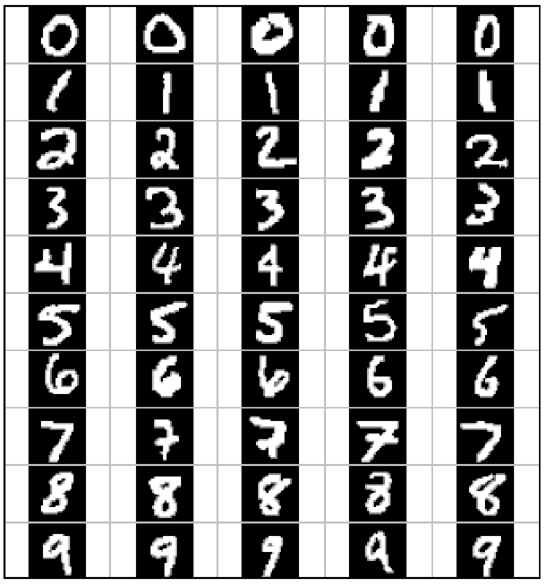
Project 3: Classification


Which digit?

Introduction

In this project, you will design four classifiers: a KNN classifier, a perceptron classifier, a SVM classifier, and a own-designed classifier. We will test these learning algorithms the handwritten digit images (MNIST) dataset.

Optical character recognition (OCR) is the task of extracting text from image sources. MNIST data set on which you will run your classifiers is a collection of handwritten numerical digits (0-9). There are systems that can perform with over 99% classification accuracy (deep learning methods).

Pre-requirement and Third-party python packages:

(0) At least 400MB disk space and 800MB memory is needed to complete this project.

(1) Python packages that should be installed:

`numpy`

`skimage`

`sklearn`

We recommend you use `Anaconda`, a python installation with hundreds of python packages integrated.

You can find the download link on the official website or on the following link:

http://ml.cs.tsinghua.edu.cn/~shifeng/download.html

(2) Python package that you should be familiar with:

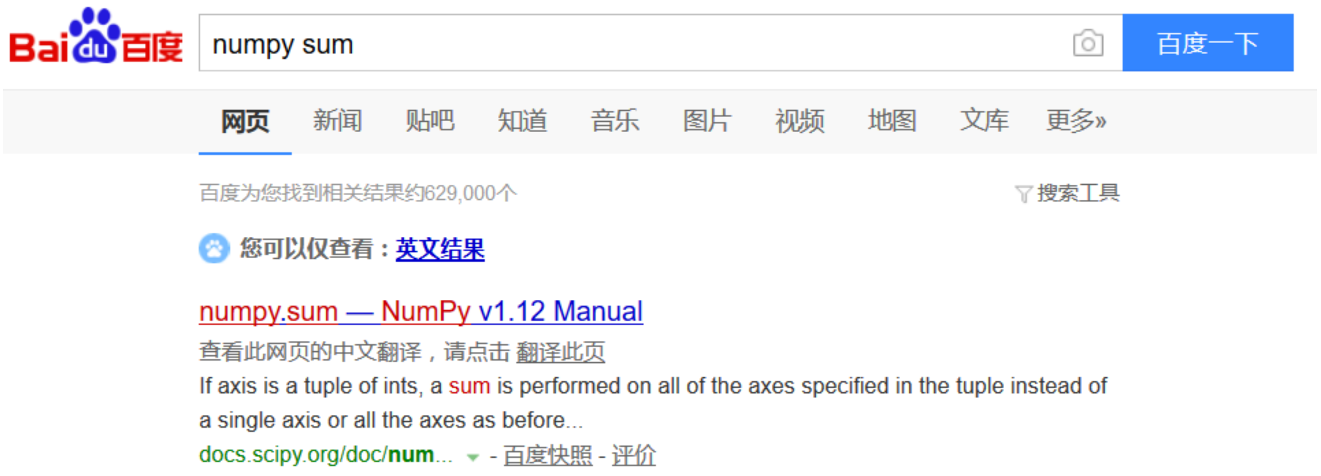
`numpy`

`sklearn`

Numpy is widely used in scientific computation, and it appears almost everywhere in our project. You can find numpy tutorial on the e-learning website (in ipython notebook format) or on the official website:

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

You can easily find the API documentation at the website above, or simply type "numpy <the API function you want to know>" on Baidu. Just like this:



百度为您找到相关结果约629,000个

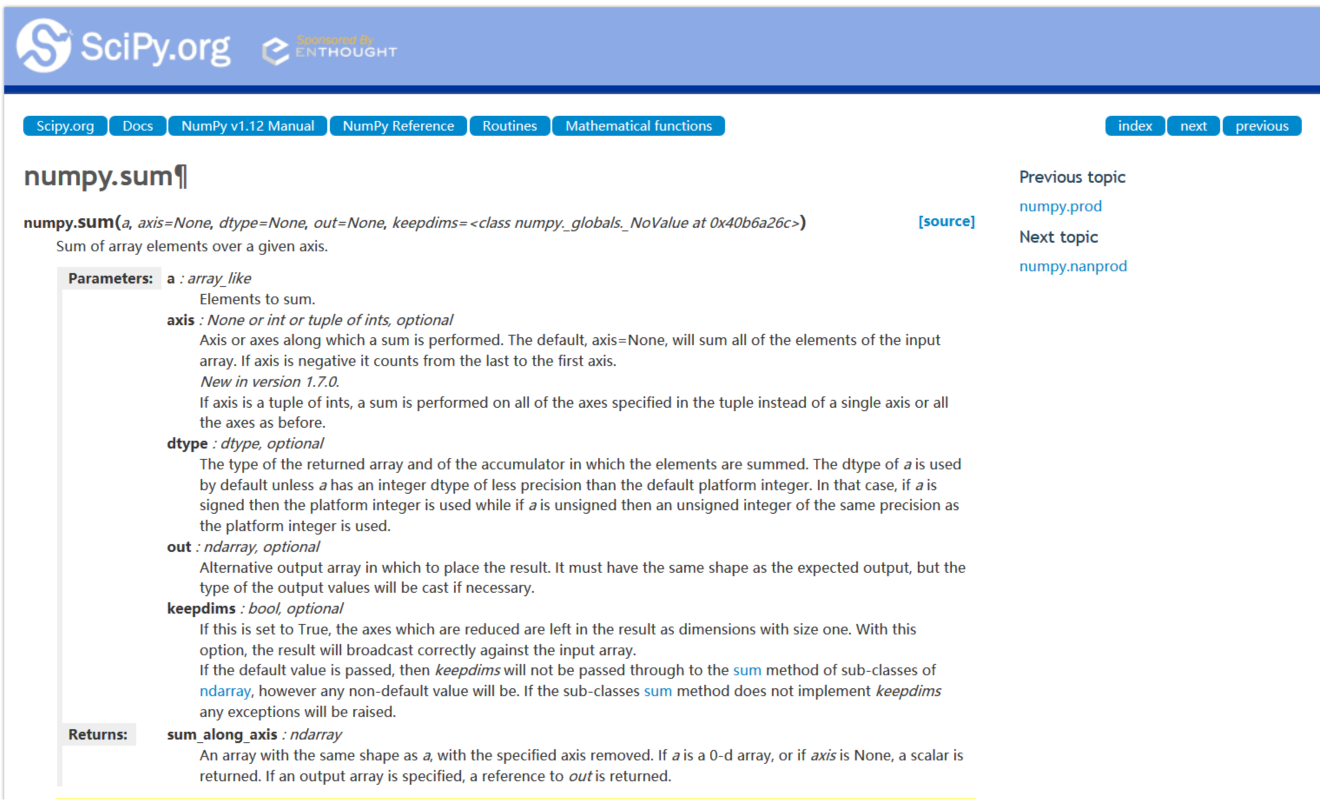
您可以仅查看：英文结果

numpy.sum — NumPy v1.12 Manual

查看此网页的中文翻译，请点击 翻译此页

If axis is a tuple of ints, a sum is performed on all of the axes specified in the tuple instead of a single axis or all the axes as before...

docs.scipy.org/doc/num... - 百度快照 - 评价



numpy.sum¶

numpy.**sum**(*a*, *axis=None*, *dtype=None*, *out=None*, *keepdims=<class numpy._globals._NoValue at 0x40b6a26c>*) [source]

Sum of array elements over a given axis.

