Python Project: Supervised Machine Learning and Optimization, with MNIST dataset

Documentation

Created By: Yiwen Jin, Weiqi Liu

Assignment Description

MNIST is a database of handwritten digit images with 1 color channel (black and white), 784 features (28×28 pixels). It is a widely used real database to test machine learning methods. Python offers a library sklearn which provides Bunch type MNIST data and machine learning functions, including Principal Component Analysis (PCA), k-Nearest Neighborhoods (k-NN) and Support Vector Machine (SVM). In this project, we are going to answer this question: what are the best k-NN and SVM setups for learning and predicting the MNIST images, with regard to the fit time and accuracy?

Development Environment

Google Colaboratory is a platform built on Jupyter Notebook, allowing team members to program in Python together, compile Python programs interactively, combine the code and text cells and run all codes in our submitted ".ipynb" file to achieve all the outcomes below.

Link for Submission

https://github.com/AdvancedProgrammingProject2022/MachineLearningPythonProject/blob/main/FS2022_Assignment_Python_Project.ipynb

Task List

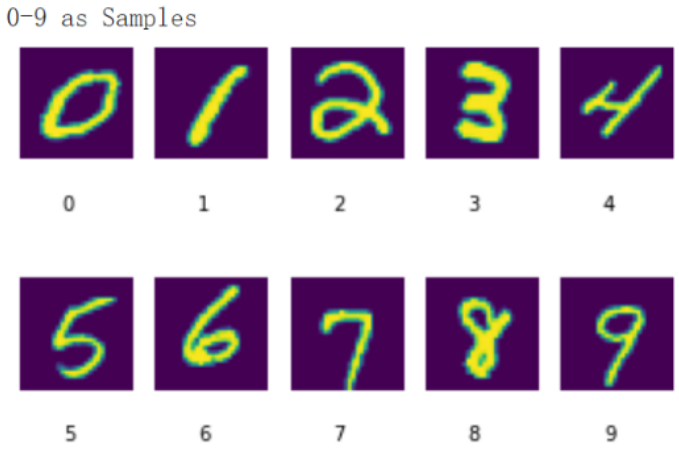
1. Download the Packages and Dataset
   Import sklearn and other packages. Download MNIST with *fetch_openml()*.

2. Create the Functions
   Encapsulate functions for calculation and plots.

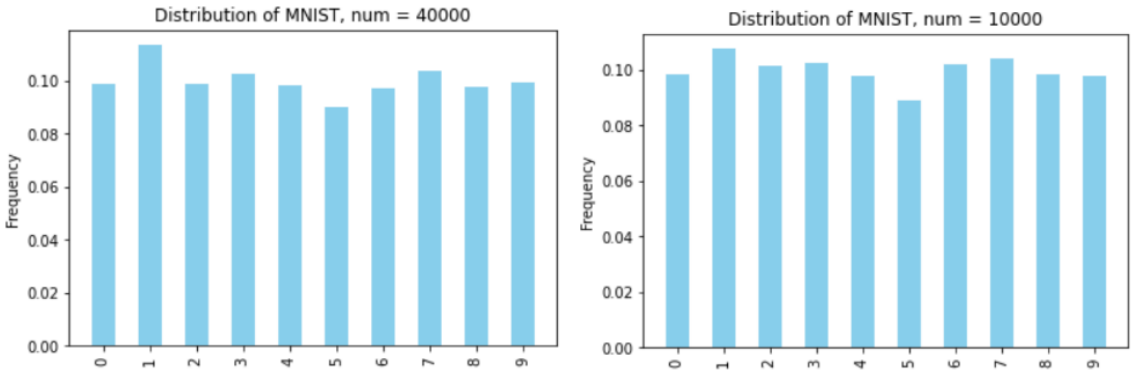
3. Explore the Dataset
   1) MNIST Data Structure
      MNIST contains 70000 images, each with 784 features. The images are classified into 10 classes from 0 to 9.
   2) Train-Test Split

Split the MNIST dataset into the training, validation and test sets with the sizes of 40000, 10000 and 20000, considering the running time of the project.

