma * (Sigmoid(\beta_i X) - H_i)X + \lambda * sign(\beta_i)$$

Where λ is a positive tuning parameter.

C. Performance and Accuracy

In the logistic-related project, I use 10,000 pictures as training data and 1,000 pictures as testing data. I set the learning rate of gradient descent as 0.001. The lambda for Ridge and Lasso loss is set as 0.01. After 100 epochs, the result is shown in the table below.

Method	Test Accurate (%)
Logistic	78
Logistic+Ridge	75.6
Logistic+Lasso	60.8

TABLE I
Logistic Regression with Different Loss

IV. Kernel-based logistic regression + LASSO loss

Kernel-based logistic is similar to original logistic regression. We will use training data again in test stage as a kernel.

In training stage, we use training data itself to form a kernel. The shape of input data changes from (samples, dimension) to kernel (samples, samples). After that, we use the same gradient descent with Lasso loss to optimize the model for different categories, which is in shape of (categories, samples)

In testing stage, we apply the kernel on testing data and training data. Therefore, the final kernel input is in shape of (test-samples, train-samples).

A. Performance and Accuracy

In this project, I implement three kind of kernels, rbf kernel, cosine kernel and polynomial kernel. Due to memory limitation, I use 3,000 pictures for training and 500 pictures for testing. The learning rate and lambda is the same as logistic regression above. After 300 epochs, the result is shown below.

Kernel	Test Accurate (%)
RBF($\sigma = 1$)	67.6
Poly(d=2)	46.4
Cosine	38.6

TABLE II
Kernel-based Logistic Regression with Different Kernel

V. SVM

At this part, I use tool packages ***sklearn*** SVM library to implement SVMs. I use linear and non-linear with various kernels for a better comparison. I check the document of sklearn SVM, the implementation of different kernels are implemented as follows:

A. Linear SVM

Given training vectors X, in two classes $Y \in \{-1, +1\}$, the goal is to find w and b such that the prediction given by $sign(w^T\phi(x) + b)$ is correct for most samples. The primal problem can be formulated as

$$min_{w,b}\frac{1}{2}w^T w + C\sum_{i=1}^{n} max(0, y_i(w^T\phi(x_i) + b))$$

where they make use of the hinge loss. This is the form that is directly optimized by LinearSVC function.

B. Kernel SVM

SVC library optimizes the dual problem to the primal.

$$\begin{aligned} \max_{\alpha} \ & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ s.t. \quad & y^T\alpha = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, 2, ..., n \end{aligned}$$

where e is the vector of all ones, and Q is an n by n positive semidefinite matrix, $Q_{ij} = y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. The terms α_i are called the dual coefficients, and they are upper-bounded by C. This dual representation highlights the fact that training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

Once the optimization problem is solved, the output of decision function for a given sample x becomes:

$$\sum_{i \in SV} y_i\alpha_i K(x_i, x) + b$$

I try four different SVM kernels as mentioned on class

- Linear: $< x, x' >$
- Polynomial: $(\gamma < x, x' > +r)^d$, where d is specified by parameter degree, r by coef0.
- RBF: $exp(-\gamma||x - x'||^2)$, where γ is specified by parameter gamma, must be greater than 0.
- Sigmoid: $tanh(\gamma < x, x' > +r)$, where r is specified by coef0

C. Performance and Accuracy

I use all the default value for different SVM types in tool packet. I use 10,000 pictures for training and 1,000 pictures for testing. The result for different kernels is shown below.

SVM Type	Train Acc (%)	Test Acc (%)
Linear SVM	99.52	74.3
RBF Kernel	76.91	74.5
Poly Kernel	19.26	18.5
Linear Kernel	99.95	76.2
Sigmoid Kernel	63.27	62.2

TABLE III
Accuracy and Recall Rate

VI. LDA

Here we first use LDA to reduce dimension, then use Gaussian distribution for classification.

A. Dimension Reduction

Suppose the input dimension is d_{in}, the output dimension is d_{out}. The initial input training data is in shape $(d_{in}, samples)$. The LDA dimension reduction process is shown below.

