Stone Barrett

CSE 590: Intro to Machine Learning – Section 53

Homework 4

For this assignment, I will be applying two different classification algorithms to the provided fashion-MNIST dataset that will be preprocessed using four different methods. 4-fold cross validation will be used to determine the optimal parameters for each algorithm. For the Kernel Support Vector Machine classifier, I will be varying parameters C, gamma, and the choice of kernel. For the Multilayer Perceptron classifier, I will be varying parameters solver, max_iter, alpha, and random_state. The four methods of preprocessing that will be applied to the data before fitting to a model are no preprocessing, standard scaling, robust scaling, and minmax scaling.

Problem 1: Kernel Support Vector Machines (KSVM)

No Preprocessing & Parameter Tuning

Running this classification algorithm with default parameters (C = 1, gamma = scale, kernel = rbf) on data that has no preprocessing applied to it results in the following:

```
Cross-validation scores: [0.894     0.8968    0.8988    0.89155662]
Average cross-validation score: 0.8952891556622649
```

Figure 4.1

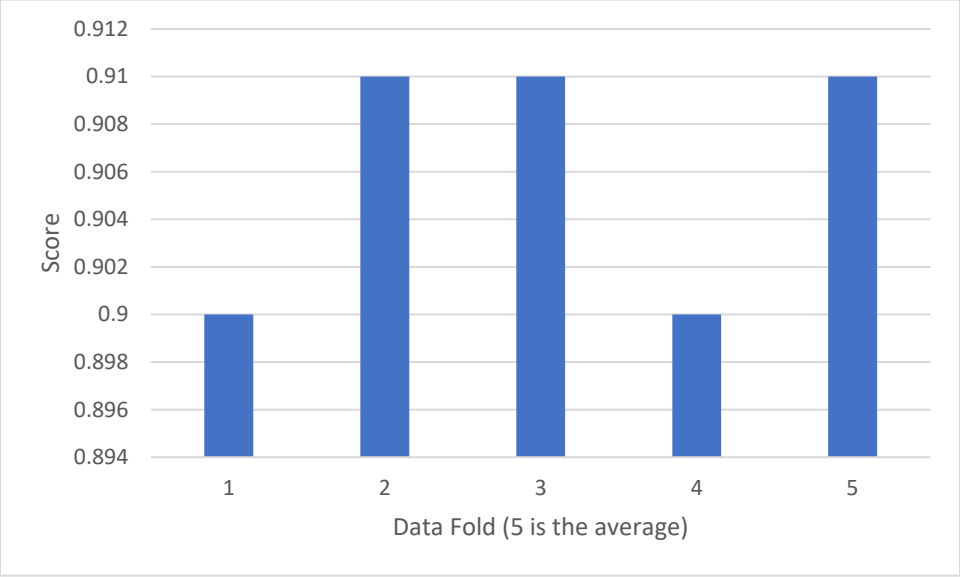

Figure 4.1G

Next I will try changing gamma to auto.

```
Cross-validation scores: [0.2004     0.2        0.2        0.39615846]
Average cross-validation score: 0.24913961584633854
```
Figure 4.2

