COMP9123 Assignment 1

S1 2021

Yangxuan Qian	Xumou Zhang
SID: 510128443	SID: 510317173
Unikey: yqia8393	Unikey: xzha6842

1. Introduction

1.1 Aim of the study and importance of the study

In the past few decades, computers have changed almost all industry operations. Now, because of the rise of Machine Learning, the progress and speed of automation in various industries in the world have increased rapidly. The influence of Machine Learning on our human society may be comparable to the invention of electricity.

This report is aiming to practice the fundamental of the data analysis procedures from the data pre-processing, to build the basic classification models, parameter tuning and finally evaluation of our tuning result. It is important to complete this study because it gives us an opportunity to practice the basis of the data analysis procedures. It also forces us to actual practice the implementation of the classifiers and let us try to tune the parameters of those chosen classifiers.

Also, it is a test run for us to discover the actual time and resource consumption of the classifier models. It is important for us to go through this experience not only as the student, but also as the future data scientist. Because we will not always have time to run all the models with real hardware, sometimes we must choose the model in our brain using our experience on those models before we implement the model on the computer.

1.2 Dataset Overview

The dataset we are using in the project is provided by our teaching team. It is a MNIST dataset contains 4 files in total. Two are given as the training process, and the other two are given for the testing and evaluating in the end. The label set in our dataset have 10 different categories in terms of the numerical labels: 0 T-shirt/Top; 1 Trouser;

2 Pullover; 3 Dress; 4 Coat; 5 Sandal; 6 Shirt; 7 Sneaker; 8 Bag; 9 Ankle boot. There are 30,000 sets of data in the training set and 5000 in the testing set. The dataset contains 784 columns which is the squared value of 28 because each data could show us a picture with the resolution of 28*28. Here is an example of the picture shown from the dataset:

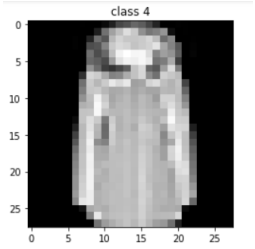


Figure 1 - Random Example of 28*28 Picture from MNIST Dataset

1.3 Hardware and Software Specifications:

The CPU we are using is Intel i7-11375H (11th gen H35 CPU) which has 4 cores and 8 threads. The laptop has 16GB of RAM and 1TB of NVME SSD storage. The dedicate GPU is Nvidia mobile 3060 65W.

The system is Windows 10, and we are running the code on Jupyter Notebook.

2. Method

We are going to discuss the procedures details of data pre-processing, classifiers implementation and the parameter tuning in this section.

2.1 Data Pre-processing

The methods adopted in this case are data normalization and Principal Component Analysis (PCA). The main purpose of this section is to adjust the dataset so that the classifiers would perform better or use significance less time in the further study.

The first step is using data normalization. Here is the general equation of the normalization:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Figure 2 - Data Normalization General Formula

The equation of the normalization has three different elements in it. “x” is the initial value, “min(x)” is the minimum value of the x set and the “max(x)” is the maximum value of the x set. It would re-size each data point into [0,1] intervals, so the classifiers in the future sections would not facing the calculation with the large numbers. The

reason we want to avoid that is because the computer would need much more resource to process the large number multiplications compare to the small numbers such as the number in the [0,1] interval. In Python, the normalization is achieved using the MinMaxScaler from the sklearn.preprocessing package with simple implementation.

The next step is to process the PCA analysis for the dataset. The idea of the PCA analysis is to project the shadow of the high dimension data into lower dimension so that most of the details are retained and we get the dataset with less dimensions. It is the most important data pre-processing step in this project because we have 784 dimensions in our dataset initially. To perform the PCA analysis in Python, we need to use the PCA from the sklearn.decomposition package.

To implement the classifiers for the dataset with so many dimensions, we need an extremely long period of time to run those classifiers and even more times to run the tuning function afterward. We cannot afford that kind of time.