1) Calculate between-class divergence matrix S_b

$$S_b = \sum_{i=1}^{10} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$

where N_i is the number of samples in category i, μ_i is the average vector for category i, μ is the average vector for all samples.
The shape of S_b is (d_{in}, d_{in})

2) Calculate within-class divergence matrix S_w

$$S_w = \sum_x (x - \mu_{lx})(x - \mu_{lx})^T$$

where μ_{lx} is the average vector for the category of one training sample x.
The shape of S_w is also (d_{in}, d_{in})

3) Problem Transformation. In original problem, we want to maximize $\frac{\beta^T S_b \beta}{\beta^T S_w \beta}$. We can transform the formula into Rayleigh quotient[2].

4) Now we need to calculate matrix $S_w^{-1} S_b$, then calculate the maximum $d_o ut$ eigenvalue and corresponding eigen-vector $(w_1, w_2, ..., w_{d_{out}})$. Now we get the projection matrix W.(This is also the learned model).
The shape of projection matrix W is (d_{out}, d_{in})

5) Finally, we can reduce input dimension with the help of projection matrix W. The reduced data z_i for one sample can be calculated by

$$z_i = (W x_i)^T$$

The reduced training data is in shape $(samples, d_{out})$

B. Gaussian Distribution Classification

Here we suppose training data in each category follows Gaussian distribution. After we perform linear discriminant analysis to reduce dimension, we get projected data in a lower dimension.

For projected data in each category, we use a Gaussian distribution to represent it. The detailed process is shown below.

1) For each category, we find the corresponding data with same label in training dataset. Then we calculate the mean and covariance matrix.
2) We count the number of elements in each category as prior probability.
3) We calculate the gaussian probability distribution function for all projected data in testing dataset. The Gaussian PDF can be written as

$$P(X|\theta) = \frac{1}{(2\pi)^{\frac{d_{out}}{2}} |\Sigma|^{1/2}} exp(-\frac{(X-\mu)^T \Sigma^{-1} (X-\mu)}{2})$$

For each category, we can get the pdf with shape (samples,)
4) Finally, for each sample data, we choose the index of highest probability in ten categories as predicted label.

C. Performance and Accuracy

I reduce the dimension to 20 with 20,000 training pictures and 5,000 testing pictures. The final accuracy for classification on testing dataset is 68.62%

VII. Neural Networks

Due to resource limitation, I don't use deep neutral network. Here is the basic convolution network structure I use. Based on this structure, I try to modify it and add new layers to find the effect on results. The detailed experiment design and conduction will be introduced in the next section.

Fig. 1. Basic CNN Structure

A. Performance and Accuracy

Here I use all the original pictures instead of HOG picture in training and testing dataset (73257 for training, 26032 for testing). The optimizer is SGD and the loss is cross entropy loss. The batch-size is set 128. The initial learning rate is 0.1 and decreases to half every five epochs. After 20 epochs, the training accuracy is 91.1% while the testing accuracy is 89.3%.

VIII. Image Generation with GAN

I try to use convolution GAN[3] to improve the quality of generated images. The network structure of discriminator convolution GAN is shown in Figure 2. The structure of generator substitute the convolution operation into transposed convolution (deconvolution).

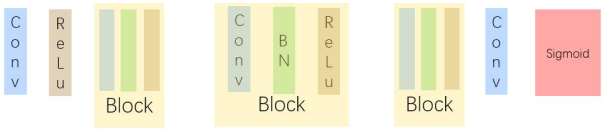

Fig. 2. Convolution GAN Structure (DCGAN)

I use Adaptive moment estimation (Adam) optimizer on Binary Cross-Entropy to train the network. The noise h is sampled on standard Gaussian distribution. The latent dimension of input is set as one hundred. The learning rate is set 0.001 and decreases to half every five epochs. After fifty epochs, the generated images are shown in Figure 3 below.

IX. Experiment and Analysis

A. Logistic Regression

It's weird that the regression without Ridge or Lasso loss has the highest accuracy. So I draw the accuracy curve during the training process. For these three logistic regression, the result is shown in Figure 4 below.

