Seung Jun Baek

1 Outline

In this assignment, you are asked to design three kinds of classifiers: (1) k-nearest neighbors, (2) SVM-loss classifier (3) softmax classifier.

2 Specification

In this assignment, you are asked to write a Python code for three classifiers:

- **k-Nearest Neighbor (kNN)**
- **SVM-loss classifier**
- **softmax classifier**

The dataset is based on randomly generated points in $\mathbb{R}^2$ according to Gaussian distribution. Each class will have its mean vector and covariance matrix. An example plot of the synthesized dataset is shown in Fig. 1.

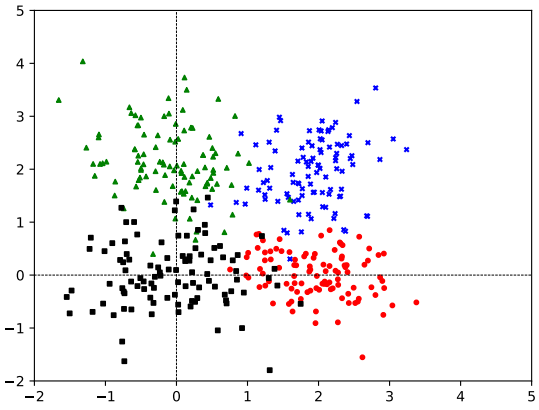

Figure 1: Plot of sample training dataset points with 4 classes.

The module for data generation is given in the file `data_generator.py`. You will need to put this file in the same directory as `hw2.py`. You will need to modify `hw2.py` file to implement the classifiers.

The module `data_generator.py` contains the following function:
`generate(number=100, seed=None, plot=True, dataset=True, num_class=3,sigma=1.0)`
The parameters are:

- `number`: the total number of data points to generate. This number will be divided into the number of classes in the dataset. For example, if there are 4 classes, the data points per class will be 100/4=25.
- `seed`: to control the seed for random number generator.
- `plot`: toggles the plotting of data points.
- `num_class`: number of classes in the dataset.
- `sigma`: Sigma parameter controls the degree of scattering of data points around the mean. Larger sigma means larger variance in the dataset, which will result in lower accuracy. Default value is 1.0. For default value of sigma=1.0, you will see the accuracy will be around 80–90% for all classifiers. When sigma=2.0, you will see the accuracy drops to 60–70%.
- returns `input_value`,`output_value` where `input_value` is the input dataset, a matrix of shape `number` by 2; `output_value` is the output dataset, which contains the labels from 0 to `num_class`-1, a list of size `number`.

2.1 kNN classifier

You are asked to design a k-nearest neighbors classifier. There is no function needed for training – only a function for testing is needed for kNN classifier. **You should use $L2$ distance (Euclidean distance) in this kNN classifier.** Implement the function
`knn_test(X_train, y_train, X_test, y_test, n_train_sample, n_test_sample, k)`

- `X_train`: input training dataset. A matrix (2-d numpy array) of shape `(n_train_sample, 2)`.
- `y_train`: output training dataset. A vector (1-d numpy array) of shape `(n_train_sample,)`. Contains ground truth data labels.
- `X_test`: input test dataset. A matrix (2-d numpy array) of shape `(n_test_sample, 2)`
- `y_test`: output test dataset. A vector (1-d numpy array) of shape `(n_test_sample,)`. Contains ground truth data labels.
- `n_train_sample`: integer, the size of training dataset
- `n_test_sample`: integer, the size of test dataset
- `k`: integer, k parameter for kNN. Note that k should be either 1 or 3 in this assignment.

- returns a number between 0 and 1, which is the accuracy. The accuracy is defined as

$$\frac{(\text{the number of correctly classified labels from test data})}{(\text{n\_test\_sample})}$$

If necessary, you can define and use whatever extra functions that is needed to implement this function.

2.1.1 implementation hints

The below are hints that you can use to efficiently implement (that is, speeding up the run of) the classifier. You may use other methods to implement the function – but it is strongly recommended that the below methods be used, because when the dataset is large, the speed-up may become critical.

Hint 1: Here you need to find the $L2$ distance between every training data point and every testing data point. Of course, you can run code something like

```
for i in range(n_train_sample):
        for j in range(n_test_sample):
                # finding L2 distance between X_train[i] and X_test[i]
```

However, this code can be really slow! What can be done to speed up the implementation?

Suppose there is a vector u and v. It is known that

$$\|u - v\|_2^2 = \|u\|_2^2 + \|v\|_2^2 - 2u^T v$$

A similar concept can be used when u and v are training and test datasets (i.e., they are matrices) respectively. For details, see link
`https://medium.com/@souravdey/l2-distance-matrix-vectorization-trick-26aa3247ac6c`.

Hint 2: you can use `numpy.argsort` to retrieve the indices of sorted vector. Also `scipy.stats.mode` to find the most frequently occurring elements from a vector.