Parameters: **a** : *array_like*
Elements to sum.

axis : *None or int or tuple of ints, optional*
Axis or axes along which a sum is performed. The default, axis=None, will sum all of the elements of the input array. If axis is negative it counts from the last to the first axis.
New in version 1.7.0.
If axis is a tuple of ints, a sum is performed on all of the axes specified in the tuple instead of a single axis or all the axes as before.

dtype : *dtype, optional*
The type of the returned array and of the accumulator in which the elements are summed. The dtype of *a* is used by default unless *a* has an integer dtype of less precision than the default platform integer. In that case, if *a* is signed then the platform integer is used while if *a* is unsigned then an unsigned integer of the same precision as the platform integer is used.

out : *ndarray, optional*
Alternative output array in which to place the result. It must have the same shape as the expected output, but the type of the output values will be cast if necessary.

keepdims : *bool, optional*
If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.
If the default value is passed, then *keepdims* will not be passed through to the sum method of sub-classes of ndarray, however any non-default value will be. If the sub-classes sum method does not implement *keepdims* any exceptions will be raised.

Returns: **sum_along_axis** : *ndarray*
An array with the same shape as *a*, with the specified axis removed. If *a* is a 0-d array, or if *axis* is None, a scalar is returned. If an output array is specified, a reference to *out* is returned.

Previous topic
numpy.prod
Next topic
numpy.nanprod

`sklearn` is a python package for machine learning with lots of popular machine learning algorithms. It should be used in SOME project. You can find API in the following page:

official tutorial

http://scikit-learn.org/stable/user_guide.html

sklearn tutorial in Chinese

http://cwiki.apachecn.org/display/sklearn/Index

(3) Python packages that **SHOULD NOT BE USED** in **SOME** tasks (or **you will not pass the autograder**):

`sklearn`

The code for this project includes the following files and data, available on the e-learning website.

Fills you will edit:

`classifiers.py`: The location where you will write your KNN/Perceptron/SVM classifier;

`answers.py`: Answers to Question 5 goes here.

Files you should be NOT edit:

`dataClassifier.py`: The wrapper code that will call your classifiers.

`featureExtractor.py`: The location where there are some methods for feature extractor.

`classificationMethod.py`: Abstract super class for the classifiers you will write. (You **should** read this file carefully to see how the infrastructure is set up.)

What to submit:

You will fill in portions of `classifier.py, answers.py` (only) during the assignment, and submit them.

Evaluation:

Your code will be autograded for technical correctness. Please **DO NOT** change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Academic Dishonesty:

We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. Instead, contact the course staff if you are having trouble.

Getting Started

To try out the classification pipeline, run `dataClassifier.py` from the command line.

```
python dataClassifier.py -c lr
```

This will classify the digit data using the default classifier (Linear Regression Classifier), and achieve about 78% test accuracy. Linear Regression Classifier is trained with linear regression. Consider training data $\mathrm{X} = [\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n}] \in \mathrm{R}^{m\times n}$ and label $\mathrm{Y} = [\mathrm{y}_1, \mathrm{y}_2, \dots, \mathrm{y}_\mathrm{n}] \in \mathrm{R}^{l\times n}$, $\mathrm{W} \in \mathrm{R}^{l\times m}$ is trained by optimizing

$$\min \|Y - W^T X\|_F^2 + \frac{\lambda}{2}\|W\|_F^2$$

Thus $\mathrm{W} \in \mathrm{R}^{m\times l}$ can be computed as follows:

$$W = (XX^T + \lambda I)^{-1} XY^T$$

For a given feature vector x, we score each class with:

$$\text{score}(x, y) = \|y - W^T x\|^2$$

And the predicted class of given feature vector x is

$$y' = \arg\min_{y''} \text{score}(x, y'')$$

For MNIST data set, in which each datum is a 28*28 gray-scale digit image, we consider the value of each pixel as the features of data, i.e. each image has the feature with dimension $m = 784 (= 28 \times 28)$. We use the one-hot encoding to construct y, thus for MNIST with 10 labels, $l = 10$. All features and labels are in numpy format.

