2020 Pattern Recognition and Machine Learning Technical Report

Ruian He, 16307110216,

I. Task Description

BASED on the MNIST dataset[1], design and implement classifiers including linear & nonlinear SVM and MLPNN with two different error functions.

II. Data Description

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. [2]

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices.

A. Label File

The label file is like the following.The labels values are 0 to 9.

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
...	...	...	...
xxxx	unsigned byte	??	label

B. Image File

And the image file is as followed.Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...	...	...	...
xxxx	unsigned byte	??	pixel

III. Data Preprocessing

First of all, we uncompress the `.gz` files to get ubyte files described as above. And we use struct module in python to extract the magic number, the image number, the row number and col number. Then we read $num * row * col$ numbers stored in unsigned bytes from the file which is the data. We can visualize using `matpoltlib` module.

Fig. 1. Sample Digit

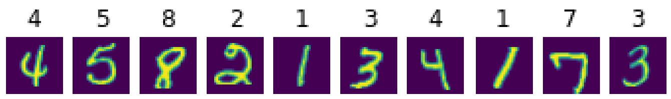
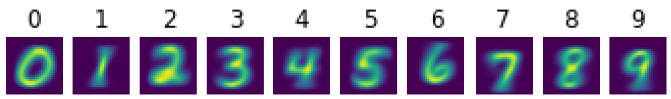


Fig. 2. Average Digit



We also can plot the average image for every digit, and get a general view over all training data.

Because we will run bayesian decision for $row * col$ features in a image, we also need to flat the matrix of the image to get a vector of $row * col$ length.

IV. Algorithm Introduction

A. Support Vector Machine(SVM)

1) Algorithm principle: The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. [3] We can define the hyperplane as follows:

$$y = w^T x + b$$

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. We define ρ_i as the distance of point i from the hyperplane.

$$\rho_i = \frac{1}{||w||} \cdot y_i(w^T x_i + b)$$

We introduce y_i in the distance because for positive examples $y_i = 1$ and we expect distance greater than a specific value $d_i = \frac{1}{||w||}(w^T x_i + b) \geq \delta$ and for negative examples $y_i = -1$ and and we expect distance $d_i = \leq -\delta$. We can unify this two presentations with ρ_i which y_i multiple $\frac{1}{||w||}(w^T x_i + b)$. And therefore $\rho_i \geq \delta$.

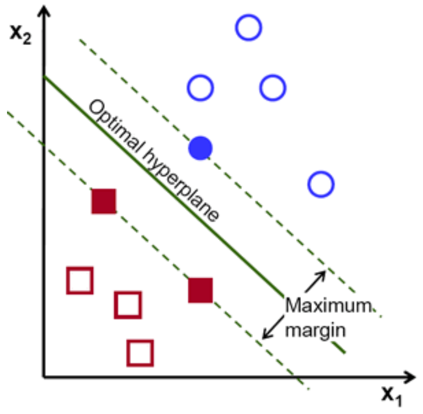
Our purpose is to maximize the minimum of ρ_i, i.e δ.

$$\max_{w,b} \delta$$

$$s.t. \frac{1}{||w||} \cdot y_i(w^T x_i + b) \geq \delta$$

[1]http://yann.lecun.com/exdb/mnist/.

Fig. 3. Support Vector Machine



Let $w' = \frac{w}{||w||\delta}$ and $b' = \frac{b}{||w||\delta}$, and our purpose becomes minimize $||w||$ so that $y_i(w'^T x + b') \geq 1$. And that is equivalent to minimize $\frac{1}{2}||w||^2$.

$$\min_{w,b} \frac{1}{2}||w||^2$$
$$s.t. y_i(w^T x_i + b) \geq 1$$

This is called the primal problem. It requires learning a large number of parameters in the w feature space.

The dual problem results in some beneficial properties that will aid in the computation, including the use of Kernel functions to solve non-linearly separable data.

