Report For CS385 Personal Project

Yanming Liu; ID: 518030910393

June 18, 2021

Declare
All the code in this project is written by myself, and you can check it : `https://github.com/lym01803/CS385_Project1`, which is my personal repository. I only use third-party libraries for the following:

- Visualization, e.g. PCA, t-SNE, draw curve and histogram.
- SVM.
- HOG feature.
- For CNN model. The network is built by myself. But I use API for network layers, e.g. Conv, BN.
- Basic matrix operations. For simple models, I only use pytorch to do simple matrix operations on cuda. I do not use any advanced API, such as built-in network layers, optimizers and auto-gradient. In fact, to confirm it, I set the requires_grad attribute of tensors to False.

Note
During the project, I discussed and exchanged opinions with my classmates many times. In the analysis of logistic regression model, my opinion may be similar to that of my classmate Yuxuan Xiong. Please note that if there are some similarities in our analysis, it is the result of our discussion, not plagiarism.

1 Introduction

In this project, I do the following things:

- Processed the data set;
- Implement several basic models: Logistic Regression, LDA, kernel based Logistic Regression, SVM and CNN;
- Try two regular terms: Ridge and Lasso;
- Visualization: HOG feature, CNN feature and distribution of LDA and Logistic Regression;
- Analysis: mainly about the logistic regression.

In fact, the models in this project are all discriminative models, and they are all closely related to logistic regression: LDA and logistic regression have different goals, but they take the same way to solve the discrimination problem: learning a projection direction to separate data; The motivation of the SVM model comes from the discussion of the margin in linear binary discrimination problem; The kernel is an extension of the linear model to the nonlinear model, which is equivalent to a linear model for high-dimensional (probably non-linear) features $\phi(X)$; CNN can be regarded as using a learnable deep network to produce the features of input images instead of using the manual designed HOG features, which is essentially a binary classifier (logistic regression) or a multi-classifier (extension of logistic regression, sigmoid -> softmax) of the CNN based features. Therefore, in the subsequent analysis of the project, the discussion will mainly focus on a basic point, the linear logistic regression model.

2 Data Process

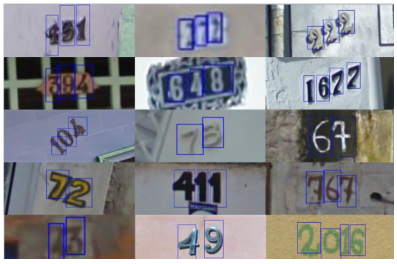
2.1 Crop and Padding

The given data set has two format: 1. original images with character level bounding boxes; 2. The cropped images.

Since the borders of digits are usually not square, the square cropped images mostly contain part of other digits. Although someone says that such noise does not affect the performance of the classifier (most likely because the translation invariance of HOG is not good), I still do not plan to use it.

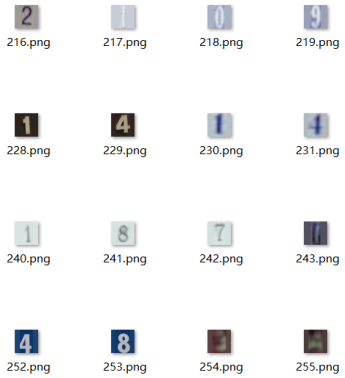
Therefore, I crop the original images by myself according to the given digit boxes, then padding the images into squares symmetrically, and then resize them to a size of 32×32. For the color to padding, I choose the average color of the border pixels of cropped images (averaged for each channel). I am not sure if such an operation meets the specifications of the CV field, but at least the images obtained in this way seems to be of high quality to my eyes and is suitable for training discriminative models, I think.

The comparison between the two official offered data and the data processed by myself is shown in the followed Fig. 1.



Format 1: original images

Format 2: cropped images



Images cropped and paddinged by myself

Figure 1: Comparison between official data and data processed by myself

2.2 HOG Feature

General experience tells us that directly using the three-channel pixels of an image as the input features may not be effective. It is usually believed that the local gradient (both norm and direction) of an image can reflect image features such as object edges and textures. Therefore, the HOG feature of the image is used as the input of models (except CNN) here. A third-party library cv2.HOGDescriptor is used here. The parameters are shown in the following Table 1, and finally 1764-dimensional feature vectors are obtained.

Table 1: The HOG parameters setting

input size	$32 \times 32 \times 3$	window size	32×32
block size	8×8	block stride	4×4
cell size	4×4	bin	9

3 Logistic Regression

3.1 Implement

I follow eq. 1.

$$\begin{aligned} L_i &= y_i \log p_i + (1 - y_i) \log(1 - p_i) \\ &= y_i X_i^\top \beta - \log(1 + \exp(X_i^\top \beta)) \\ \frac{\partial}{\partial \beta} L_i &= y_i X_i - \frac{\exp(X_i^\top \beta)}{1 + \exp(X_i^\top \beta)} \cdot X_i \\ &= (y_i - p_i) \cdot X_i \end{aligned} \tag{1}$$

For efficient calculation, I do not calculate the loss L_i in the program. Instead I directly calculate the gradient $\partial L_i / \partial \beta$ and perform gradient descent.

For prediction, typical logistic regression is a bianry discriminative model, so for the 10-classes discrimination problem, I use 10 independent logistic regression models. The $i-$th $(0 \leq i \leq 9)$ model predict the probability $Pr(y = i|X, \beta_i)$. So predicted class is calculated by $\texttt{predict} = \arg\max_{0 \leq i \leq 9} Pr(y = i|X, \beta_i)$.

3.2 Result

Because 10 independent binary discriminative models are used here, and the result of argmax is used as the final predicted value, the model evaluation indicators here are shown as follows:

- Evaluate the performance of the logistic regression model: the accuracy of a single logistic regression model on their respective binary discrimination tasks.
- Evaluate the performance of the multi-discrimination model composed of 10 binary discriminator:
  - Evaluate the recall rate corresponding to each digit.
  - Evaluate the overall accuracy of the model on the multi-discrimination task.