As usual, you can learn more about the possible command line options by running:

```
python dataClassifier.py -h
```

Implementation Note: you can find the implementation of Linear Regression Classifier at `LinearRegressionClassifier.train/classify` in `classifiers.py`. A lot of numpy API functions is used in this class. You may find how to use numpy here.

Question 1: KNN classifier (4 points)

NOTE: sklearn SHOULD NOT BE USED in this task, OR you will not pass the autograder.

In this question, you will implement a simple KNN classifier. The features for KNN are all normalized, and the similarity of x, y is defined as $\text{sim}(x, y) = x^T y$.

Fill in the `KNNClassifier.classify` method in `classifiers.py`. Test 5-NN algorithm with 5000 training data, 1000 validation data and 1000 test data. Run the code with

```
python dataClassifier.py –c knn –n 5
```

Grading (Expected accuracy: 93%):

Points	Test Accuracy
1	>= 50%
2	>= 70%
3	>= 80%
4	>= 88%

Algorithm:

You may find KNN classifier algorithms on the slides provided by the teacher.

Question 2: Perceptron (or Softmax Regression) (5 points)

NOTE: sklearn SHOULD NOT BE USED in this task, OR you will not pass the autograder.

In this question, you will implement a perception (or it can also be called softmax regression) classifier. We use Batch Stochastic Gradient Descent (SGD) to train the classifier. The algorithm is shown below.

Fill in the `PerceptronClassifier.train` in `classifiers.py`. Test perceptron algorithm with 5000 training data, 1000 validation data and 1000 test data. Run the code with

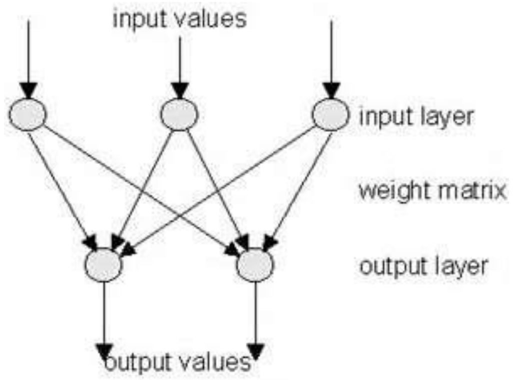
```
python dataClassifier.py –c perceptron
```

Grading (Expected accuracy: 87%)

Points	Test Accuracy
1	>= 40%
2	>= 60%
3	>= 70%
4	>= 78%
5	>= 83%

Algorithm:

Perceptron is shown as follows:



Given data $x \in \mathrm{R}^m$, then the output ($t \in \mathrm{R}^l$) is determined by the following function:

$$\underline{t = f(\mathrm{x}) = g\ (W^T x + b) = [f_1(x), \dots, f_l(x)]^T}$$

$$f_i(x) = g_i(w_i^T x + b_i)$$

where $\mathrm{W} = [\mathrm{w}_1, \dots, \mathrm{w}_l] \in \mathrm{R}^{m\times l}, b = [b_1, \dots, b_l] \in R^l, g$ is a non-linear function.

