2020 Pattern Recognition and Machine Learning Technical Report

Ruian He, 16307110216,

I. TASK DESCRIPTION

IMPLEMENTATION of the k-nearest neighbor (k-NN) algorithm by yourself and Using the k-NN for classification based on MNIST dataset available at mnist website[1].

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. [1]

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

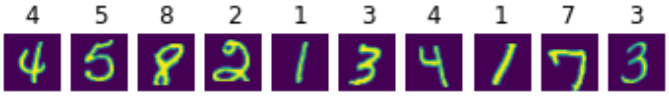
III. DATA PREPROCESSING

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.

Because we will run knn algorithm 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.

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

Fig. 1. Sample Digit



IV. ALGORITHM INTRODUCTION

A. Algorithm principle

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method proposed by Thomas Cover used for classification and regression.In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. [2]

For specified train set and test set, we can easily calculate the distance between test point and every point in training set.We know in a large space the distance d_{ij} is as followed and k is the index of features.

$$d_{ij} = \sqrt{\sum_{k=1}^{784}(x_{i,k} - x_{j,k})^2}$$
$$= \sqrt{\sum_{k=1}^{784}(x_{i,k})^2 + \sum_{k=1}^{784}(x_{j,k})^2 - \sum_{k=1}^{784} 2x_{i,k}x_{j,k}}$$

B. Algorithm implementation

For training, we simply store all samples in the model. When predicting we calculate the distance between target point and all sample points.

We use `argsort` function in numpy module to find the points with the smallest k distance.The `argsort` function returns the indices that would sort an array. And use `argmax` and `bincount` function to find the label with the most votes from its neighbors. The `bincount` fuction count number of occurrences of each value in array of non-negative in.ts

Listing 1. K Nearest Neighbor

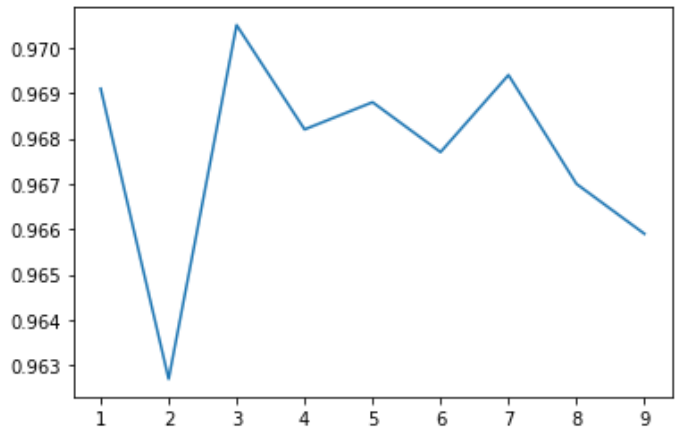
```
class KNearestNeighbor(object):
  def __init__(self):
    self.pre_X = None
    self.pre_dists = None
```

```
def train(self, X, y):
  self.X_train = X
  self.y_train = y

def predict(self, X, k=1):
  if (self.pre_X != X).any():
    self.pre_X = X
    self.pre_dists = self.
        compute_distances(X)
  return self.predict_labels(self.
     pre_dists, k=k)

def compute_distances(self, X):
  num_test = X.shape[0]
  num_train = self.X_train.shape[0]
  dists = np.zeros((num_test, num_train
     ))
  dists = np.sqrt((self.X_train.dot(X.T
     )*(-2)+np.sum(np.square(X),axis=1)
     ).T+np.sum(np.square(self.X_train)
     ,axis=1))
  return dists

def predict_labels(self, dists, k=1):
  num_test = dists.shape[0]
  y_pred = np.zeros(num_test)
  for i in range(num_test):
    closest_y = []
    closest_y=self.y_train[np.argsort(
        dists[i,:])][:k]
    y_pred[i]=np.argmax(np.bincount(
        closest_y))
  return y_pred
```

V. EXPERIMENTAL RESULTS AND ANALYSIS

Now we get the model and the data,then we can start training and testing.We choosed different k for knn algorithm and run the prediction.The following graph shows the variation of accuracy between different k from 1 to 9.

Fig. 2. Selection of k



We can get from the graph that it's not true that the bigger the k, the better the result. Among all the values of k, 3 get the best result and from 7 to 9 the accuracy is dropping as the k increases. Finally, the best result is the same as K-nearest-neighbors, Euclidean (L2) model with 3.09% error rate from Kenneth Wilder, U. Chicago. [1]

REFERENCES

[1] 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.

[2] Z.H. Zhou. *Machine Learning*. Tsinghua University Press, 2016.