Image Classification

Lab4 | Special Topics in Information Systems

Names	ID
Abdelrhman Yasser	37
Amr Mohamed Fathy Hendy	46

Introduction

In this assignment we will practice putting together a simple image classification pipeline, based on the k-Nearest Neighbor or the SVM/Softmax classifier. The goals of this assignment are as follows:

1. understand the basic Image Classification pipeline and the data-driven approach (train/predict stages)
2. understand the train/val/test splits and the use of validation data for hyperparameter tuning.
3. develop proficiency in writing efficient vectorized code with numpy
4. implement and apply a k-Nearest Neighbor (kNN) classifier
5. implement and apply a Multiclass Support Vector Machine (SVM) classifier
6. implement and apply a Softmax classifier
7. implement and apply a Two layer neural network classifier
8. understand the differences and tradeoffs between these classifiers
9. get a basic understanding of performance improvements from using higher-level representations than raw pixels (e.g. color histograms, Histogram of Gradient (HOG) features)

We will discuss each of KNN, SVM, Softmax and Two layer neural network with their

1. Pseudocode
2. Tuning parameters and reporting best accuracy
3. How data is presented in the classifier
4. Reporting model accuracies and results

SVM

1) Pseudocode

- We can calculate the loss value using the svm loss function as following

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \qquad \left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)\right] + \lambda \|\vec{w}\|^2$$

  Where Li represent the loss existing in the training sample i , then we sum over all the training samples and average the result by dividing by the number of training examples. Also adding the regularization term to get the total loss value.

- We can calculate the gradient value of the loss function w.r.t y(i) for the training example i using the following rule

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)\right) x_i$$

  Then we sum over all the training examples and average the result by dividing by the number of training examples. Also adding the regularization term to get the total gradient value w.r.t y(i).

2) Tuning Parameters

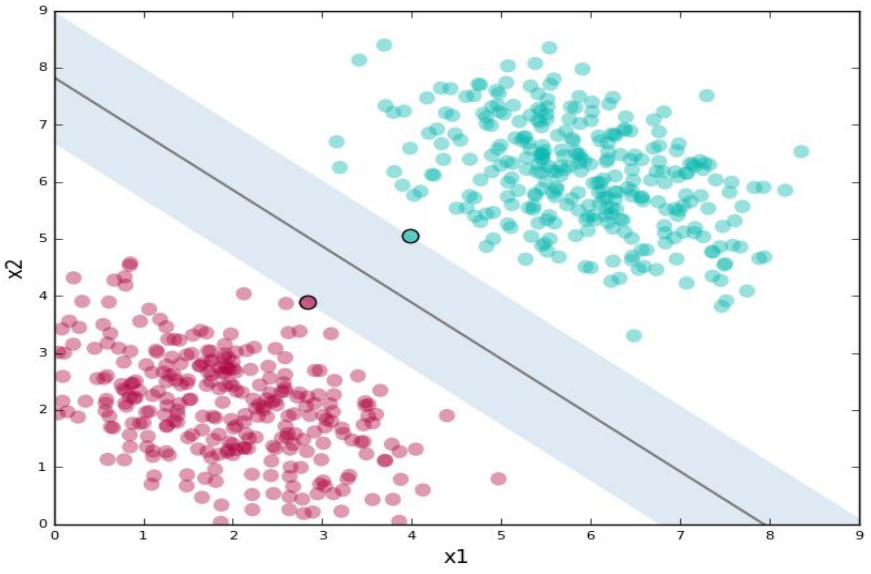
We used the validation set to tune the following hyperparameters

- **regularization strength** (trying 2.5e4, 5e4 and 5e5)
- **learning rate** (trying 1e-8, 1e-7, 1e-6, 5e-6, 1e-5 and 1e-4)

```
lr 1.000000e-08 reg 2.500000e+04 train accuracy: 0.269163 val accuracy: 0.277000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.336531 val accuracy: 0.345000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.305857 val accuracy: 0.321000
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.368306 val accuracy: 0.387000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.359857 val accuracy: 0.360000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.302224 val accuracy: 0.299000
lr 1.000000e-06 reg 2.500000e+04 train accuracy: 0.286306 val accuracy: 0.289000
lr 1.000000e-06 reg 5.000000e+04 train accuracy: 0.233061 val accuracy: 0.221000
lr 1.000000e-06 reg 5.000000e+05 train accuracy: 0.184714 val accuracy: 0.177000
lr 5.000000e-06 reg 2.500000e+04 train accuracy: 0.213816 val accuracy: 0.237000
lr 5.000000e-06 reg 5.000000e+04 train accuracy: 0.194041 val accuracy: 0.203000
lr 5.000000e-06 reg 5.000000e+05 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-05 reg 2.500000e+04 train accuracy: 0.136531 val accuracy: 0.134000
lr 1.000000e-05 reg 5.000000e+04 train accuracy: 0.159551 val accuracy: 0.155000
lr 1.000000e-05 reg 5.000000e+05 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-04 reg 2.500000e+04 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-04 reg 5.000000e+04 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-04 reg 5.000000e+05 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.387000
```

3) Data Representation

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.