The experiment settings are shown in the Table 2: **Note** Here all parameters tuning are in order to study the model, not to improve the score on test set, so there is no need to set a validation set.

Table 2: Experiment Settings

Train Data	73257 images from train folder
Test Data	26032 images from test folder
Learing Rate	1.0, 0.1, 0.01 or 0.001
Ridge Coefficient	0.1, 0.01, 0.001 or 0
Lasso Coefficient	0.1 or 0
Epoch	100
Batch Size	256
Optimizing Method	SGD or Adam

3.2.1 Performance - learning rate

Use Adam. No Ridge loss or Lasso loss. The result is shown in Table 3

Table 3: Acc-lr

lr	Train Acc	Test Acc
1.0	0.7902	0.7105
1e-1	0.8673	0.7939
1e-2	0.8716	0.8137
1e-3	0.8545	0.8175

There are more details in Fig. 2.

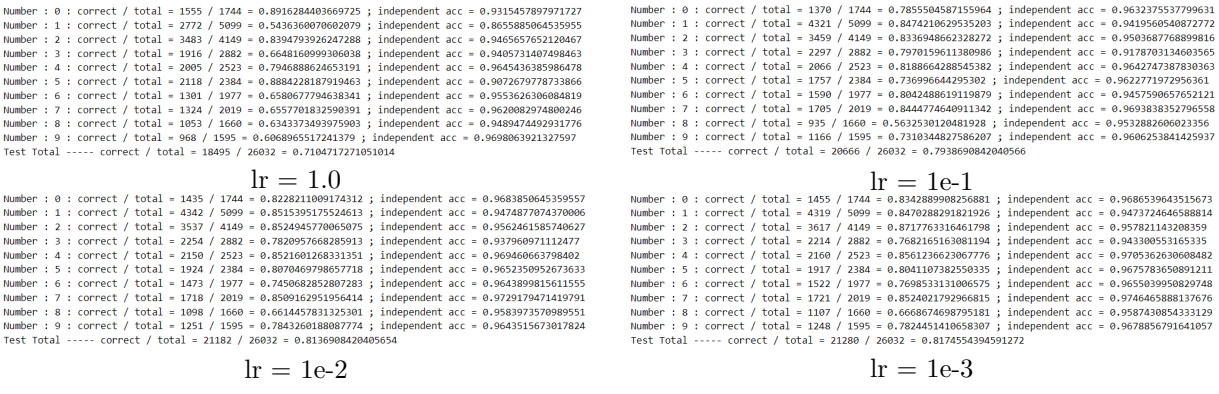


Figure 2: Acc-lr details

Trivial (so I will show as few of these curves as possible later) performance curves are shown in Fig. 3.

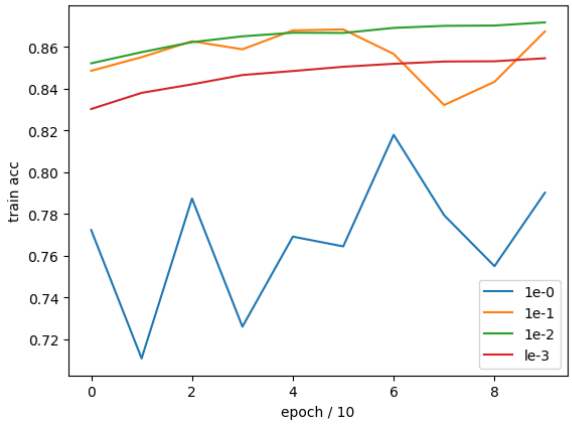


Figure 3: Performance curves

Conclusion: I think it is appropriate to set the learning rate to 1e-2 here.

3.2.2 Performance - Adam

No Ridge or Lasso loss. lr = 1e-2. The result is shown in Table 4

Table 4: Acc-Adam

Optimizer	Train Acc	Test Acc	Running Time
SGD	0.8402	0.8105	about 30 s
Adam	0.8716	0.8137	about 60 s

Conclusion: SGD is a little faster. Adam is better at fitting training data, in other words, its ability to solve the function minimums is stronger. However, there is no significant difference in the performance on the test set. Therefore, the Optimizer is not a particularly import thing.

3.2.3 Performance - Ridge / Lasso

Use Adam. Set lr = 1e-2. The result is shown in Table 5.

Table 5: Performance-Ridge / Lasso

Ridge coef	Lasso coef	Train Acc	Test Acc	Norm1	Norm2	Norm1 / Norm2
1e-1	-	0.8366	0.8036	160.22	5.56	28.79
le-2	-	0.8548	0.8165	382.20	12.57	30.40
1e-3	-	0.8647	0.8121	865.84	26.67	32.47
-	1e-1	0.8222	0.7983	76.73	9.01	8.52
-	1.0 (decay) *	0.8638	0.8185	761.72	25.36	29.82

* The decay method : lasso coef start at 1.0, and decay by 0.1 every 20 epochs. This idea is from the Lasso Regression discussed in class. But the above experimental results show that this method, decaying during iteration, has no effect **here**.

Conclusion: As regular term, Ridge loss and Lasso loss will reduce the fitting performance on the train set. The ability to avoid over-fitting is also limited, at least no significant improvement on test set can be seen from the above experimental results. Ridge loss can indeed reduce the Norm2 of the parameter (β here), but notice that it will also reduce Norm1 at the same time. In fact, when the coef of Ridge takes different values, the values of Norm1 / Norm2 are almost the same. The Lasso loss more significantly reduces the train set fitting ability. Lasso loss can reduce both Norm1 and Norm2, but it is worth noting that it can significantly reduce Norm1 / Norm2, which is a quantity that I think can reflect the sparseness of the parameter.