3) Visualize the Traning Set Samples



4) Balance Check for Class Distributions in the Training and the Validation Set

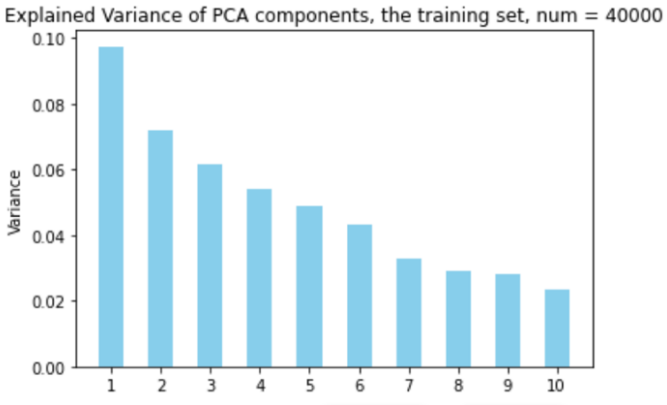


The 10 classes are all balanced in both the training (left) and the test (right) sets.

4. Principal Component Analysis (PCA)

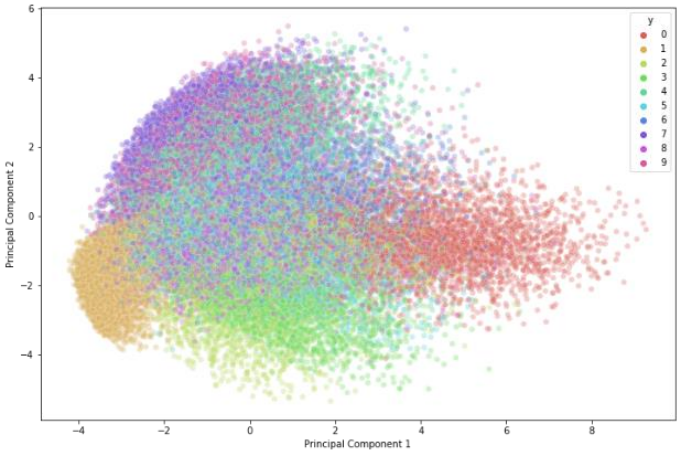
PCA is a feature extraction method, reducing the data dimension by projecting original data linearly into a space spanned by principal components.

1) Create PCA projection on the training set with total explained variance of **90%**.
2) The first **87 components** capture 90% of the variance. View the variance for the first ten components.



3) Transform the original training, validation and test sets into 87 principal components with latents learned from the training set.

4) View the first two principal components and check their classification ability.



5. k-Nearest Neighborhood (k-NN)

k-Nearest Neighborhood is a non-parametric supervised learning method. In classification, the input consists of the k closest training examples and the k-NN classifier determines the class of a data point by majority voting principle. In particular, **k** is a hyperparameter determining the degree of closeness. The biggest challenge of k-NN is to determine the optimal value for k.

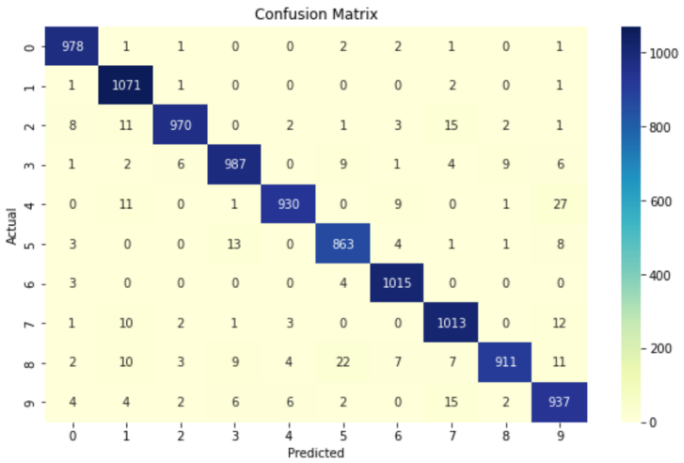
This section trains k-NN on the **training set** and tests k-NN on the **validation set**.

1) k-NN with Original Images

Run default *sklearn.KNeighborsClassifier()* with the **original data** as a naïve test.