Suppose $f(\mathrm{x})$ can be defined as follows:

$$f_i(\mathrm{x}) = \frac{e^{w_i^T x + b_i}}{\sum_{j=1}^{l} e^{w_j^T x + b_j}}$$

Then the output of $f(x)$ can be regarded as the following multinomial distribution:

$$p(y = i|x) = f_i(\mathrm{x}) = \frac{e^{w_i^T x + b_i}}{\sum_{j=1}^{l} e^{w_j^T x + b_j}}$$

For training data $(x_1, y_1), \dots, (x_n, y_n)$, it is expected that the joint probability $\prod_{i=1}^{n} p(y = y_i|x_i)$ should be maximized. Thus the objective is

$$\min_{W,b} L(W,b) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y = y_i|x_i)$$

To avoid over-fitting, L2-panelty is added in the objective to reduce the model complexity:

$$\min_{W,b} L(W,b) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y = y_i|x_i) + \frac{\lambda}{2}(\|W\|^2 + \|b\|^2)$$

Batch Stochastic Gradient Descent(SGD) is widely used for optimizing the perceptron. For mini-batch data $(x_1, y_1), \dots, (x_k, y_k)$ of size k, the parameters is updated by

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W L'(W^{(t)}, b^{(t)})|_{W^{(t)}}$$

$$b^{(t+1)} = b^{(t)} - \eta \nabla_b L'(W^{(t)}, b^{(t)})\ |_{b^{(t)}}$$

Where η is the learning rate (we set $\eta = 1$ in this project) and

$$L'(W,b) = -\frac{1}{k}\sum_{i=1}^{k} \log p(y = y_i|x_i) + \frac{\lambda}{2}(\|W\|^2 + \|b\|^2)$$

Thus W and b is updated as follows:

$$w_j^{(t+1)} = w_j^{(t)} - \eta\lambda w_j^{(t)} - \frac{\eta}{k}\sum_{i=1}^{k} \nabla_{w_j} - \log p(y = y_i|x_i)\,|_{w_j = w_j^{(t)}}$$

$$b_j^{(t+1)} = b_j^{(t)} - \eta\lambda b_j^{(t)} - \frac{\eta}{k}\sum_{i=1}^{k} \nabla_{b_j} - \log p(y = y_i|x_i)\,|_{b_j = b_j^{(t)}}$$

where

$$\nabla_{w_j} - \log p(y = y_i|x_i) = \begin{cases} p(y = j|x_i)x_i, & j \neq y_i \\ (p(y = j|x_i) - 1)x_i, & j = y_i \end{cases}$$

$$\nabla_{b_j} - \log p(y = y_i|x_i) = \begin{cases} p(y = j|x_i), & j \neq y_i \\ p(y = j|x_i) - 1, & j = y_i \end{cases}$$

By each iteration, we random sample k training pairs and update W, b by the above update rule. It should be noticed that $\boldsymbol{\lambda}$ **can be also called weight-decay** parameter.

Question 3: Visualizing the Weights (1 points)

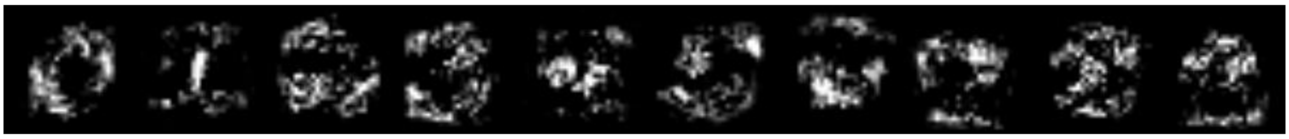
Run the code with

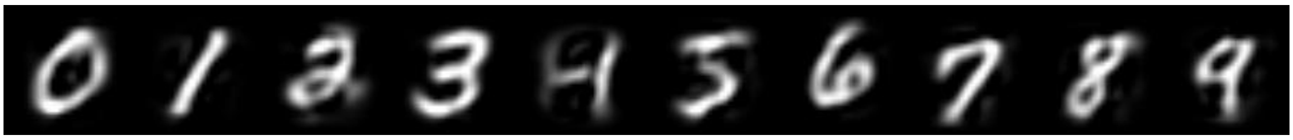
```
python dataClassifier -c perceptron -v
```

And find the image in visualize/weights.png. There are 10 sub-images looked like digits. Each sub-images represents each column of $\mathrm{W} = [\mathrm{w}_1, \ldots, \mathrm{w}_l] \in \mathrm{R}^{m\times l}, (m = 784, l = 10)$. For ease of visualization, the weights are resized to 28×28. The brighter pixel corresponds to higher value of weights in the corresponding position.

