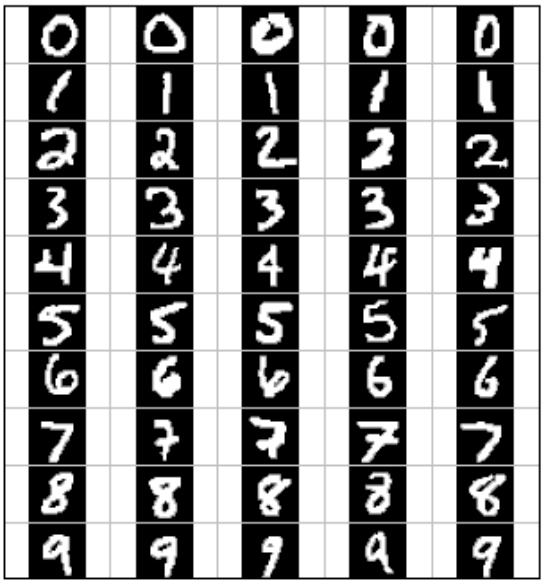
# Project 3: Classification and Learning



Which digit?


Which attitude?

## Introduction

In this project, you will design three classifiers: a KNN classifier, a perceptron classifier, a SVM classifier; and an unsupervised dimension reduction method: Principle Component Analysis (PCA). We will test these learning algorithms on two data sets: handwritten digit images (MNIST data set) and a set of text (SST data set).

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).

Sentiment analysis is a typical problem in text categorization. Given a document, we must determine whether the meaning this sentiment is positive/neutral/negative.

### *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`

`scipy`

`skimage`

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

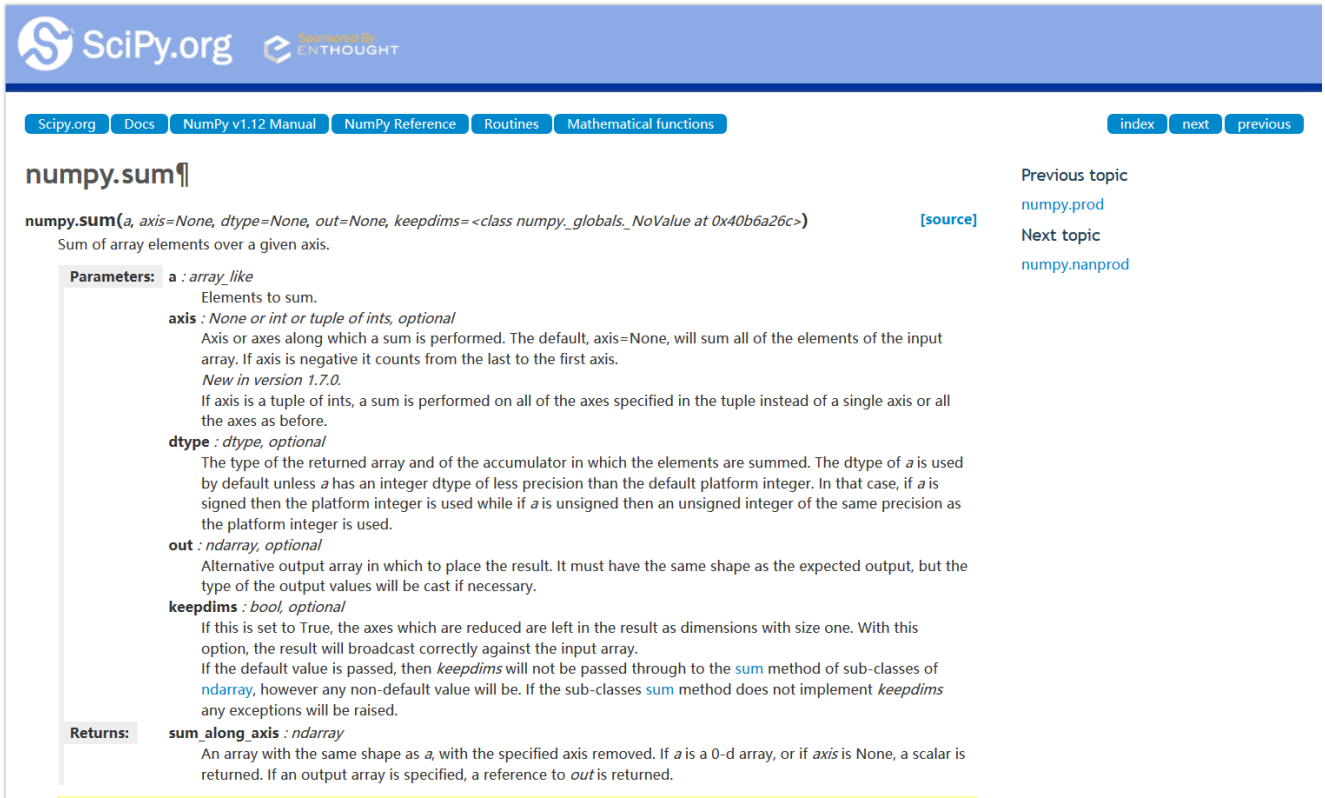
**(2) Python package that you should be familiar with:**