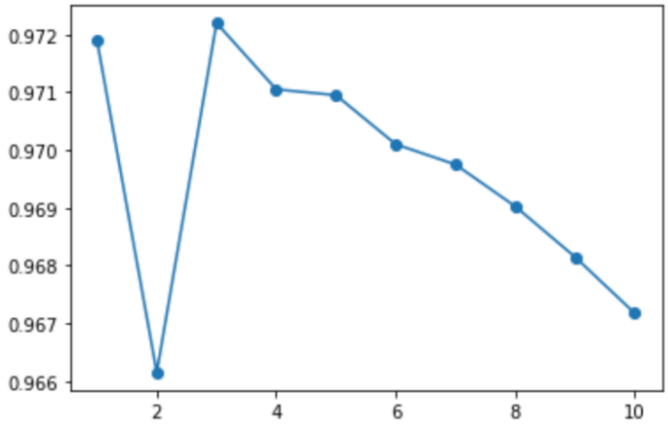
Results show that the **fit time** of the naïve k-NN (test set size: 40000) is **0.0847** minutes and the **accuracy** score (validation set size: 10000) is **0.9675**.

Plot a **confusion matrix**, in which the xlabel shows the prediction on the validation set, while the ylabel shows the ground truth. The diagonal line shows correct prediction numbers, while the remaining data show mis-classifications, e.g. 22 images of 8 are classified as 5.



2) k-NN with Principal Components and Tuning

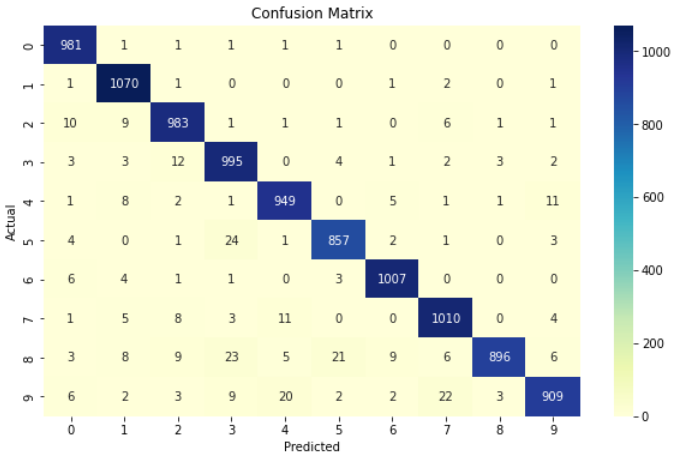
Run *sklearn.KNeighborsClassifier()* with **the 87 principal components** and search for the **k** with highest accuracy score by testing k from 1 to 10 with 5-fold cross-validation.



The graph shows that k = 3 yields the highest accuracy.

Results show that with k = 3, the **fit time** of the k-NN (test set size: 40000) is **0.0108** minutes and the **accuracy** score (validation set size: 10000) is **0.9657**. The fit time reduces significantly when using principal components with reduced dimensions while the accuracy remains almost the same.

The **confusion matrix** shows that a little more 5s and 8s are misclassified as 3, while only 21 images of 8s are classified as 5 this time.



6. Support Vector Machine (SVM)

Support Vector Machine is a supervised machine learning algorithm, in which **support vectors** refer to the data points that are close to the decision boundary, and a **decision boundary** is drawn in a way to maximize the distance to them. The choice of the kernel functions, the hyperparameters degree, gamma and C affects SVM performance on a particular dataset.

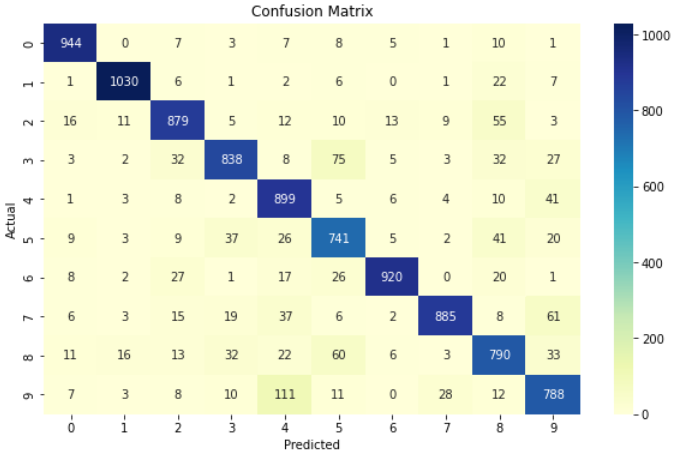
This section trains SVM on the **training set** and tests SVM on the **validation set**.