After finishing training we can save only the support vectors to be used to get the best hyperplane which is used in the prediction.

4) Model Accuracy

The best model we got after tuning the parameters using the validation set has

- Training accuracy 0.383531
- Validation accuracy 0.3920
- Testing accuracy 0.1030

KNN

1) Pseudocode

- For computing distances using two loops, it is very trivial just iterating over all the testing samples then iterating over the training examples and calculate the Euclidean distance between each pair

```
dists[i, j] = np.sqrt(np.sum((self.X_train[j] - X[i])**2))
```

- For computing distances using single loop, it is quite easy also, we will easily iterate over the testing samples only and subtract each testing sample from all the training examples, So now we have a matrix of the resultant after subtraction then we will square it and sum each row then take the square root for the sum vector, we will get the final distances matrix.

```
    for i in range(num_test):
      dists[i, :] = np.sqrt(np.sum((self.X_train - X[i, :])**2, axis=1))
```

- For computing distances using no loops, it is some tricky. The “trick” is to expand the l2 distance formula. For each vector x and y, the l2 distance between them can be expressed as:

$$(x-y)^2 = x^2 + y^2 - 2xy$$

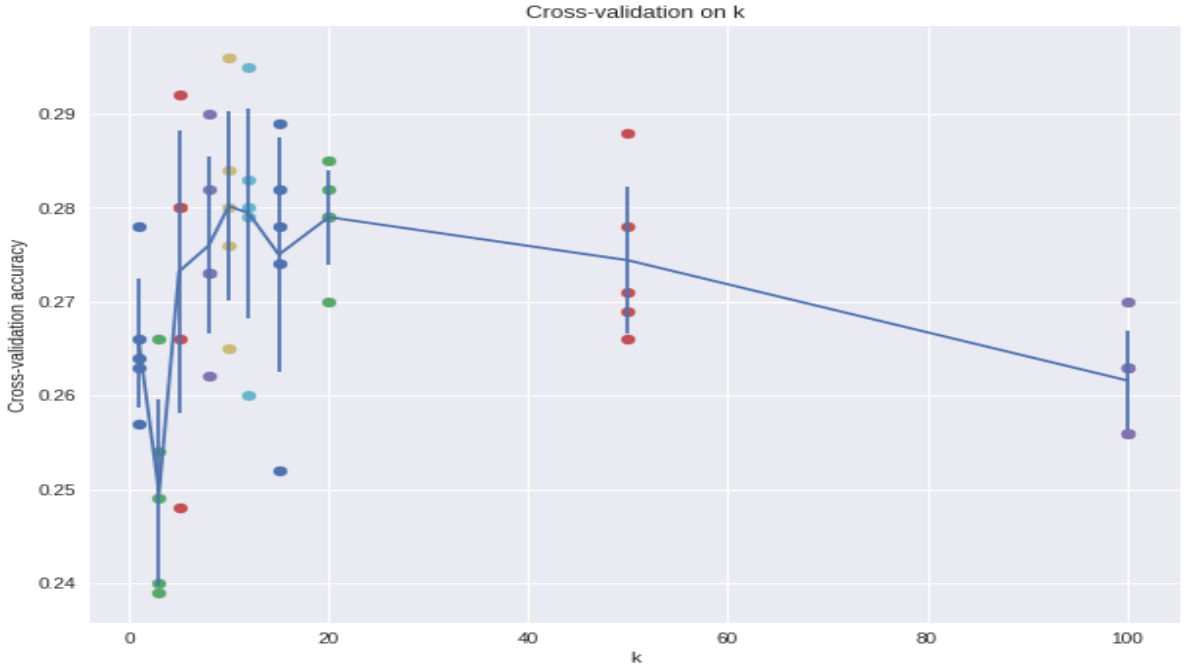
To vectorize efficiently, we need to express this operation for ALL the vectors *at once* in numpy. The first two terms are easy ， just take the l2 norm of every row in the matrices X and X_train. The last term can be expressed as a matrix multiply between X and transpose(X_train). Numpy can do all of these things super efficiently.

```
dists = np.sqrt(-2 * np.dot(X, self.X_train.T) + \
                np.sum(self.X_train**2, axis=1) + \
                np.sum(X**2, axis=1)[:, np.newaxis])
```

2) Tuning Parameters

We used the validation set (5 folds) to tune **k parameter** for these values **[1, 3, 5, 8, 10, 12, 15, 20, 50, 100]**.
We got the following validation accuracies:

K	Fold1 Accuracy	Fold2 Accuracy	Fold3 Accuracy	Fold4 Accuracy	Fold5 Accuracy
1	0.263000	0.257000	0.264000	0.278000	0.266000
3	0.239000	0.249000	0.240000	0.266000	0.254000
5	0.248000	0.266000	0.280000	0.292000	0.280000
8	0.262000	0.282000	0.273000	0.290000	0.273000
10	0.265000	0.296000	0.276000	0.284000	0.280000
12	0.260000	0.295000	0.279000	0.283000	0.280000
15	0.252000	0.289000	0.278000	0.282000	0.27400
20	0.270000	0.279000	0.279000	0.282000	0.285000
50	0.271000	0.288000	0.278000	0.269000	0.266000
100	0.256000	0.270000	0.263000	0.256000	0.263000



The best model got accuracy of 0.2800 on validation samples.