Besides, there are many columns of data in this dataset are unrelated to the result of prediction since they always show zero information which means those columns should be reduced. Thus, the PCA analysis comes handy because it could reduce the dimension of the dataset significantly without losing too much information compared to delete parts of the columns directly. Here is the graph of PCA analysis which shows the total variance explained by the dataset with respect to the number of the components:

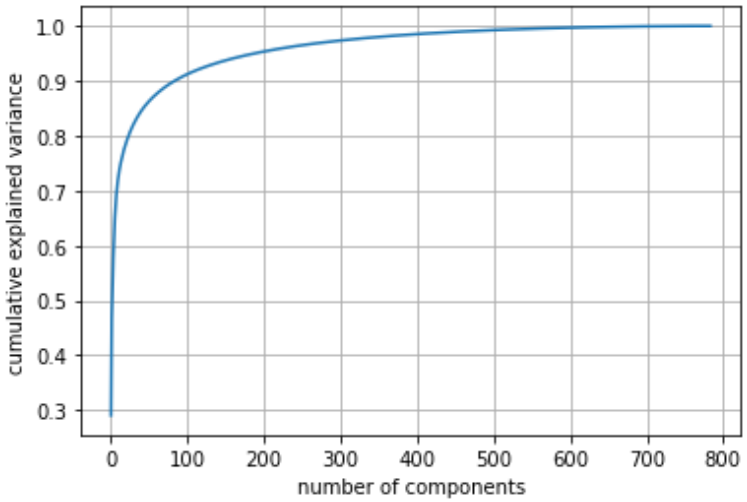


Figure 3 - PCA analysis graph

The elbow point of the PCA analysis occurs around 50 to 60 components where the total explained variance at that point would be around 87%. To ensure the classification quality, we choose around 95% of the total variances in this case. There are 188 components left in the dataset after the pre-processing step, while the original number of components is 784 (28*28). Thus, we successfully reduced 76% of dimension with only 5% losses in the total variance explained.

2.2 Classifiers and Initial Implementation

We want to use the classifiers in this assignment rather than the regression models because the key idea of this assignment is to sort the 30,000 data in the training set and the 5,000 in the testing set into 10 different categories of clothing which indicated in the beginning of the assignment questions. Therefore, it is only logical to implement classifiers instead of regression model here since the purpose of the regression model is to predict the further data trend instead of the outcomes of inputs.

We are going to implement KNN, Decision Tree and SVM classification models in this assignment. In Python, we need to use the sklearn package to implement these classifers.

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm. KNN can be used not only for classification but also for regression. The prediction will be the average value of the class values (numerical) of the k nearest neighbors. With different values of k, the prediction accuracies are different. The key issue is to find out the k value with the best accuracy via training data.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers' detection. Different Kernel functions of SVM can be specified for the decision function.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The model is used to predict the value of a target variable by learning simple decision rules inferred from the data features Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

We are asked to implement the selected models with default parameters and perform the 10-fold cross-validations. The reason behind this is to recognize and record the initial performance of each classification model, and to prepared for the comparison after the parameter tuning. The 10-fold validation used in this step is an often-used validation technique in the classification and regression model analysis. It split the input data into 10 partitions, and then it picks one partition as the test set and left rest of them as the training set. The 10-fold validation then will check the accuracy score of each case. The 10-fold validation would perform 10 times accuracy check in total. This gives us a more accurate idea of how well the model is performing or how accurate the model is. The 10-fold validation and the accuracy scores are implemented by sklearn package in Python.

2.3 Parameter Tuning

The essential of parameter tuning is to find out the best parameter for a model via comparing the accuracy among the results base on a set of predefined values for each

parameter. For each classifier, we would like to find the best parameters using grid search with 10-fold stratified cross-validation.

Classifier	Parameter	Tuning values
KNN	n_neighbors	2,4,6,8,10,12,14,16
SVM	kernel C	'rbf','linear' 5,7,9,11,13
Decision Trees	max_depth splitter	3,7,11,15,19,23 'best','random'

Table 1 - Details of tuning parameters

We choose to use GridSearchCV from sklearn.model_selection package in Python to implement the parameter tuning step. The above table shows the parameters list we are going to use as the input parameters for the GridSearchCV function in Python. The GridSearchCV function will then run through every possible combination of the given parameters and finally give us the best parameter combination and the accuracy scores with respect to the training set.

2.4 Model Comparison

In this step, we would have the best parameter combination for each model. Therefore, it is appropriate to do the model comparison and find out which one is the best model with respect to the accuracy. We will implement this step by using the best parameter combination as the input parameters in each classifier model, and then fit the model with the first 2,000 data from the test set. After that, we will evaluate the prediction values of each model with respect to the actual values from test set and get the accuracy scores. In the end, simply comparing the accuracy scores across three models would give us the best model in the project which we will be using the model to generate the output H5 file in the end.

3 Experiments result and discussion

We will present and discuss the result data before, during and after tuning in this part. In the first two parts of the assignment, we will be using the training set only. The training set is the dataset contains 30,000 data in it. The purpose of this is to isolate the testing set from the training procedures so that the training result would not contaminate by the testing set. The testing set will be using in the last part for only evaluation and comparison purpose. We can produce fair results if we are following these steps.