1) Naïve SVM with a Linear Kernel

Run default *sklearn.svm.LinearSVC(C=1)* with **the original data** as a naïve test.

Results show that the **fit time** of the naïve k-NN (test set size: 40000) is **2.3559** minutes and the **accuracy** score (validation set size: 10000) is **0.8714**. This result underperforms at both dimensions compared with k-NN.

The **confusion matrix** shows more mis-classifications, e.g. 111 images of 9s are classified as 4.

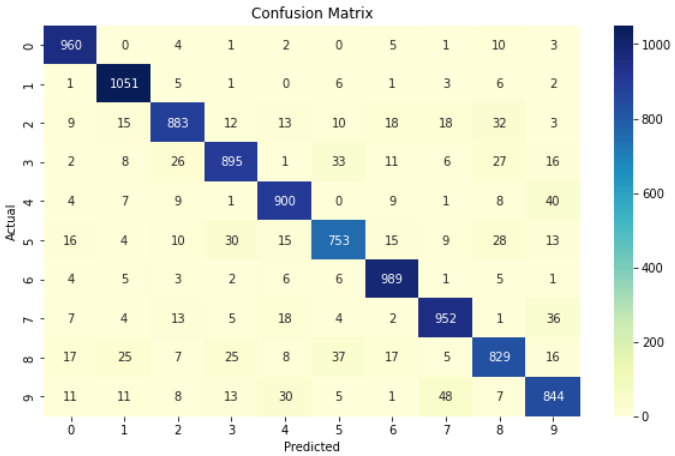


2) Naïve SVM with Principal Components

Run default *sklearn.svm.LinearSVC(C=1)* with **the 87 principal components.**

Results show that the **fit time** of the naïve k-NN (test set size: 40000) is **0.9661** minutes and the **accuracy** score (validation set size: 10000) is **0.9056**. This result is much improved in accuracy.

The **confusion matrix** shows that still 30 and 48 images of 9s are classified as 4 and 7.



3) Grid Search

Grid research is a traditional technique for tuning, and sklearn provides *GridSearchCV()* to combine the parameters set and return a table of grid research results with time and accuracy.

For the **linear kernel**, we search for the best **C** within {0.1, 1, 5, 10}, which is a regularization parameter defining the tolerance level of mis-specified data.

Results show that at the best, **C = 1** yields **0.9063** accuracy on average in 3-fold cross-validations.

For the **polynomial kernel**, we search for the best **degree** within {2, 3, 4}, **gamma** within {0.001, 0.01, 0.1}, and **C** within {0.1, 1, 10}, where the hyperparameters are plugged in the following kernel formula:

$$(\gamma < x', x > + r)^d$$

Results show that at the best, **degree = 3, gamma = 0.1 and C = 0.1** yields **0.9808** accuracy on average in 3-fold cross-validations.

View the best results of polynomial kernel in a table.

Mean Fit Time	Std Fit Time	Mean Score Time	Std Score Time	Param C	Param Degree	Param Gamma	Params	Split0 Test Score	Split1 Test Score	Split2 Test Score	Mean Test Score	Std Test Score	Rank Test Score
23.2112	0.7040	9.3626	0.0782	0.1	3	0.1	{'C': 0.1, 'degree': 3, 'gamma': 0.1}	0.9814	0.9791	0.9818	0.9808	0.0012	1

Report of Best Results

1. Preprocess

   Use the 87 principal components generated from the original training set.