The dual problem requires learning only the number of support vectors, which can be much fewer than the number of feature space dimensions. The dual problem is found by constructing the Lagrange, by combining both the objective function and the equality constraint function. The Lagrange formulation is

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$$

Here, we we define w and b as the primal variables and α_i as the dual variables. According to the Karush-Kuhn-Tucher(KKT) conditions, the formulation gets its optimal value when $\frac{\partial L(w,b,\alpha)}{\partial w} = \frac{\partial L(w,b,\alpha)}{\partial b} = 0$ and must fulfill the following conditions:

1) $\alpha_i \geq 0$
2) $y_i(w^T x_i + b) - 1) \geq 0$
3) $\alpha_i(y_i(w^T x_i + b) - 1) = 0$

Then we can get $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$ from the above conditions.Finally, the formulation becomes like the following formula, and can be solved using Sequential Minimal Optimization(SMO) algorithm.

$$\min_{\alpha_i} L(\alpha) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) - \sum_{i=1}^{n} \alpha_i$$
$$s.t. \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

In this formulation, $(x_i \cdot x_j)$ stands for the linear kernel function. We can replace this with other non-linear kernels like Radial Basis Function(RBF) to get better representation of features.

$$K(x_i, x_j) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

Algorithms such as the Perceptron, Logistic Regression, and Support Vector Machines were designed for binary classification and do not natively support classification tasks with more than two classes.

One approach for using binary classification algorithms for multi-classification problems is to split the multi-class classification dataset into multiple binary classification datasets and fit a binary classification model on each. Two different examples of this approach are the One-vs-Rest and One-vs-One strategies.

We choosed one-vs-One strategy which splits a multi-class classification into one binary classification problem per each pair of classes.

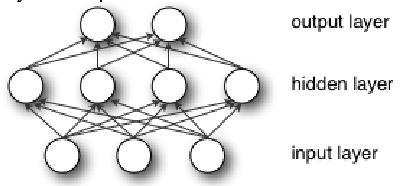
2) Algorithm implementation: The `sklearn` library provides the implementation of Support Vector Machine classifier, i.e SVC. And we can easily change the kernel function like `linear` and `rbf`.

Listing 1. SVC
```
from sklearn import svm
from sklearn.metrics import
    accuracy_score,f1_score

model = svm.SVC(kernel='linear')
model.fit(X_train,y_train)

y_pred = model.predict(X_test)
print("The␣model␣accuracy␣is:",
    accuracy_score(y_test,y_pred),
'f1_score␣is:',f1_score(y_test,y_pred,
    average='micro'))
```

B. MultiLayer Perceptron Neural Network(MLPNN)

1) Algorithm principle: A Multi-Layer Perceptron(MLP) can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation Φ. This transformation projects the input data into a space where it becomes linearly separable. This intermediate layer is referred to as a hidden layer. A single hidden layer is sufficient to make MLPs a universal approximator. [1]

Fig. 4. Multi-Layer Perceptron



For every hidden layer node z_i, there is a linear combination w of the features from the input layer x_i and a non-linear activation σ function to introduce complex representations. It is the same as in output layer y_i.

$$\begin{aligned} z_i = \sigma(net_i^{(1)}) &= \sigma(\sum_j w_{ij}^{(1)} x_j + b_i^1) \\ y_i = \sigma(net_i^{(2)}) &= \sigma(\sum_j w_{ij}^{(2)} z_j + b_i^2) \end{aligned}$$

Then, we calculate the loss function, the distance from predition $\hat{y}$ and the ground truth y, like cross entropy loss between a probability distribution $\hat{y}$ and a class truth y.

$$L(\hat{y}, y) = -\log(\frac{exp(\hat{y}[y])}{\sum_j exp(\hat{y}[j])})$$

Except for cross entropy loss, multiclass classification can also use multi-margin loss.

$$L(\hat{y}, y) = \frac{\sum_i \max(0, \delta - \hat{y}[y] + \hat{y}[i])}{\text{num\_class}}$$

Finally, we perform backpropagation to use the loss gradient to update perceptron parameters, i.e computing the gradient for each weight and bias according to the chain rule. Take the cross entropy loss for example.

$$\begin{aligned} \frac{\partial L}{\partial y_i} &= \frac{\exp(y_i)}{\sum_i \exp(y_i)} \\ \frac{\partial y_i}{\partial w_{ij}^2} &= \frac{\partial y_i}{\partial net_i^{(2)}} \cdot \frac{\partial net_i^{(2)}}{\partial w_{ij}^2} \\ &= \sigma'(net_i^{(2)}) z_j \\ \frac{\partial L}{\partial w_{ij}^2} &= \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_{ij}^2} \\ &= \frac{\exp(y_i)}{\sum_i \exp(y_i)} \cdot \sigma'(net_i^{(2)}) z_j \end{aligned}$$

When we update the parameters, some strategies are applied to get faster trainning speed and better coverge point. Adam is one of the best optimizer to do this.