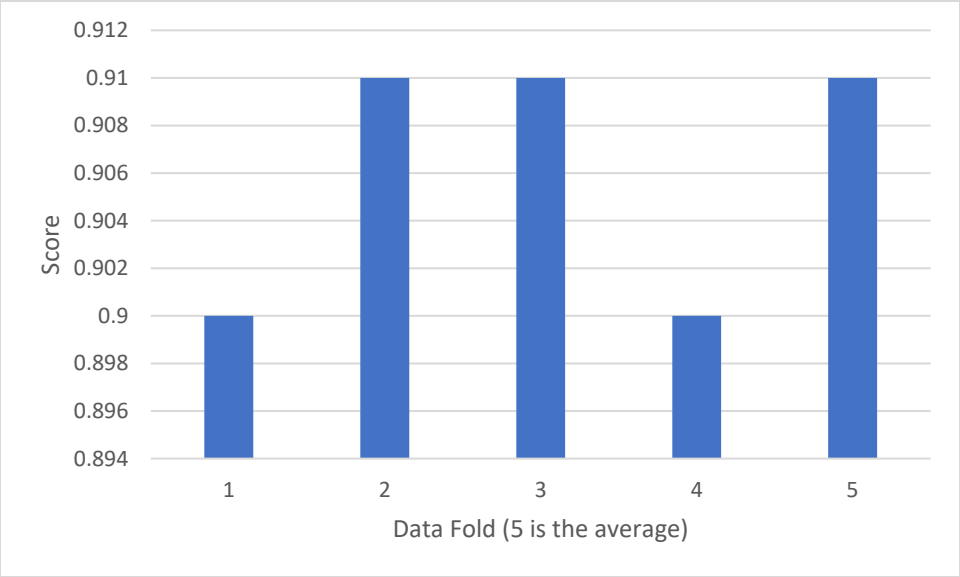

Figure 4.2G

That turned out very poorly, so I will revert gamma to scale and try kernel = poly, then sigmoid. (An attempted run with kernel = linear resulted in the script seemingly making no progress for hours of runtime.)

```
Cross-validation scores: [0.8744     0.8804     0.8748     0.85994398]
Average cross-validation score: 0.8723859943977591
```
Figure 4.3

```
Cross-validation scores: [0.4428     0.4484     0.4424     0.43857543]
Average cross-validation score: 0.4430438575430172
```
Figure 4.4

So, it would seem that rbf is the optimal kernel setting for this situation. Now I will vary the parameter C at increasing factors of ten.

C = 10 results in the following:

```
Cross-validation scores: [0.904      0.9104     0.9116     0.90196078]
Average cross-validation score: 0.9069901960784313
```
Figure 4.5

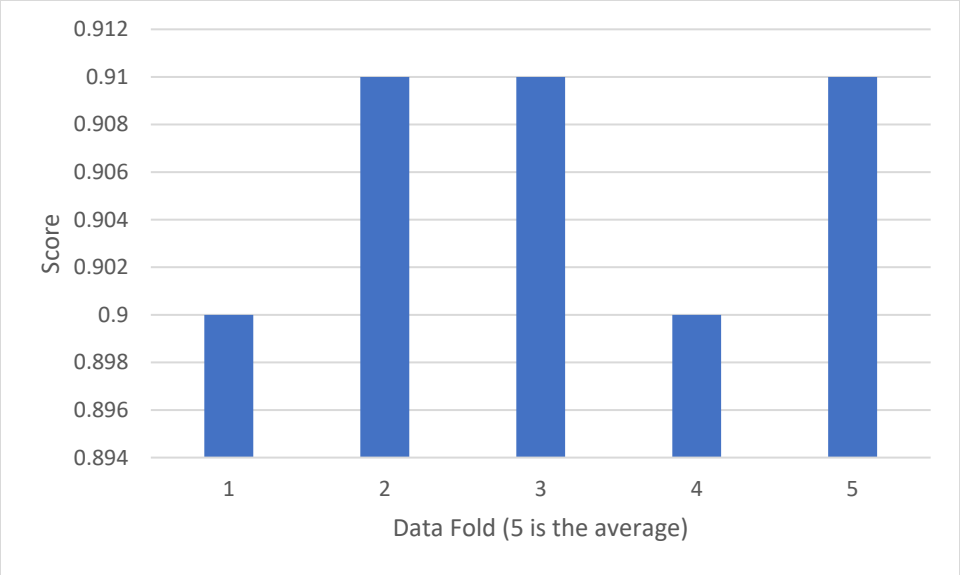

Figure 4.5G

C = 100:

```
Cross-validation scores: [0.896      0.896      0.9076     0.89835934]
Average cross-validation score: 0.8994898359343738
```
Figure 4.6

C = 1000:

```
Cross-validation scores: [0.896      0.8972     0.9076     0.89755902]
Average cross-validation score: 0.8995897559023609
```
Figure 4.7

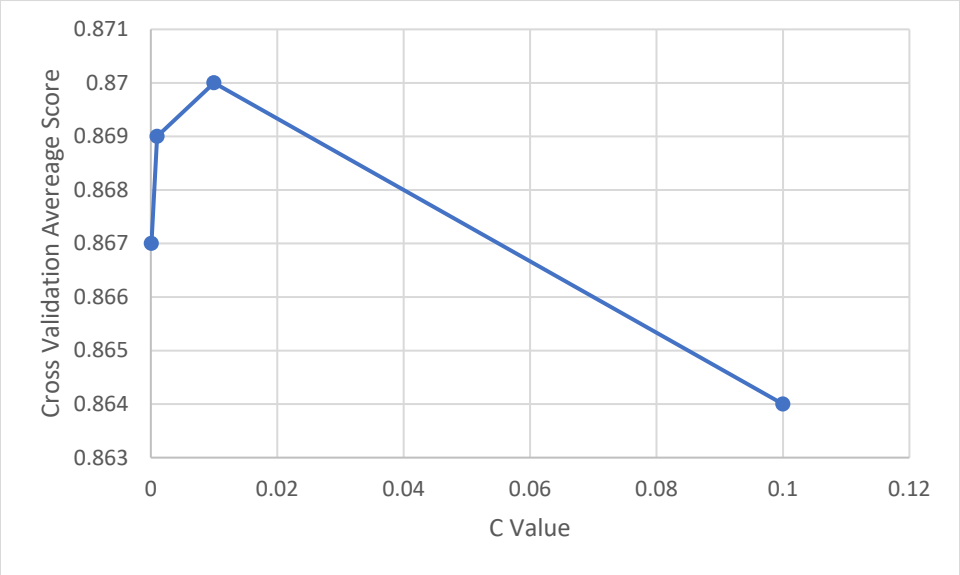

Figure 4.7G

From these tests, we can see that kernel and gamma are optimized at their defaults, and C's optimal value is at or near 10. With this in mind, the final accuracies scored on the dataset with no preprocessing done is:

```
Accuracy on training set: 0.98
Accuracy on test set: 0.90
```
Figure 4.8