3.1 Result Before Parameter Tuning

As we discussed in the section 2.2, we are required to implement the chosen

classification models with their default parameters. In this section, we first go through the model with original data and PCA processed data, and then implement 10-fold validation for each model. The dataset we used in this section is the training set which contains 30,000 data. The table 2 shows the result accuracy scores of each model using the original data which contains 784 dimensions and the PCA processed data which left with 188 dimensions:

Classifier	**Original Data Accuracy**	**PCA Data Accuracy**	**Time Consuming (second)**
KNN	0.844	0.850	25.658
Decision Trees	0.792	0.740	126.953
SVM	0.878	0.882	285.884

Table 2 - Accuracy Comparison between Original and PCA Processed Data

Based on the comparison result, we can conclude that the reduced data has the almost same accuracy as the original data. We can directly use the reduced data for the further operations. If we go through the results in each column, we could find out that the SVM model perform the best under the default condition, following by the KNN model, and left the Decision Tree model at the end.

3.2 Result of Parameter Tuning

The result of this section is based on the procedures we discussed in the section 2.3. We are using the parameter lists in the Table 1 as the input of the GridSearchCV function and collect the outcome from the function to make our decision for the best parameters combination for each of these three models. We also set cv to 10 for the internal 10-fold cross validation and set n_jobs to -1 to get the full performance from the CPU.

3.2.1 KNN Classification Model Parameter Tuning Result:

The KNN model only have one parameter to manipulate with, it is the n_neighbors. Here is the graph of the KNN parameter versus the mean test score:

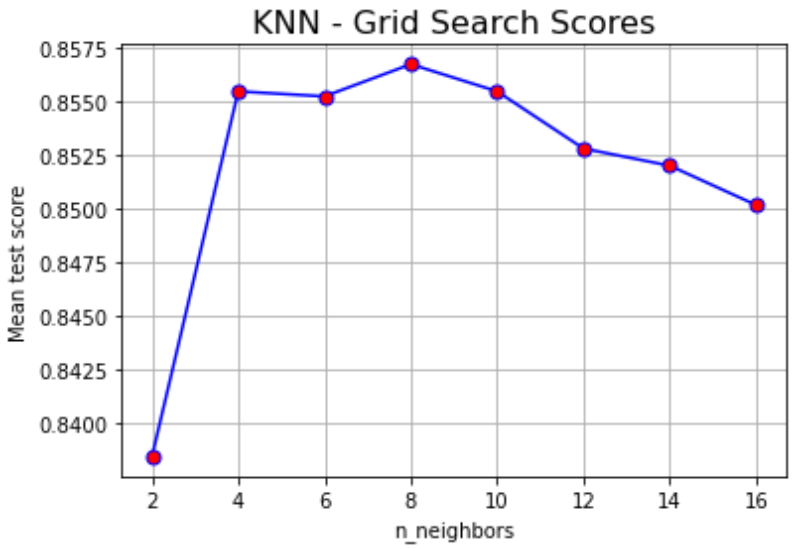


Figure 4 – n_neighbors VS. Mean test score for KNN Model

According to figure 4, when n_neighbors reach value of 8, the test has the best test score which is 0.857.

3.2.2 SVM Classification Model Parameter Tuning Result:

As for the SVM model, it has 3 different parameters that would affect the result, however, we find out the best value for the gamma parameter is "scale" which is the default value. Thus, we are only tuning the criterion and the C parameters here:

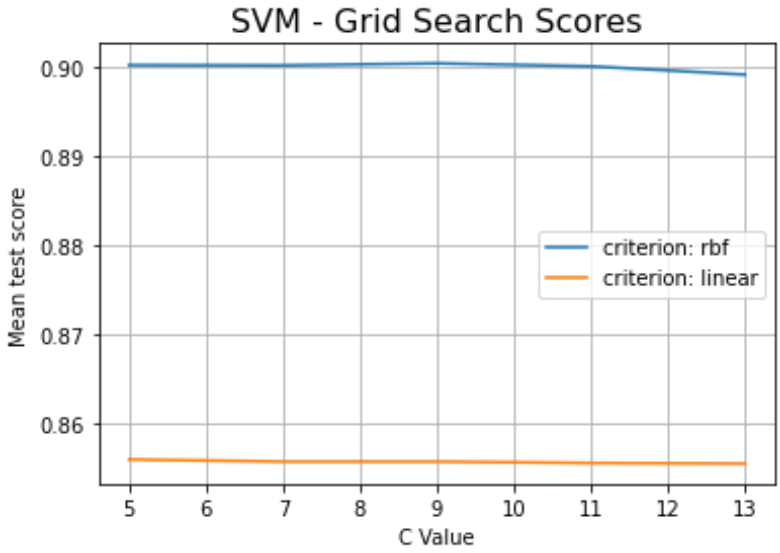


Figure 5 - C VS. Mean test score for SVM Model with Different Criterion

We can conclude that it has the best accuracy when set kernel = 'rbf' and C = 9 from figure 3. The best accuracy is about 0.900.

3.2.3 Decision Tree Classification Model Parameter Tuning Result:

In the decision tree model, we are focusing on the max_depth parameter and the splitter parameter. The criterion parameter, similar to the SVM case, we find out that the "entropy" criterion always shows the better result than other inputs. Here is the graph of the decision tree model:

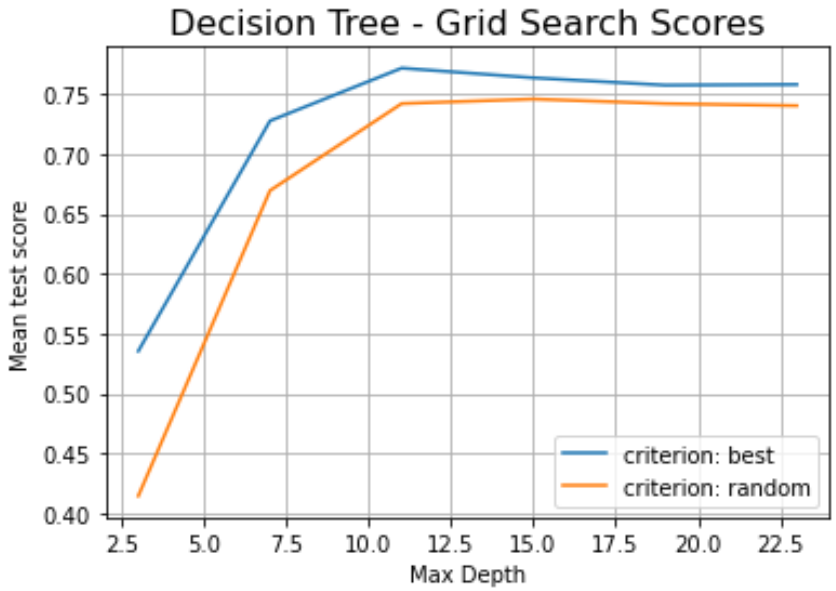


Figure 6 – Max Depth VS. Mean test score for Decision Tree Model with Different Splitter (*it should be splitter instead of criterion in the label)

We can get the conclusion that it has the best accuracy when set splitter = 'best' and max_depth = 11 from figure 4. The best accuracy is about 0.772.

3.3 Final comparison After Parameter Tuning

Since the best parameters of the 3 classifiers are tuned, now we can compare these classifiers with the best parameters using the training set and the testing set for the evaluation purpose. In Python, after we initialize the model with the results we got from previous section, we would fit the model using the training set as usual. After that, we would predict the result with both training set and the testing set. Here is the result:

Classifier	Accuracy on Train Set	Accuracy on Test Set	Time(second)
KNN	0.873	0.835	13.540
SVM	0.951	0.881	50.438
Decision Trees	0.853	0.757	7.773

Table 3 - Final comparison

We can observe from table 3 that the ranking of the model accuracy stays the same even

though the actual accuracy reduced when we switch the dataset from the training set to the testing set.

4 Conclusion

4.1 General Conclusion

Before building the classifier model, normalization and PCA are introduced to pre-process the train and test data to reduce the data dimension. The reduction will increase the performance of training and prediction significantly. In the 3 classifiers: KNN, SVM and Decision Trees, KNN has the highest performance and good accuracy. SVM has the highest accuracy but the lowest performance.

4.2 Future Recommendation

In our project we find out that the SVM model preform the best in all 3 of them but still the accuracy of the model has a hard time to reach 90% accuracy. The suggestion of that is to using bagging classifier with SVM as the base classifier or using other boosting classifiers.

5 Appendix

5.1 Details about the submission files

There are 3 main files in our submission:

1. A1 Report PDF file.
2. Complete version of Code file in PDF which contains everything we had done in the assignment includes the GridSearchCV part where we tune the parameters.
3. a compressed ZIP file named Classifiers.