Finally, after several times of traversing all dataset, i.e epochs, the loss of the training set and validation set stagnated at a lower point. And we take the parameter at the time to predict.

2) Algorithm implementation: We use Pytorch deep learning framework to build our model. There are several losses and optimizers to choose from. We take cross entropy loss and multi-margin loss and pass them to the model to get different results. And Adam is applied as the optimizer.

The implementation follows the Pytorch lightning template[2].

Listing 2. MultiLayer Perceptron

```
class LitClassifier(pl.LightningModule):
  def __init__(self, loss, hidden_dim
      =128, learning_rate=1e-3):
    super().__init__()
    self.save_hyperparameters()
    self.l1 = torch.nn.Linear(28 * 28,
        self.hparams.hidden_dim)
    self.l2 = torch.nn.Linear(self.
        hparams.hidden_dim, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = torch.relu(self.l1(x))
    x = torch.relu(self.l2(x))
    return x

  def training_step(self, batch,
      batch_idx):
    x, y = batch
    y_hat = self(x)
    loss = self.hparams.loss(y_hat, y)
    return loss

  def validation_step(self, batch,
      batch_idx):
    x, y = batch
    y_hat = self(x)
    loss = self.hparams.loss(y_hat, y)
    y_hat = torch.argmax(y_hat, dim=1)
    acc = pl.metrics.functional.
        classification.accuracy(y_hat,y)
    self.log_dict({'val_loss': loss,'
        val_accuracy': acc})

  def test_step(self, batch, batch_idx):
    x, y = batch
    y_hat = self(x)
    loss = self.hparams.loss(y_hat, y)
    y_hat = torch.argmax(y_hat, dim=1)
    f1 = pl.metrics.functional.
        classification.f1_score(y_hat,y)
    acc = pl.metrics.functional.
        classification.accuracy(y_hat,y)
    self.log_dict({'test_loss': loss,'
        f1_score':f1 ,'accuracy': acc})

  def configure_optimizers(self):
    return torch.optim.Adam(self.
        parameters(), lr=self.hparams.
        learning_rate)
```

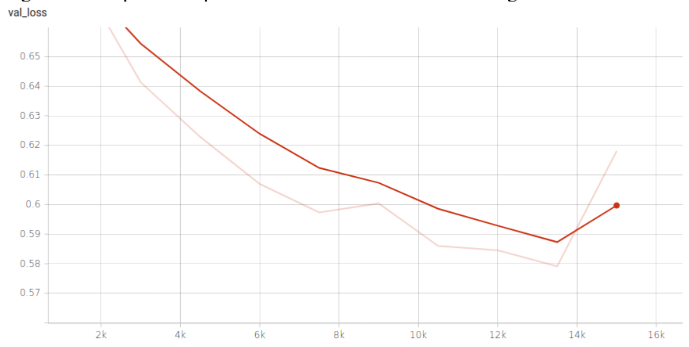
[2]https://github.com/PyTorchLightning/deep-learning-project-template/blob/master/project/lit_mnist.py

V. EXPERIMENTAL RESULTS AND ANALYSIS

We train different models on the same training set and test on the same test set but the sizes of training set differ because of too long training time for SVM models.

SVM need to solve optimization problem with precise method and one-vs-one strategy requires many SVM models built for every pair of classes. That cause huge repeated float calculation on CPU.

Fig. 5. Multiple Perceptron Validation Loss when training



For multilayer perceptron models, we trained 10 epochs on the whole training set accelerated by GPU. If the training set is too small, the model won't behave fairly well.

TABLE I
COMPARISON OF ACCURACY OF DIFFERENT MODELS

Model	Classification	Training Set	Accuracy
SVM with linear kernel	one-vs-one	10000	0.9172
SVM with rbf kernel	one-vs-one	10000	0.9594
MLP with ce loss	multiclass	60000	0.8421
MLP with margin loss	multiclass	60000	0.6244

We can learn from the table that rbf kernel have better representation of the data than linear model. With small data set, SVM perform better even with less data. The multi-margin loss have weak constraint for the classification, while cross entropy loss perform better.

REFERENCES

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.
[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[3] Z.H. Zhou. *Machine Learning*. Tsinghua University Press, 2016.