It also confirms a problem discussed in class. A linear model can perform well on the test set, but this does not necessarily mean that it is a good model. We should pay attention to what the parameters of linear model have learned. In order to prevent the parameters from learning too much redundant information, we introduced

Lasso. Lasso can reduce Norm1 / Norm2, which means that the parameters are sparser. That is actually what we expect. We do not expect Lasso to significantly improve the performance of model on the test set.

3.2.4 Distribution of $X^\top\beta$

This problem is very interesting. I will analysis it later together with LDA.

4 Linear Discriminant Analysis (LDA)

4.1 Implement

4.1.1 Train

Just follow the eq. 2:

$$\begin{aligned} S_W &= n^+\Sigma^+ + n^-\Sigma^- \\ \beta &= S_W^{-1}(\mu^+ - \mu^-) \end{aligned} \tag{2}$$

4.1.2 Predict

LDA is originally a visualization tool and is not directly used as a binayr classifier. But considering that, through LDA we can get the projection $X^\top\beta$, and can classify it according to this projection. The simplest way is to calculate a threshold to classify.

However, here we need to construct 10 binary LDA classifiers to do the multi-classify task. That is to say, it is unreasonable to directly predict a 0 or 1 result. Because of the model errors, it is possible that an image is predicted as a positive sample in more than one classifiers, which makes it difficult to perform multi-classfication. Therefore, I am here to give an approach based on conditional probability:

We **assume** that the positive and negative samples of $X\beta$ after the LDA projection respectively obey the normal distribution: $(X^\top\beta)^+ \sim N(\mu^+, {\sigma^+}^2)$ and $(X^\top\beta)^- \sim N(\mu^-, {\sigma^-}^2)$. Then we can write the conditional probability :

$$\begin{aligned} P(y=+|X^\top\beta) &= \frac{P(y=+)\cdot P(X^\top\beta|y=+)}{P(y=+)\cdot P(X^\top\beta|y=+) + P(y=-)\cdot P(X^\top\beta|y=-)} \\ &= \frac{n^+ \cdot \frac{1}{\sqrt{2\pi}\sigma^+}\exp\left(-\frac{1}{2{\sigma^+}^2}(X^\top\beta-\mu^+)^2\right)}{n^+ \cdot \frac{1}{\sqrt{2\pi}\sigma^+}\exp\left(-\frac{1}{2{\sigma^+}^2}(X^\top\beta-\mu^+)^2\right) + n^- \cdot \frac{1}{\sqrt{2\pi}\sigma^-}\exp\left(-\frac{1}{2{\sigma^-}^2}(X^\top\beta-\mu^-)^2\right)} \end{aligned} \tag{3}$$

Follow the eq. 3, the predicted value of LDA model is a probability between 0 and 1, and its meaning is the same as the predicted value in logistic regression. In this way, we can use the same approach as the logistic regression to do the multi-classfication task.

4.2 Result

4.2.1 Settings

Just use the same train and test data set as logistic regression.

4.2.2 Accuracy

The result is shown below in Fig. 4:
The first 10 lines show the recall rate for each digit. The last line shows the overall accuracy of the multi-classifier.

Figure 4: Acc for LDA

```
Number : 0 : correct / total = 1434 / 1744 = 0.8222477064220184
Number : 1 : correct / total = 4311 / 5099 = 0.8454598940968817
Number : 2 : correct / total = 3559 / 4149 = 0.8577970595324175
Number : 3 : correct / total = 2164 / 2882 = 0.7508674531575295
Number : 4 : correct / total = 2125 / 2523 = 0.8422512881490289
Number : 5 : correct / total = 1815 / 2384 = 0.7613255033557047
Number : 6 : correct / total = 1414 / 1977 = 0.7152250885179565
Number : 7 : correct / total = 1690 / 2019 = 0.8370480435859337
Number : 8 : correct / total = 1077 / 1660 = 0.6487951807228916
Number : 9 : correct / total = 1207 / 1595 = 0.7567398119122257
Test Total ----- correct / total = 20796 / 26032 = 0.7988629379225568
```

4.2.3 Distribution of $X^\top\beta$

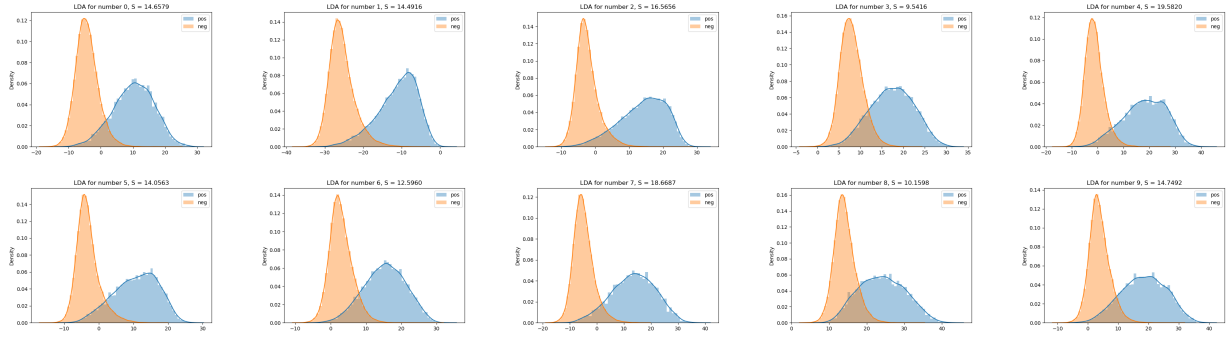


Figure 5: Distribution of $X^\top\beta$ for LDA

I am interested in what the distribution of $X^\top\beta$ of logistic regression looks like.