You are suggested to the analyse the visualization results of the weights. And answer the following question:

Which following images is most likely represent the weights learned by the perception?

(a) 

(b) 

Answer the question `answers.py` in the method `q3`, returning either `'a'` or `'b'`. **Note: the autograder will not grade this question.**

Question 4: SVM with sklearn (3 points)

sklearn is recommended to be used in the following tasks. You may find it helpful to perform ML algorithms with `sklearn`.

In this question, you should utilize the SVM algorithm in `sklearn` for classification. You should first of all be familiar with basic `sklearn` APIs, then implement a SVM algorithm with the following hyper-parameters:

`C=5;`

`kernel type: RBF(`$K(x,y) = \exp -\frac{(x-y)^2}{2\sigma^2}$`) with` $\sigma = 10$`;`

Fill in the `SVMClassifier.train, SVMClassifier.classify` in `classifiers.py`. Test SVM with 5000 training data, 1000 validation data and 1000 test data. Run the code with

```
python dataClassifier.py -c svm
```

It will take at most 3 minutes to learn.

Hint: You may find the SVM API in the following website page:

http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

More details in SVM can be found at

http://cwiki.apachecn.org/pages/viewpage.action?pageId=10031359

You should figure out the relationship between the `gamma` parameter in the API and σ in original RBF.

Grading (Expected accuracy: 93%):

2 points for correctly call the SVM API. Another 1 points for test accuracy >= 91%.

Algorithm (just for reference):

For two-class training data $(x_1, y_1), \dots, (x_n, y_n), y_i \in [-1,1]$, the dual problem of SVM is the following constrained quadratic programming:

$$\min_{\alpha} \frac{1}{2}\alpha^T A \alpha - \mathbf{1}^T \alpha$$

$$\text{s.t.} \begin{matrix} y^T\alpha = 0 \\ 0 \le \alpha_i \le C \end{matrix}$$

where $\alpha = [\alpha_1, \dots, \alpha_n]^T, y = [y_1, \dots, y_n]^T$, $\mathbf{1} = [1,1,\dots,1]^T \in R^n, A = \left(a_{ij}\right)_{n\times n} = \left(y_i y_j K(x_i, x_j)\right)_{n\times n}$ and $K(\cdot,\cdot)$ is the kernel function (e.g. RBF).

Suppose the result of the above constrained quadratic programming is $\alpha^* = [\alpha_1^*, \dots, \alpha_n^*]$, then the hyperplane is

$$f(x) = \sum_{\alpha_i^* \neq 0} y_i \alpha_i^* K(x_i, x) + b$$

$$b = y_i - \sum_{\alpha_j^* \neq 0} y_j \alpha_j^* K\left(x_j, x_i\right) \quad \text{for any} \quad 0 < \alpha_i^* < C$$

For multi-class SVM, we use one-vs-the-rest(one against all) algorithm, in which the ith $(1 \le i \le m)$

classifier $\mathrm{f_i}(x)$ is trained with the following training set:

(1) All the samples in the ith class as positive samples;

(2) All other samples as negative samples

And we use the following equation for classification:

$$y = \arg\max_{\mathrm{i}} f_i(x)$$

Question 5: Better Classification Accuracy (2 points+1 point bonus)

In this question, you will design a classification algorithm to get as best accuracy as you can. For this purpose, same algorithms in `sklearn` may be applied.

Fill in the `BestClassifier.train, BestClassifier.classify` in `classifiers.py`. Test the proposed algorithm with 5000 training data, 1000 validation data and 1000 test data. Run the code with

```
python dataClassifier –c best
```

Hint: you may find the following tips useful: (1) try other algorithms (Naïve Bayes, Decision Tree, Perceptron, SVM, Neural Network, etc.) (2) tune better hyper-parameters; (3) design better digit features; (4) analyse the wrong results and determine how to avoid that.

Grading:

Points	Test Accuracy
1	>= 94%
2	>= 95%
3(1 point bonus)	(Top 20% students)

Congratulations! You've finished with Projects 3.