Fig. 3. Images Generated by DCGAN

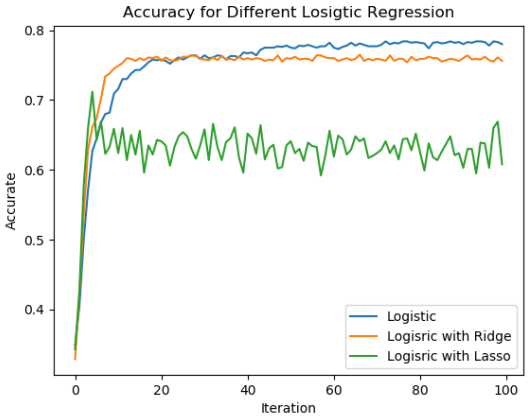

Fig. 4. Logistic with Different Loss

1) Ridge Loss: We can find that Ridge loss leads to a lower accuracy after 100 epochs compared with original logistic regression. However, it goes to convergence mush faster. In other words, Ridge loss takes less time to reach the optimum than logistic regression.

I think the lower accuracy comes from the feature of training data. For some specific categories, the HOG data feature is obvious. However, Ridge loss limits the impact of such obvious feature for robustness. As a matter of fact, the accuracy is a little lower.

Besides, the accuracy difference is small. We can infer that from number label 0 to 9, the number of categories with obvious feature is not too much.

2) Lasso Loss: I think the lower accuracy of Lasso loss comes from the feature of training data and the property of Lasso loss. The Lasso regression prefers sparse model. It will discard dimension information when training, which is regarded unimportant. However, the discarded dimension information may have a great impact on classification. In other words, for some specific category, the HOG feature is not so obvious and we can't classify it only based on a little feature information.

3) Conclusion: Combined with the performance of Ridge loss. We can infer that most categories (maybe 8 categories) have non-obvious features while a few categories (maybe the left 2 categories) can be easily classified with obvious feature.

4) Conclusion Validation: To verify our conclusion that the accuracy difference comes from dataset itself. We count the mis-classified data by Ridge loss and Lasso loss from original logistic classification.(i.e. we count the data classified correctly by logistic regression but classified wrongly after adding loss)

Based on our conclusion, the mis-classified data should have distinct category difference between two losses. However, the result is opposite, both Ridge and Lasso loss mis-classify similar categories in percentage.

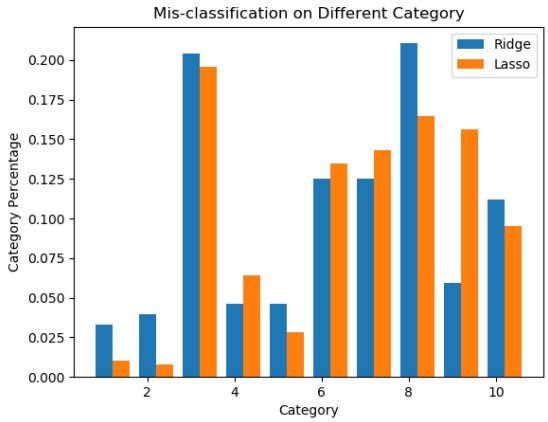

Fig. 5. Mis-classified Category Percentage

Therefore, we can exclude the dataset problem. Now we know these two loss act similarly in mis-classification. It means they haven't learnt enough like original logistic regression for classification. The problem is that the regularity term weights a lot and results in low accuracy. So we try to lower the coefficient of regularity term to verify again, from 0.01 to 0.001

Now we redraw the accuracy graph of these two losses. We can find the accuracy becomes similar. Especially for Lasso loss, the difference is obvious. Therefore, we can get the conclusion about low accuracy in Figure 4 compared with original logistic regression. The reason is that the coefficient of regularity term is too large for model and it prevents model to learn from data.

B. Kernel-based Logistic Regression

We can find in previous result table that the choice of different kernel has great impact on the model performance. We draw the accuracy curve of different kernel first.