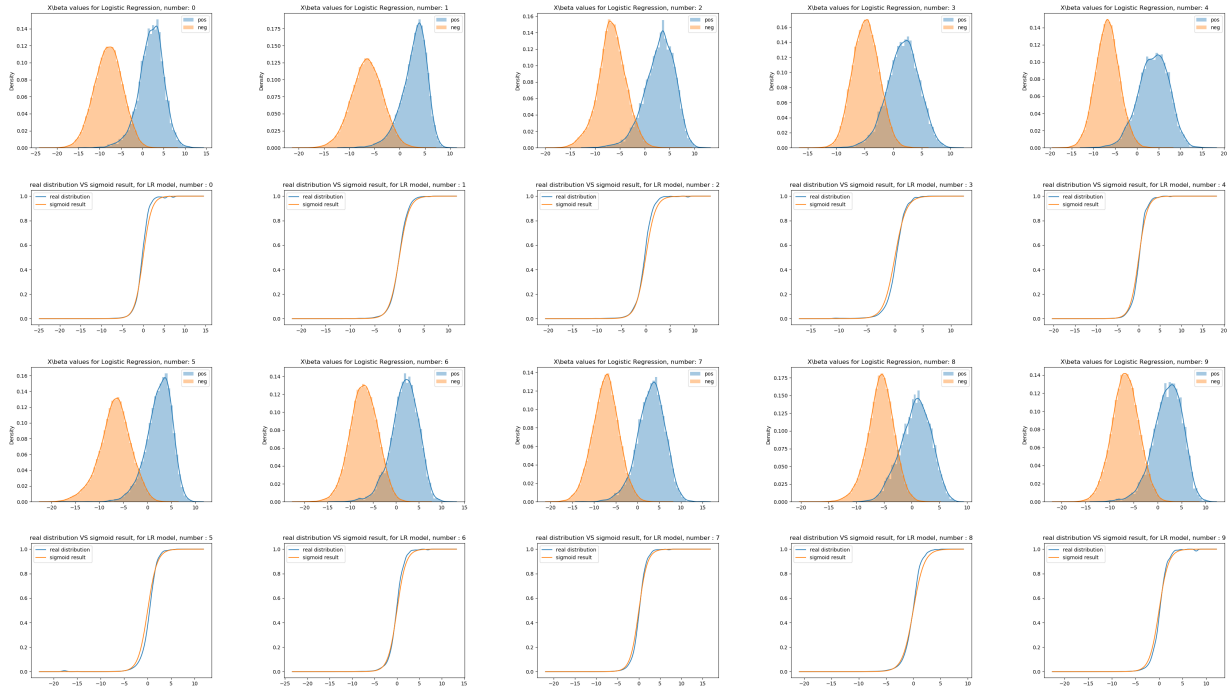


Figure 6: Distribution of $X^\top\beta$ and $Pr(y = +1|X^\top\beta)$ for LR

Fig. 6 show the distribution of $X^\top\beta$ and $Pr(y = +1|X^\top\beta)$. The figure shows:

- The projection $X^\top\beta$ is close to normal distribution, intuitively.
- Projections of LDA and LR are similar, but different.
  - LDA learn a projection direction to maximize $S = \dfrac{\sigma^2_{inter}}{\sigma^2_{intra}}$;

  - LR learn a projection to maximize the likelihood

- LR learn a projection direction to make the conditional probability $f(X^\top\beta) = Pr(y = +1|X^\top\beta)$ close to the sigmoid function $g = \texttt{sigmoid}(X^\top\beta)$

Note that there is a logic chain: LR learn a projection direction to make the conditional probability distribution close to sigmoid function. And LR use sigmoid / logistic function. -> If we choose another function, and make the conditional probability distribution close to it, then we will get another regression model ! -> The reason why we choose sigmoid function may be that sigmoid function is a reasonable conditional probability distribution. But why?

I think I can give an explanation for leftover problem in our class: Why we use logistic function to do logistic regression ? Why we use $X^\top\beta = \log\frac{p}{1-p}, p = \texttt{sigmoid}(X^\top\beta)$?

Explanation

Assume that $Pr(X^\top\beta|y = +) \sim N(\mu_+, \sigma_+^2)$, $Pr(X^\top\beta|y = -) \sim N(\mu_-, \sigma_-^2)$. Why assume a normal distribution? When there is no known condition, if we have to make a assumption, then it is more reasonable to assume it as a normal distribution. And the above visualization of LDA also tells us that this distribution is like a normal distribution. Then we can write :

$$Pr(y = +|X^\top\beta) = \frac{Pr(y = +) \cdot \texttt{const}_1 \cdot \exp\left(-\frac{1}{2\sigma_+^2}(X^\top\beta - \mu_+)^2\right)}{Pr(y = +) \cdot \texttt{const}_1 \cdot \exp\left(-\frac{1}{2\sigma_+^2}(X^\top\beta - \mu_+)^2\right) + Pr(y = -) \cdot \texttt{const}_2 \cdot \exp\left(-\frac{1}{2\sigma_-^2}(X^\top\beta - \mu_-)^2\right)} \tag{4}$$

Assume that $\sigma_+ = \sigma_-$, and hence $\texttt{const}_1 = \texttt{const}_2$. Why? No reason, just for calculation convenience. Let $\alpha = \frac{Pr(y = -)}{Pr(y = +)}$, then we can rewrite it as :

$$\begin{aligned} Pr(y = +|X^\top\beta) &= \frac{\exp\left(-\frac{1}{2\sigma^2}(X^\top\beta - \mu_+)^2\right)}{\exp\left(-\frac{1}{2\sigma^2}(X^\top\beta - \mu_+)^2\right) + \alpha \cdot \exp\left(-\frac{1}{2\sigma^2}(X^\top\beta - \mu_-)^2\right)} \\ &= \frac{1}{1 + \exp\left(\log(\alpha) - \frac{1}{2\sigma^2}\left(2(\mu_+ - \mu_-)X^\top\beta + (\mu_-^2 - \mu_+^2)\right)\right)} \\ &= \frac{1}{1 + \exp(-(aX^\top\beta + b))} = \texttt{sigmoid}(aX^\top\beta + b) \end{aligned} \tag{5}$$

where $a = \frac{\mu_+ - \mu_-}{\sigma^2}$, $b = \frac{\mu_-^2 - \mu_+^2}{2\sigma^2} - \log\alpha$. And $b \approx -\log\alpha$, if we assume $\mu_- + \mu_+ \approx 0$.