This shows a slight overfit when using this model. However, preprocessing the data could help this. Since I used cross-validation to find the optimal parameters for this situation, I will continue to use those parameters for the remainder of problem 1. (kernel = rbf, gamma = scale, C = 10)

Standard Scaling

Using the StandardScaler preprocessing method results in the following scores:

```
Accuracy on training set: 0.99
Accuracy on test set: 0.89
```
Figure 4.9

These scores mean that using Standard Scaling results in an even worse case of overfitting.

Robust Scaling

Using the RobustScaler preprocessing method results in the following scores:

```
Accuracy on training set: 0.86
Accuracy on test set: 0.81
```
Figure 4.10

This is a noticeably lower training accuracy score, but not so low that it could be considered underfitting. The test accuracy is close enough that it's a less severe case of overfitting as well!

MinMax Scaling

Using the MinMaxScaler preprocessing method results in the following scores:

```
Accuracy on training set: 0.98
Accuracy on test set: 0.90
```
Figure 4.11

This is an identical set of scores to what was returned in the test without preprocessing.

Predict_proba

I was unable to utilize the *predict_proba* method to identify sample classification probabilities, as it returned errors stating that the data was not correctly formatted for its use. I tried different approaches to passing the data to the method (including the one covered in lecture) and none seemed to work.

Discussion

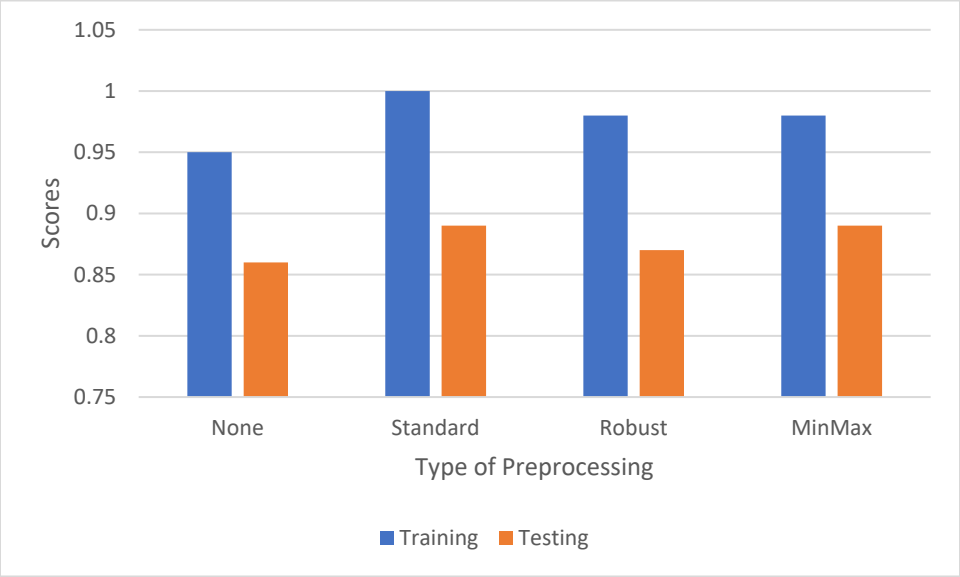

Figure 4.11G

Given the results of these experiments, it is safe to say that the KSVM classifier is a decent algorithm to use with the provided dataset. It is quite accurate, does not overfit or underfit when the data is preprocessed correctly, and does not require much of computational resources.

Problem 2: Multilayer Perceptrons (MLP)

No Preprocessing + Parameter Tuning

Running this classification algorithm with default parameters (solver = adam, max_iter = 200, alpha = .0001, random_state = None) on data that has no preprocessing applied to it results in the following:

```
Cross-validation scores: [0.8668     0.8724     0.876      0.85314126]
Average cross-validation score: 0.8670853141256503
```
Figure 4.12

Since it has a very limited number of options, I will narrow down the optimal solver first.