`numpy`

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:


(3) Python packages that **MUST NOT BE USED** (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;

`featureExtractor.py`: The location where you will write the PCA algorithm;

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

### *Files you should be NOT edit:*

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

`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, featureExtractor.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 $\mathrm{X} = [\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_n] \in \mathrm{R}^{m \times n}$ and label $\mathrm{Y} = [\mathrm{y}_1, \mathrm{y}_2, \dots, \mathrm{y}_n] \in \mathrm{R}^{l \times n}$, $\mathrm{W} \in \mathrm{R}^{l \times m}$ 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:

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

For a given feature vector $\mathrm{x}$, we score each class with:

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

And the predicted class of given feature vector $\mathrm{x}$ is

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

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 $\mathrm{m} = 784 (= 28 \times 28)$. We use the one-hot encoding to construct $\mathrm{y}$, thus for MNIST with 10 labels, $\mathrm{l} = 10$. 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)

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

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)

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

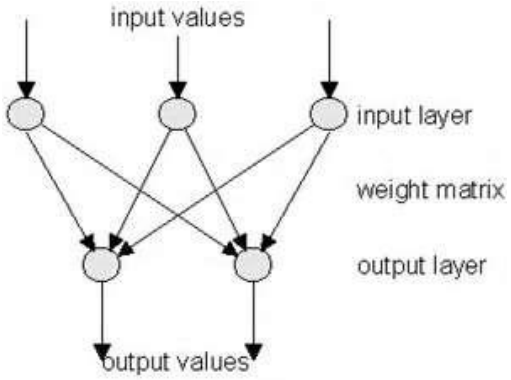
```
python dataClassifier.py –c perceptron
```

### *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 $x \in R^m$, then the output ($t \in R^l$) is determined by the following function:

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

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

where $W = [w_1, \dots, w_l] \in R^{m \times l}, b = [b_1, \dots, b_l] \in R^l, g$ is a non-linear function.

Suppose $f(x)$ can be defined as follows:

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

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 $\eta$ 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: One-vs-the-Rest Multiclass SVM (6 points)

In this question, you will implement the SVM classifier with the kernel trick. To apply SVM to multi-class condition, the one-vs-the-rest(one-against-all) algorithm should also be implemented.

Fill in the `SVMClassifier.train, SVMClassifier.classify, SVMClassifier.trainSVM` in `classifiers.py`. `SVMClassifier.trainSVM` just train two-class SVM, and multi-class SVM will be implemented in `SVMClassifier.train, SVMClassifier.classify`. Test one-vs-the-rest multiclass SVM with 5000 training data, 1000 validation data and 1000 test data. Run the code with

```
python dataClassifier.py –c svm –k rbf –C 10
```

**It will take at most 5 minutes to learn.**

**Hint**: to simplify the implementation of SVM, a constrained quadratic programming algorithm is implemented, and you will be free to use it. **You just need to write the dual problem of SVM and the** `SVMClassifier.optimizeConstrainedQuad` **function will optimize that**.