Furthermore, notice such a feature of the distribution of projection $X^\top\beta$ in LR, that the distributions of $X^\top\beta$ for positive and negative samples are not symmetrical about the axis origin, they are shifted to the negative half axis instead. From the intersection of the two curves on the figure, it can be seen that the offset is close to $\log 9$. The positive-negative ratio of our train samples is about 1:9, i.e. here $\alpha = 9$. It seems to be in line with our inference above $b \approx -\log\alpha$.

Now, I assert that if the positive-negative ratio is 1:1, then the distributions will almost not shifted, and the intersection will be around 0.

I perform an experiment to check this assertion. However, for convenience, instead of really sampling 1:1 samples, I use a trick: give positive samples 9 times the weight when calcuting the gradient. I think it is almost equivalent to sample at a ratio 1:1, at least almost the same during performing gradient descent.

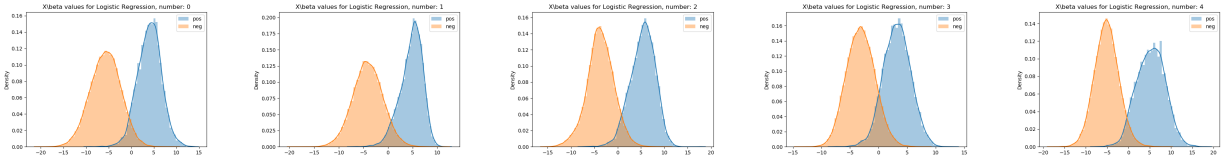


Figure 7: Distribution of $X^\top\beta$ for weighted LR (equivalent to sampling positive and negative samples at a ratio 1:1)

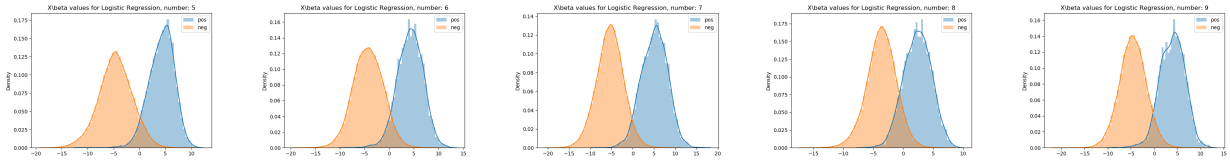


Figure 8: Distribution of $X^\top \beta$ for weighted LR (equivalent to sampling positive and negative samples at a ratio 1:1)

5 Kernel Based LR

5.1 Implement

All are based on $\beta = \sum_i \alpha_i X_i$, i.e. parameter must be the linear combination of input features. So for kernel trick, $\beta = \sum_i \alpha_i \phi(X_i)$. So for training, we have:

$$p_i = \texttt{sigmoid}(\phi(X_i)^\top \beta) = \texttt{sigmoid}(K_{i,:}\alpha) \tag{6}$$

$$\frac{\partial}{\partial \alpha} L_i = (y_i - p_i) K_{i,:}^T \tag{7}$$

For prediction, it is almost the same.
Here I use the **rbf** kernel.

5.2 Result

5.2.1 Settings

The parameters are the same as LR. But because the kernel matrix can not be too big, so in this model, I only use the first 10000 images in the original train folder as the train set.

5.2.2 Performance - rbf coef

In rbf kernel, we have a hyperparameter σ, $K(X_i, X_j) = \exp\left(-\frac{1}{2\sigma^2}||X_i - X_j||_2^2\right)$. From my personal opinion, rbf kernel is just to find the data similar to the input data among the saved train data. And the measure of similarity is the rbf kernel.

In my opinion, σ is an estimate of the size of the space that each support vector can support. A too small σ corresponds to that each point only cares about the local space near it, which will degenerate the model into a look-up table, resulting in overfitting.Too large σ corresponds to that each points supports a large piece of space, which is beyond the scope of itself, so that the similarity between the input data and the training samples can not be accurately measured, which will lead to under-fitting.

The results are shown in Table 6.

Table 6: Performance - rbf coef

$2\sigma^2$	Train Acc	Test Acc
1	1.0000	0.6917
10	0.9847	0.8182
100	0.8167	0.7472

The results are in line with my expectations, and can verify the assertion I mentioned above.
Kernel based LR model is better than linear LR model in terms of fitting ability, at least the latter can not even fit the train data set to 1.0000 Acc.

6 SVM

6.1 Implement

Use the third-party library sklearn.svm.SVC.
Use 10 independent binary SVM classifiers to do the multi-classify task.

6.2 Result

6.2.1 Settings

Train and test data set settings are the same as that of Kernel based LR model.
Use rbf kernel. I do not try others kernel. In the absence of sufficient theoretical support, trying many different kernel will yield little benefit. The rbf kernel is used because in the subsequent t-SNE visualization, the features appear clustered.