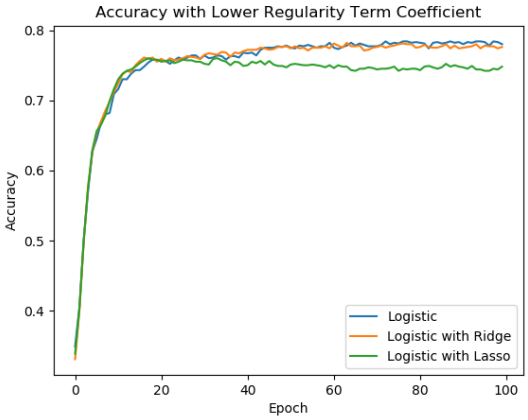

Fig. 6. Logistic with Different Loss Under Lower Coefficient

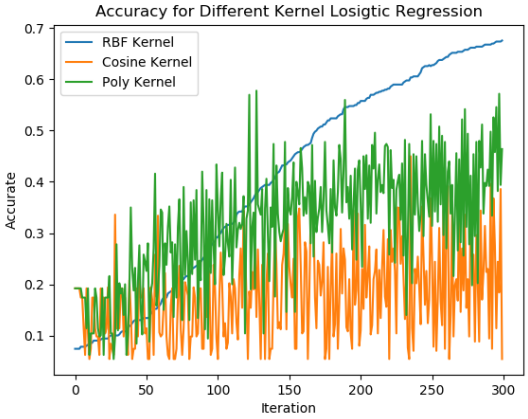

Fig. 7. Logistic with Different Kernel

It's obvious RBF kernel has best performance. The left two kernels leads to unstable model and low accuracy. It's taught on class that the choice of kernel is based on experience. For different data, we need to try different kernel for better performance and there is no solid theory basis.

Therefore, our conclusion is that, for SVHN dataset, RBF kernel has best effect for kernel-based logistic regression.

C. Support Vector Machine

This is the same with kernel-based logistic regression. We can notice that the training accuracy of linear SVM is much higher than testing accuracy due to over-fitting. When more data are used to for training, the accuracy of linear SVM may get lower. Therefore, the RBF kernel also has best effect. And this can also work as a cross-validation for the good performance of RBF kernel in SVHN dataset.

D. LDA

In Linear Discriminant Analysis, we first project input data to a lower dimension. The projection maximizes the inter-class variance and minimizes the intra-class variance. Therefore, the projected data can be classified easier that origin data.

Then based on projected data, we use ten Gaussian distribution to describe ten different categories. We use the same distribution learned from training data to classify testing data.

To verify the influence of LDA, we try to apply Gaussian distribution on origin HOG input data instead of dimension reduced data. In other words, we skip the LDA process to find the difference. The accuracy comparison is listed below.

whether LDA?	Test Accuracy (%)	Train Accuracy (%)
Use LDA	68.62	77.03
No LDA	9.78	6.97

TABLE IV
Effect of LDA

To make the result more intuitive, I use t-SNE to visualize the distribution of 2,000 training data before and after LDA.

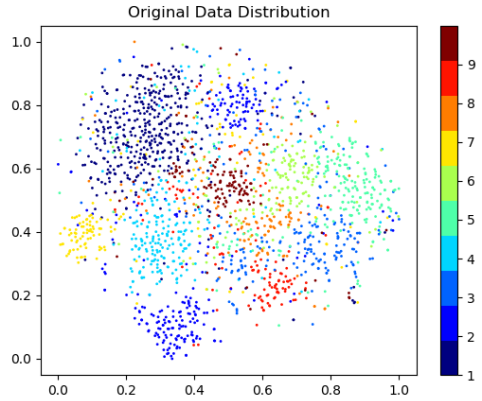

Fig. 8. Distribution of Data before LDA

It's obvious that LDA successfully classify the input data into different categories by projecting it to lower dimension. The result also shows the distribution of projected training data is similar to Gaussian distribution. The conclusion comes from the training accuracy after LDA. The higher training accuracy is, the more likely projected data follows Gaussian distribution

E. Neural Network

The curve of train and test accuracy in 20 epochs are drawn below based on the basic CNN architecture in previous section. We can find the accuracy converges quickly after first a few epochs. The final testing accuracy is 89.3%

We can notice that the accuracy for testing is even higher than training at first few epochs. I think this is because the pictures in testing dataset contain more