`SVMClassifier.optimizeConstrainedQuad(alpha, A, b, bounds, E, e)` will return the optimum of the following objective:

$$\min_{\alpha} \frac{1}{2}\alpha^T A\alpha + b^T\alpha$$

$$\text{s.t.} \quad \begin{matrix} \alpha \in R^n, A \in R^{n\times n}, b \in R^n, E \in R^{m\times n}, e \in R^m \\ l_i \le \alpha_i \le h_i \\ E\alpha = e \end{matrix}$$

where $\alpha = [\alpha_1, \dots, \alpha_n]^{\mathrm{T}}, \text{bounds} = [(l_1, h_1), \dots, (l_n, h_n)]$.

### *Grading (Expected accuracy: 94%):*

3 points for correctly implement the two-class SVM (correctly implemented in method `SVMClassifier.trainSVM`). Another 3 points will be graded according to the test accuracy:

| Points | Test Accuracy |
|---|---|
| 1 | >= 60% |
| 2 | >= 80% |
| 3 | >= 90% |

### *Algorithm:*

For two-class training data $(\mathrm{x}_1, \mathrm{y}_1), \dots, (\mathrm{x}_\mathrm{n}, y_n), \mathrm{y}_\mathrm{i} \in [-1,1]$, the dual problem of SVM is the following constrained quadratic programming:

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

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

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.

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

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

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

We recommend to use the following to compute $b$:

$$b = \frac{1}{|\{i: 0 < \alpha_i^* < C\}|} \sum_{0 < \alpha_i^* < C} \left( \mathrm{y_i} - \sum_{\alpha_j^* \neq 0} y_j \alpha_j^* K(x_j, x_i) \right)$$

For multi-class SVM, we use one-vs-the-rest(one against all) algorithm, in which the $i\text{th}$ $(1 \leq i \leq m)$ 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 4: Principle Component Analysis (PCA) (4 points)

In this question, you will implement PCA.

Fill in the `PCAFeatureExtractorDigit.fit/extract/reconstruct` methods in `featureExtractor.py`. Run the code with

```
python dataClassifier –f pca –m 32
```

Which means the dimension of feature is reduced to 32.

You can also train any classifier with features with PCA. Run the code with

```
python dataClassifier.py –f pca –m 32 –c knn
```

The expected classification accuracy is 93.2%, a little better than using the original feature.

### *Grading:*

3 points for correctly implement the `PCAFeatureExtractorDigit.fit/extract` methods, 1 point for correctly implement the `PCAFeatureExtractorDigit.reconstruct` method.

### *Algorithm:*

You may find PCA algorithm on the slides provided by the teacher.

## Question 5: Visualizing PCA (1 points)

Run the code with

```
python dataClassifier –f pca –m 32 –v
```

And find the reconstructed digit images in visualize/pca_digits.png. Answer the following question:

***Which following images is most likely represent the reconstructed digit images?***

(a) 

(b) 

Answer the question `answers.py` in the method `q5`, returning either `'a'` or `'b'`. **Note: the autograder will not grade this question.**

## Question 6: Sentiment Analysis (5 points extra credit)

In this question, you will perform text classification on **Stanford Sentiment Treebank (SST)** data set. There are about 8534 training sentences, 1099 validation sentences and 2208 testing sentences, each of which is assigned to one of 5 labels (very negative, negative, neutral, positive, very positive). You should analyse the sentiment of each sentence (very negative, negative, neutral, positive, very positive) in the SST data set.

Fill in the `FeatureExtractorText` and `ClassifierText` class in `sentimentAnalysis.py`. Test text classification by running the code with

```
python dataClassifier –d text
```

### *Grading：*

The best classification accuracy known is about 51%. With bag-of-words text feature, it is expected to get about 40% test accuracy with the perceptron classifier.

| Points | Test Accuracy |
|---|---|
| 1 | >= 30% |
| 2 | >= 35% |
| 3 | >= 40% |
| 4 | >= 43% |
| 5 | >= 46% |

**Note**: the vocabulary and the number of occurrences of a word in SSD sentences is provided. Only **3/4 of**

**test data** is provided, and the autograder will not grade this question. You can get the final classification results after the final grading is released.

`sklearn` can be used in this question.

***Congratulations! You've finished with Projects 3.***