6.2.2 Performance

I tried two ways in prediction step:

- SVM only output 0 or 1 as the prediction result. If there are more than one models output 1, than randomly choose one as the prediction result of multi-classify task.
- SVM report the decision function value, i.e. tells me the distance to the bound. Then take argmax as the result for multi-classify task.

```
Number : 0 : correct / total = 1298 / 1744 = 0.7442660550458715; single acc: 0.9729563614013522
Number : 1 : correct / total = 4022 / 5099 = 0.7887821141400274; single acc: 0.9491779348494161
Number : 2 : correct / total = 3458 / 4149 = 0.8334538442998313; single acc: 0.965119852489244
Number : 3 : correct / total = 1769 / 2882 = 0.6138098542678695; single acc: 0.9507529194837123
Number : 4 : correct / total = 1943 / 2523 = 0.7701149425287356; single acc: 0.9708819913952059
Number : 5 : correct / total = 1740 / 2384 = 0.7298657718120806; single acc: 0.9700368776889982
Number : 6 : correct / total = 1316 / 1977 = 0.6656550328780981; single acc: 0.9694222495390289
Number : 7 : correct / total = 1561 / 2019 = 0.7731550272412085; single acc: 0.9780654578979717
Number : 8 : correct / total = 811 / 1660 = 0.48855421686746986; single acc: 0.9630454824830977
Number : 9 : correct / total = 1082 / 1595 = 0.6783699059561129; single acc: 0.97529963122311
Total correct / total = 19000 / 26032 = 0.7298709280885064
```

report 0 or 1

```
Number : 0 : correct / total = 1501 / 1744 = 0.8606651376146789; single acc: 0.9701521204671174
Number : 1 : correct / total = 4398 / 5099 = 0.8625220631496372; single acc: 0.9192916410571604
Number : 2 : correct / total = 3769 / 4149 = 0.908411665461557; single acc: 0.9457590657652121
Number : 3 : correct / total = 2195 / 2882 = 0.7616238723108952; single acc: 0.935540872771973
Number : 4 : correct / total = 2185 / 2523 = 0.8660325009908839; single acc: 0.9569376152427781
Number : 5 : correct / total = 1972 / 2384 = 0.8271812080536913; single acc: 0.9562845728334358
Number : 6 : correct / total = 1566 / 1977 = 0.7921092564491654; single acc: 0.9571296865396435
Number : 7 : correct / total = 1742 / 2019 = 0.8628033680039624; single acc: 0.9686539643515673
Number : 8 : correct / total = 1140 / 1660 = 0.6867469879518072; single acc: 0.951521204671174
Number : 9 : correct / total = 1269 / 1595 = 0.7956112852664576; single acc: 0.9666948371235402
Total correct / total = 21737 / 26032 = 0.8350107559926244
```

report decision function value, i.e. tells the distance to the bound

Figure 9: SVM results

The results are shown in Fig. 9.

7 CNN

7.1 Implement

I use a very simple toy CNN designed by myself.

Table 7: CNN layers

Layer Number	Layer	Layer Number	Layer
1	Conv(3, 64, 3, 2, 1)	2	BatchNorm + ReLU
3	Conv(64, 128, 3, 2, 1)	4	BatchNorm + ReLU
5	Conv(128, 256, 3, 2, 1)	6	BatchNorm + ReLU
7	Conv(256, 512, 3, 2, 1)	8	BatchNorm + ReLU
9	Conv(512, 1024, 3, 2, 1)	10	ReLU
11	FC(1024, 10)	12	Softmax

The network structure is shown the Table 7. The Conv layer is given in the format Conv(input channel, output channel, kernel size, stride, padding). The FC layer is given in the format FC(input channel, output channel). Why use kernel size = 3, stride = 2, padding = 1 ? Because in this way, the height and width of feature will reduce to half after each Conv layer.

7.2 Result

7.2.1 Settings

The train and test data set is the same as that of LR model.
Use Adam optimizer provided by pytorch library. lr = 1e-3. Train for 30 epochs.

7.2.2 Performance

Overall Accuracy: 0.9015 The loss curve is shown in Fig. 10.