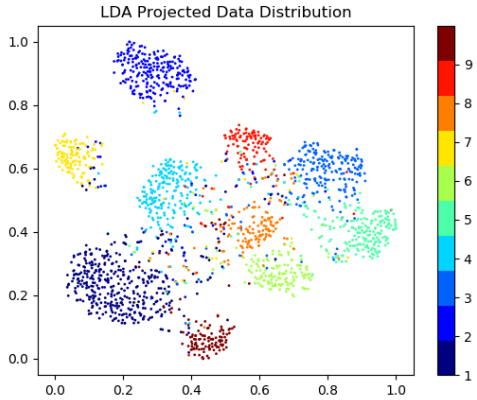

Fig. 9. Distribution of Data after LDA

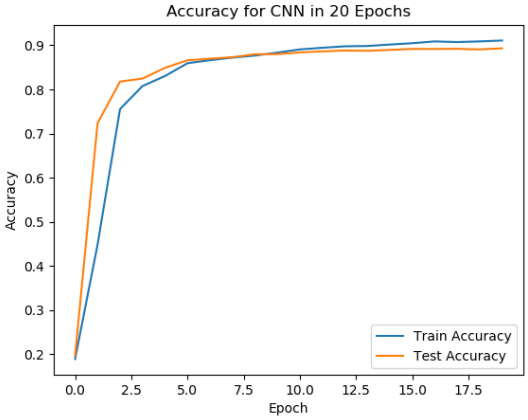

Fig. 10. Accuracy for CNN

obvious feature and the model has not learnt enough from training dataset. After training more, the accuracy for training catches up to testing accuracy.

Then we want to test the function of batch normalization layer. We add two BN layers after convolution layers separately to find the difference. The accuracy comparison Figure 11 is shown below.

We can find that the batch normalization helps improve the accuracy in finite epochs. The final testing accuracy changes from 89.31% to 91.31%.

Besides, the model can learn much more for the first epoch. The origin CNN model can only reach 19.73% for testing while the BN-added CNN can achieve 78.11% testing accuracy.

In conclusion, Batch-Normalization helps model learn from data much faster. This is because such operation ensures that the input for every layer is distributed around the same mean and standard deviation.

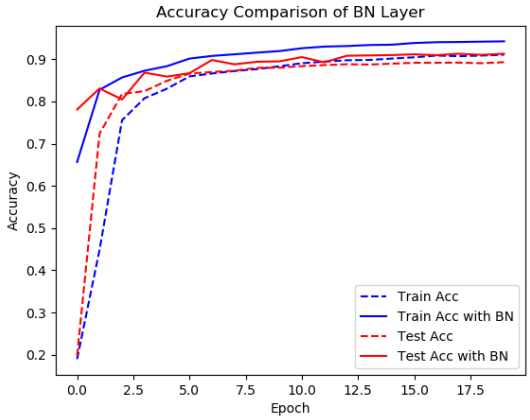

Fig. 11. Accuracy Comparison for BN Layer

X. Conclusion

In this project, I implement most of classifiers mentioned in project requirement and design experiment to verify conclusion. I learn a lot from these classifiers and have a deeper understanding of the basic theory foundation.

The abstract workflow of this project is listed here as final conclusion.

- Logistic regression with Ridge and Lasso loss. The designed experiment shows that the low accuracy of loss comes from unfitting regularization term coefficient instead of dataset itself.
- Kernel-based logistic regression with Lasso loss. I test different kernels and find the RBF kernel has the best performance for SVHN dataset in regression.
- SVM. This is similar to kernel-based logistic regression. I use sklearn tool packet to implement SVM. I try different kernel options and find RBF kernel has best effect. This is consistent with the conclusion from kernel-based logistic regression.
- LDA. I use LDA for dimension reduction and use Gaussian distribution to simulate the distribution of projected data and classify them. I compare the difference of accuracy without LDA. And I use t-SNE to visualize the distribution of data before and after LDA process.
- Neutral Network. I design the convolution network structure myself. And I design experiment to find the influence of batch-normalization layer.

References

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, 2005, pp. 886–893 vol. 1.

[2] Pinard, "Lda principle summary," accessed January 3, 2017. [Online]. Available: https://www.cnblogs.com/pinard/p/6244265.html

[3] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," arXiv e-prints, p. arXiv:1511.06434, Nov. 2015.