In the ZIP file which named Classifiers there are four files:

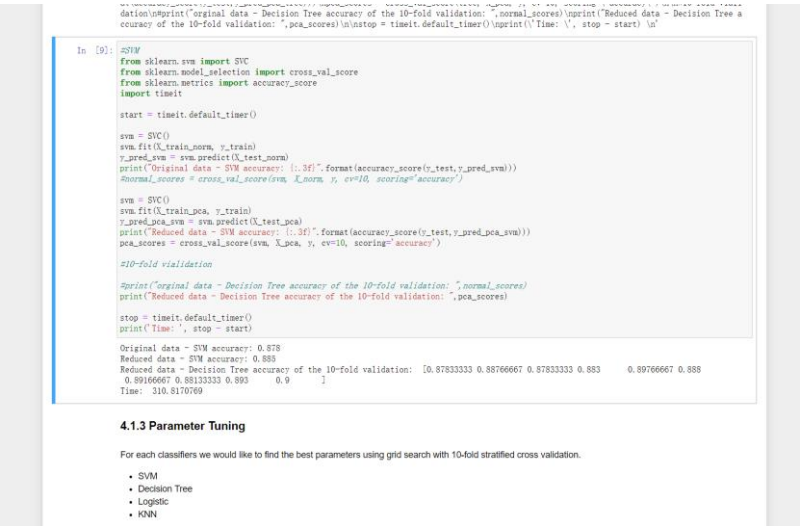
1. An ipynb file for the marker to run directly, it only contains the pre-processing steps, the SVM model with default parameters, SVM model with tuned parameters, and the part where we output the H5 file.
2. Empty folder named “Input”
3. Empty folder named “Output”

5.2 How to run our project and other notices

Please simply run the file inside ZIP Classifiers folder. I already comment out all the unwanted part.

If you need to check any other part, simply undo the comment out from the ipynb file.

Also, if you do not need to check the model run before the tuning, you can comment out the following code from the file:



```
In [9]: #SVM
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import timeit

start = timeit.default_timer()

svm = SVC()
svm.fit(X_train_norm, y_train)
y_pred_svm = svm.predict(X_test_norm)
print('Original data - SVM accuracy: {:.3f}'.format(accuracy_score(y_test,y_pred_svm)))
#normal_scores = cross_val_score(svm, X_norm, y, cv=10, scoring='accuracy')

svm = SVC()
svm.fit(X_train_pca, y_train)
y_pred_pca_svm = svm.predict(X_test_pca)
print('Reduced data - SVM accuracy: {:.3f}'.format(accuracy_score(y_test,y_pred_pca_svm)))
pca_scores = cross_val_score(svm, X_pca, y, cv=10, scoring='accuracy')

#10-fold vialidation

#print("orginal data - Decision Tree accuracy of the 10-fold validation: ",normal_scores)
print("Reduced data - Decision Tree accuracy of the 10-fold validation: ",pca_scores)

stop = timeit.default_timer()
print('Time: ', stop - start)

Original data - SVM accuracy: 0.878
Reduced data - SVM accuracy: 0.885
Reduced data - Decision Tree accuracy of the 10-fold validation:  [0.87833333 0.88766667 0.87833333 0.883      0.89766667 0.888
 0.89166667 0.88133333 0.893      0.9       ]
Time:  310.8170769
```

4.1.3 Parameter Tuning

For each classifiers we would like to find the best parameters using grid search with 10-fold stratified cross validation.

- SVM
- Decision Tree
- Logistic
- KNN

Figure 7 – SVM Code Before Question 4.1.3 Parameter Tuning

5.3 Referenece

1. E. Burns, “What is machine learning and why is it important?,” *SearchEnterpriseAI*, 30-Mar-2021. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML. [Accessed: 20-Sep-2021].
2. S. B. Kotsiantis , “Supervised Machine Learning: A Review of Classification Techniques,” thesis, 2007.
3. “sklearn.decomposition.pca¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. [Accessed: 20-Sep-2021].
4. “sklearn.metrics.accuracy_score¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. [Accessed: 20-Sep-2021].
5. “sklearn.model_selection.gridsearchcv¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed: 20-Sep-2021].
6. “sklearn.neighbors.kneighborsclassifier¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed: 20-Sep-2021].
7. “sklearn.svm.svc¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 20-Sep-2021].
8. “sklearn.tree.decisiontreeclassifier¶,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Accessed: 20-Sep-2021].