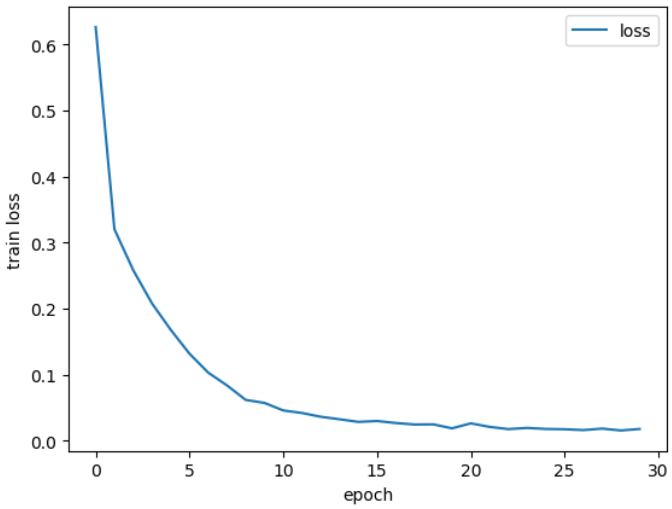


Figure 10: CNN loss curve

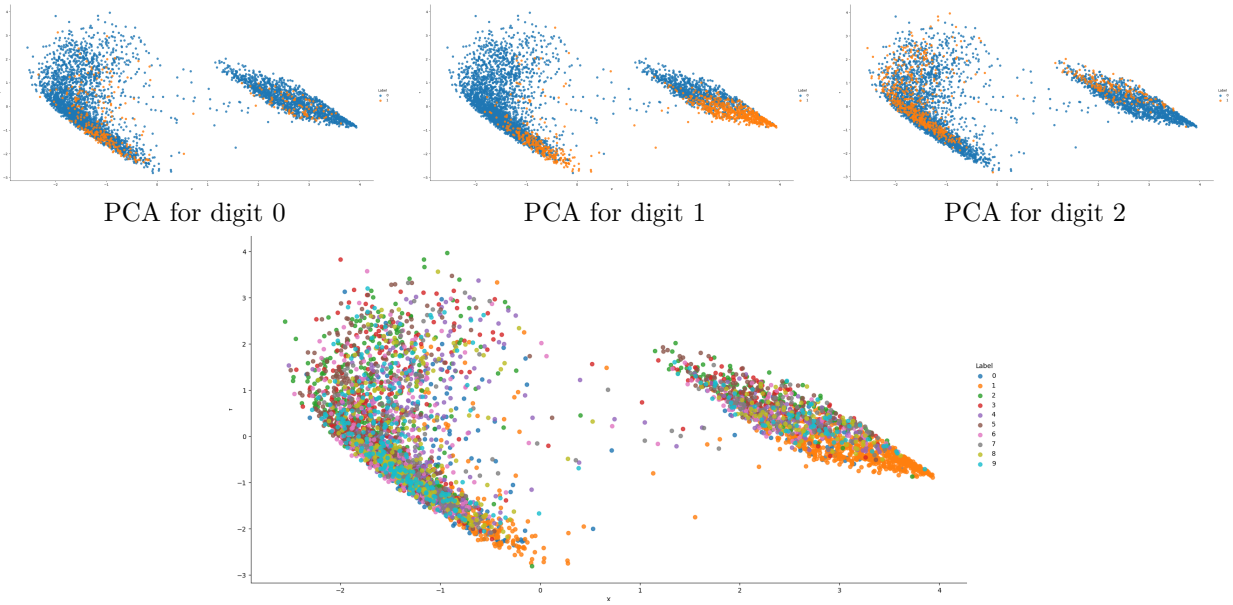
7.3 Analysis

I have not studied the nature of CNN too much. Here I just treat CNN as a learnable feature extractor + a linear multi-classifier. For linear classifier, I have discussed before. I will show the part of feature extractor in the visualization part later.

8 Visualization of Features

8.1 For HOG features

8.1.1 PCA



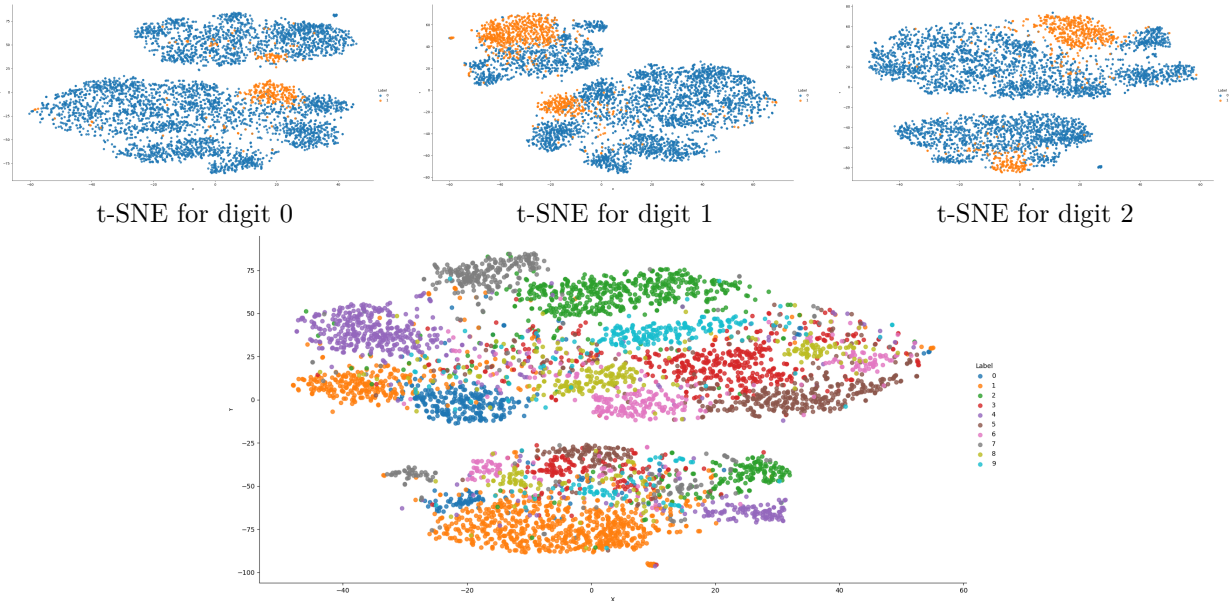
PCA for digit 0 PCA for digit 1 PCA for digit 2

Figure 11: PCA for digits 0, 1 and 2, and t-SNE for all 10 digits

Fig. 11 shows the PCA visualization of HOG features for the digit 0, 1 and 2 and for all 10 digits. Seems not very linearly separable.

8.1.2 t-SNE

First using PCA reduce the features from 1764-dim to 100-dim, then performs t-SNE.