3) Data Representation

In KNN the data is represented by points in D dimensions, KNN must keep all the data points to be used in calculating the distances with the new point to be predicted.

4) Model Accuracy

The best model we got after tuning the parameters using the validation set has

- Validation accuracy 0.28000
- Testing accuracy 0.274000

Two Layer NN

1) Pseudocode

- Model is two layer NN :
  Input Layer → First layer → relu → second layer → sigmoid → Output

- First we calculate forward pass :

```
# first layer activations
z1 = np.dot(X, W1) + b1
h1 = self.relu(z1)
#second layer activations
z2 = np.dot(h1, W2) + b2
h2 = self.softmax(z2)
```

- Then we calculate cross entropy loss using following equations :

  Information theory view. The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as:

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

- Then gradients with respect to loss is computed during backward propagation:
  - $\frac{dl}{dh2} = h2 - y$
  - $\frac{dl}{dz2} = \frac{dl}{dh2} * \frac{dh2}{dz2}$
  - $\frac{dh2}{dz2} = h2 * (1 - h2)$
  - $\frac{dl}{dw2} = \frac{dl}{dz2} * \frac{dz2}{dw2}$
  - $\frac{dz2}{dw2} = h1$
  - $\frac{dl}{db2} = \frac{dl}{dz2} * \frac{dz2}{db2}$
  - $\frac{dl}{dh1} = \frac{dl}{dz2} * \frac{dz2}{dh1}$
  - $\frac{dl}{dz1} = \frac{dl}{dh1} * \frac{dh1}{dz1}$
  - $\frac{dh1}{dz1} = 1\,(z1 > 0)$

- $\frac{dl}{dw1} = \frac{dl}{dz1} * \frac{dz1}{dw1}$
- $\frac{dz1}{dw1} = X$
- $\frac{dl}{dh1} = \frac{dl}{dz1} * \frac{dz1}{dh1}$
- $\frac{dz}{dh1} = 1$

2) Tuning Parameters

We used the validation set to tune the following hyper parameters

- **regularization strength** (trying 1e-2, 1e-3, 1e-4)
- **learning rate** (trying 1e-3, 5e-3, 1e-2, 5e-2)
- **hidden sizes** (trying 128, 256, 512)
- **batch sizes** (trying 50, 100, 200)

```
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 5.000000e+01 and r: 1.000000e-02, valid accuracy is: 0.425000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 1.000000e+02 and r: 1.000000e-02, valid accuracy is: 0.452000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 2.000000e+02 and r: 1.000000e-02, valid accuracy is: 0.465000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 5.000000e+01 and r: 1.000000e-03, valid accuracy is: 0.449000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 1.000000e+02 and r: 1.000000e-03, valid accuracy is: 0.446000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 2.000000e+02 and r: 1.000000e-03, valid accuracy is: 0.486000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 5.000000e+01 and r: 1.000000e-04, valid accuracy is: 0.420000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 1.000000e+02 and r: 1.000000e-04, valid accuracy is: 0.464000
for hs: 1.280000e+02, lr: 1.000000e-03 , bs: 2.000000e+02 and r: 1.000000e-04, valid accuracy is: 0.485000
```

3) Model Accuracy

The best model we got after tuning the parameters using the validation set has

- Validation accuracy 0.490
- Testing accuracy 0.487

Soft max

1) Pseudocode

- Calculate softmax loss as follows :

Information theory view. The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as:

$$H(p,q) = -\sum_x p(x) \log q(x)$$

- Add regularization to loss to prevent overfitting :

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

- Then gradients with respect to loss is computed during backward propagation:
  - $\frac{dl}{dz1} = h1 - y$
  - $\frac{dl}{dw1} = \frac{dl}{dz1} * \frac{dz1}{dw1}$
  - $\frac{dz1}{dw1} = X$

2) Tuning Parameters

We used the validation set to tune the following hyper parameters

- **regularization strength** (trying 2.5e4, 5e4)
- **learning rate** (trying 1e-7, 5e-7)

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.328796 val accuracy: 0.346000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.308245 val accuracy: 0.323000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.315245 val accuracy: 0.339000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.301612 val accuracy: 0.326000
best validation accuracy achieved during cross-validation: 0.346000
```

3) Model Accuracy

The best model we got after tuning the parameters using the validation set has

- Validation accuracy 0.346
- Testing accuracy 0.331