2.2 SVM classifier

You are asked to design a softmax classifier. You need to complete the function
`svm_loss(Wb, x, y, num_class, n, feat_dim)`

- `n`: dataset size.
- `num_class`: the number of classes to classify
- `feat_dim`: the dimension of input data. In this assignment, this value will be set to 2.
- `x`: input data. a matrix of shape `(n,feat_dim)`
- `y`: output data. a vector of shape `(n,)`. Contains ground truth labels, that is, a number ranging from 0 to `num_class`-1.

- `Wb`: a vector of shape `(num_class x feat_dim + num_class,)`. The first `num_class*feat_dim` elements of the vector are the parameters for W which is `num_class` by `feat_dim` matrix. The last `num_class` elements of the vector are the parameters for b, the bias.

  As a linear classifier, if you want to find the linear scores out of the input dataset, you can use the following code:

```
Wb = np.reshape(Wb, (-1, 1))
b = Wb[-num_class:]
W = np.reshape(Wb[range(num_class * feat_dim)], (num_class, feat_dim))
x=np.reshape(x.T, (-1, n))

# this will give you a score matrix s of size (num_class)-by-(n)
# the i-th column vector of s will be
# the score vector of size (num_class)-by-1, for the i-th input data point
# performing s=Wx+b

s=W@x+b
```

- returns the SVM loss (should be nonnegative scalar), averaged over the dataset.

If necessary, you can define and use whatever extra functions that is needed to implement this function.

2.3 softmax classifier

You are asked to design a softmax classifier. You need to complete the function
`cross_entropy_softmax_loss(Wb, x, y, num_class, n, feat_dim)`

- `n`: dataset size.
- `num_class`: the number of classes to classify
- `feat_dim`: the dimension of input data. In this assignment, this value will be set to 2.
- `x`: input data. a matrix of shape `(n,feat_dim)`
- `y`: output data. a vector of shape `(n,)`. Contains ground truth labels, that is, a number ranging from 0 to `num_class`-1.
- `Wb`: a vector of shape `(num_class x feat_dim + num_class,)`. The first `num_class*feat_dim` elements of the vector are the parameters for W which is `num_class` by `feat_dim` matrix. The last `num_class` elements of the vector are the parameters for b, the bias.

  As a linear classifier, if you want to find the linear scores out of the input dataset, you can use the following code:

```
Wb = np.reshape(Wb, (-1, 1))
b = Wb[-num_class:]
W = np.reshape(Wb[range(num_class * feat_dim)], (num_class, feat_dim))
x=np.reshape(x.T, (-1, n))

# this will give you a score matrix s of size (num_class)-by-(n)
# the i-th column vector of s will be
# the score vector of size (num_class)-by-1, for the i-th input data point
# performing s=Wx+b

s=W@x+b
```

- returns the cross-entropy loss (should be nonnegative scalar), averaged over the dataset.

If necessary, you can define and use whatever extra functions that is needed to implement this function.

3 What to submit

You will be given `data_generator.py` and `hw2.py`. Place them on the same directory, and run `hw2.py` with the properly implemented functions.

- Submit a modified Python file `hw2.py`. The requirements are
  1. Implement the function `knn_test`.
  2. Implement the function `svm_loss`.
  3. Implement the function `cross_entropy_softmax_loss`.
- Upload your `hw1.py` file at Blackboard before deadline. (Please submit the file in time, no late submission will be accepted).

4 How to test your module

Run your completed code `hw2.py`. At the output console you will see something like, for kNN classifier

```
number of classes:  4  sigma for data scatter: 1.2
generating training data
400 data points generated. Seed is random.
generating test data
100 data points generated. Seed is random.
testing kNN classifier...
accuracy of kNN loss:  81.0 % for k value of  1
```

To change the classifier, modify `classifiers` variable in `hw2.py`. You can modify `num_class` variable and `sigma` for different results. Example results for SVM classifier.

```
number of classes:  3  sigma for data scatter: 1.5
generating training data
300 data points generated. Seed is random.
generating test data
60 data points generated. Seed is random.
training SVM classifier...
testing SVM classifier...
accuracy of SVM loss:  88.33333333333333 %
```

Example results for softmax classifier.

```
number of classes:  4  sigma for data scatter: 2.0
generating training data
400 data points generated. Seed is random.
generating test data
100 data points generated. Seed is random.
training softmax classifier...
testing softmax classifier...
accuracy of softmax loss:  75.0 %
```

The errors of the classifier may vary. However, with `num_class` equal to 4 and `sigma` equal to 1.0, your accuracy should be around 90%.

5 Grading

We will test your classifiers with `num_class` equal to 4 and `sigma` equal to 1.0 and k to 3.

- 5 points for each of your module (kNN, SVM, softmax) works correctly. Specifically, if your accuracy is more than 85%. So the maximum point is 15 points total.
- 2 points for each of your module does not work correctly.
- 0 point if you do not submit the file by deadline.

In the blackboard, you can upload your file as many times as you like, before the deadline. The last uploaded file will be used for grading. After deadline, the submission menu will be closed and you will not be able to make submission.