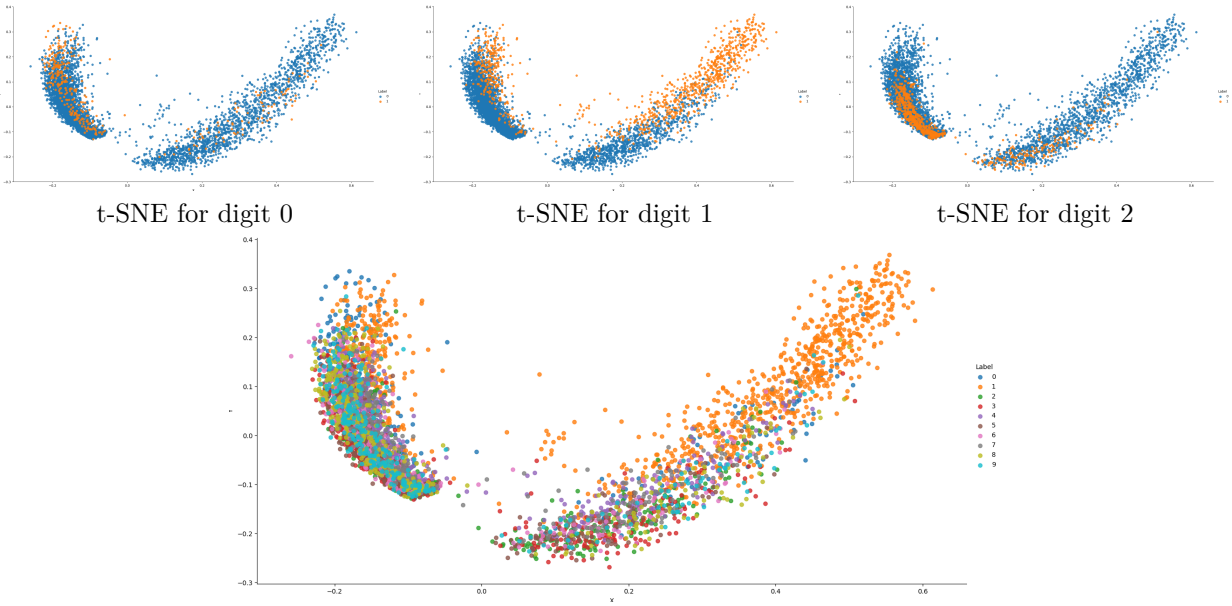
t-SNE for digit 0 t-SNE for digit 1 t-SNE for digit 2

Figure 12: t-SNE for digits 0, 1 and 2, and t-SNE for all 10 digits

Fig. 12 shows the t-SNE visualization of HOG features for the digit 0, 1 and 2 and for all 10 digits. Seems suitable to solved by kernel method, especially rbf kernel.

8.1.3 Kernel PCA

Perform Kernel PCA to do the visualization with rbf kernel (gamma $= \frac{1}{2\sigma} = 10$ here. This setting is from my experiment on Kernel based LR).



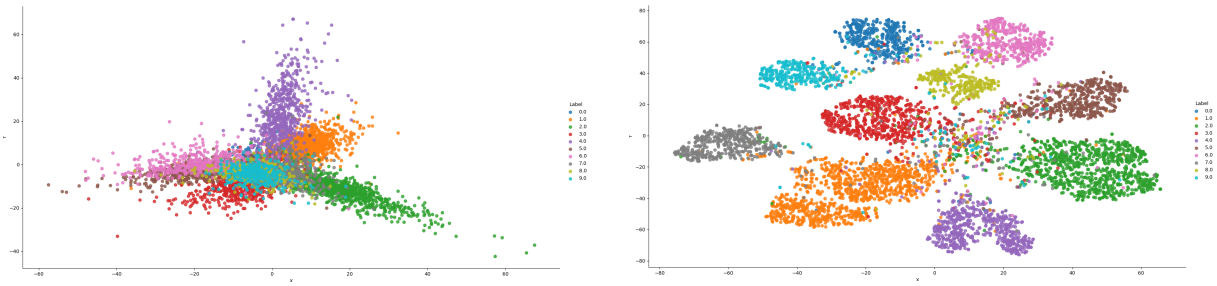
t-SNE for digit 0 t-SNE for digit 1 t-SNE for digit 2

Figure 13: t-SNE for digits 0, 1 and 2, and Kernel PCA for all 10 digits

Fig. 13 shows the Kernel PCA visualization of HOG features for the digit 0, 1 and 2 and for all 10 digits. Seems much more separable than that of PCA. This may be a reason for why I use rbf kernel in this project.

8.2 CNN Feature

I treat CNN as a feature extractor + a classifier. So I want to know the performance of this feature extractor. I visualization the features from the last layers except FC layer. The result is shown in Fig. 14



PCA PCA to 100-dim + t-SNE

Figure 14: CNN Feature Visualization

Fig. 14 tells why the test Acc of CNN model is high. The learnable convolutional feature extractor perform much better than the manual feature extractor HOG. The features are significantly more separable.

9 A Naive And Immature Idea

We have discussed in class that the error ϵ of the regression model should be uncorrelated to $X^\top\beta$, i.e. $\mathbb{E}[\epsilon_i X_i^\top\beta] = \mathbb{E}[\epsilon_i]\mathbb{E}[X_i^\top\beta] = 0$. I want to check this assertion in logistic regression model.

In the logistic regression $\epsilon_i = y_i - p_i$. From the gradient $\partial L/\partial\theta = \sum_i (y_i - p_i)X_i = \sum_i X_i \cdot \epsilon_i$, I found that $\mathbb{E}[\beta^\top X_i^\top \epsilon_i] = \texttt{const} \cdot \beta^\top \nabla_\theta L = 0$ as long as the model is optimal.

However, if performing Ridge regression (with a coef λ), $\mathbb{E}[\epsilon_i X_i^\top\beta] \propto \lambda\beta^\top\beta > 0$ when the model is optimal.

I calculate the $\mathbb{E}[\epsilon_i X_i^\top\beta]$ value according to the result from my LR model and LR + Ridge model. The results are shown in Table 8.

Table 8: Error - Ridge

λ	$\mathbb{E}[\epsilon_i X_i^\top\beta]$
0	-1.565×10^{-4}
0.1	3.756×10^{-2}
0.01	5.367×10^{-3}

The result shows that without Ridge loss, the error is really uncorrelated to the input. But with Ridge loss, the error is a little positive correlated to the projected input $X^\top\beta$.

But for this characteristic of Ridge regression, I have not come up with a reasonable explanation.

10 The End

I have implemented several typical ML model in this project. For LR, LDA and Kernel LR, I have given some intuitive or theoretical analysis. For all the models, I have performed experiments to make sure that they work. And I have shown the visualization result for the HOG feature.

The most interesting analysis, I think, is about the distribution of $X^\top\beta$, and I explanation the relationship between the distribution and the sigmoid / logistic function, and then provide a reasonable explanation for logistic regression method.