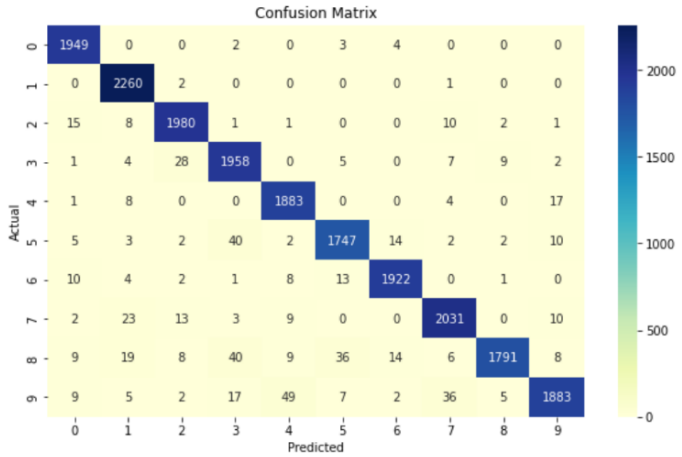
   Define a training set and a test set with the size of 50000, 20000 images, separately.

2. Result Table for best k-NN and SVM setups

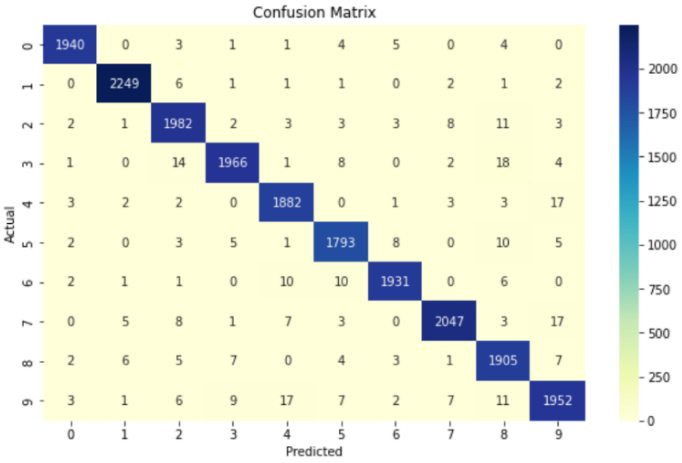
   The best k-NN setup is to choose **k = 3**, while the best SVM setup is to **choose a polynomial kernel, with degree = 3, gamma = 0.1 and C = 0.1**. The table indicates that, under the best settings and with the PCA feature extraction, the k-NN ran four times faster than the SVM, while the SVM performed slightly better on the accuracy than the k-NN.

Algorithms	With/Without PCA	Best Setting	Fit Time	Accuracy
k-NN	With PCA, n = 87	k = 3	0.0147694	0.9702
SVM	With PCA, n = 87	polynomial, degree = 3, gamma = 0.1, C = 0.1	0.406972	0.98235

The confusion matrix for best k-NN predictions shows that **97.02%** predictions are correct, while **49 of 9s are classified as 4**, 40 of 8s are classified as 3, and 40 of 5s are classified as 3, which are similar intuitively.



The confusion matrix for best SVM predictions shows that **98.24%** predictions are correct, while **17 of 9s are classified as 4**, 14 of 3s are classified as 2, and 11 of 9s are classified as 8. The most wrong predictions of 9s are similar no matter with the k-NN and the SVM, while the remaining mis-classifications of SVM seem to be less intuitive as those of the k-NN.



References

Python Tutorials

sklearn: https://scikit-learn.org

Download MNIST: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_openml.html

Cross Validation Function: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

Grid Search: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search

Confusion Matrix: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Results Table: https://towardsdatascience.com/how-to-easily-create-tables-in-python-2eaea447d8fd

Machine Learning Tutorials

Principal Component Analysis (PCA): https://www.datacamp.com/tutorial/principal-component-analysis-in-python

k-Nearest Neighbourhoods (k-NN): https://towardsdatascience.com/15-must-know-machine-learning-algorithms-44faf6bc758e https://towardsdatascience.com/knn-algorithm-what-when-why-how-41405c16c36f

Support Vector Machine (SVM): https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning#svm

https://towardsdatascience.com/15-must-know-machine-learning-algorithms-44faf6bc758e

https://dmkothari.github.io/Machine-Learning-Projects/SVM_with_MNIST.html https://seaborn.pydata.org/generated/seaborn.heatmap.html

Tuning for SVM: https://aqsa-qadir44.medium.com/tuning-parameters-of-svm-kernel-regularization-gamma-and-margin-5f2f6639121a