Incidentally, solver = lbfgs will not run unless max_iter is changed. I increased it to 1000 and got the same message. I increased it to 10000 and finally got these results:

```
Cross-validation scores: [0.4564     0.8684     0.868      0.85994398]
Average cross-validation score: 0.7631859943977591
```
Figure 4.13

Given that this is worse than before while taking FAR longer to compute, I am going to revert max_iter to default and try running with solver = sgd.

```
Cross-validation scores: [0.3396     0.378      0.364      0.19967987]
Average cross-validation score: 0.32031996798719486
```
Figure 4.14

This is significantly worse than either of the scores before, so I will revert to solver = adam going forward.

Having tested max_iter = 1000 with solver = adam and seen no difference from default max_iter, I will keep max_iter at default going forward.

Next, I will test different values of alpha in increasing factors of ten.

Alpha = .001:

```
Cross-validation scores: [0.8716     0.8616     0.8788     0.86754702]
Average cross-validation score: 0.8698867547018808
```
Figure 4.15

Alpha = .01:

```
Cross-validation scores: [0.8792     0.8592     0.8768     0.8667467]
Average cross-validation score: 0.8704866746698678
```
Figure 4.16

Alpha = .1:

```
Cross-validation scores: [0.8676     0.8684     0.862      0.85834334]
Average cross-validation score: 0.8640858343337334
```
Figure 4.17

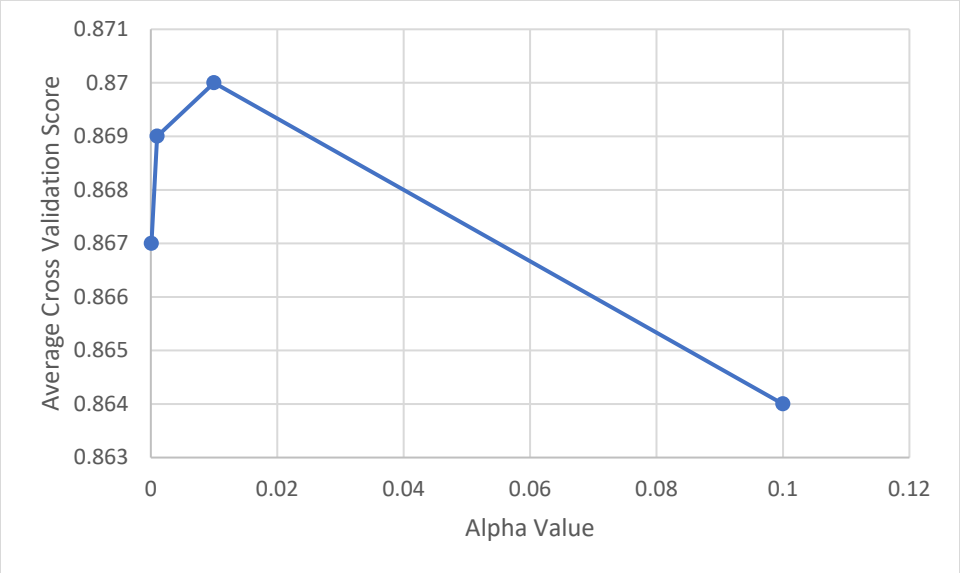

Figure 4.17G

From these tests, we can see that alpha = .01 actually performs slightly better than the other tested values, therefore I will keep alpha set to .01 from here on.

Next, I will test random_state at a different value.

Random_state = 1:

```
Cross-validation scores: [0.8688     0.8608     0.8668     0.86954782]
Average cross-validation score: 0.8664869547819127
```
Figure 4.18

Though it may be purely by chance, random_state seems to return better results in this situation when set to None. However, to achieve more consistent results, I will set it to 1.

So, the final scores for the MLP classifier with no preprocessing done and parameters set as solver = adam, max_iter = 200, alpha = .01, and random_state = 1. are:

```
Accuracy on training set: 0.95
Accuracy on test set: 0.86
```
Figure 4.19

Unfortunately, this stands as an instance of a slight overfitting.

As in the previous problem, I will use these optimal parameters going forward with all different forms of preprocessing.

Standard Scaling

Using the StandardScaler preprocessing method results in the following scores:

```
Accuracy on training set: 1.00
Accuracy on test set: 0.89
```
Figure 4.20

Robust Scaling

Using the RobustScaler preprocessing method results in the following scores:

```
Accuracy on training set: 0.98
Accuracy on test set: 0.87
```
Figure 4.21

MinMax Scaling

Using the MinMaxScaler preprocessing method results in the following scores:

```
Accuracy on training set: 0.98
Accuracy on test set: 0.89
```
Figure 4.22

Predict_proba

I was unable to utilize the *predict_proba* method to identify sample classification probabilities, as it returned errors stating that the data was not correctly formatted for its use. I tried different approaches to passing the data to the method (including the one covered in lecture) and none seemed to work.

Discussion

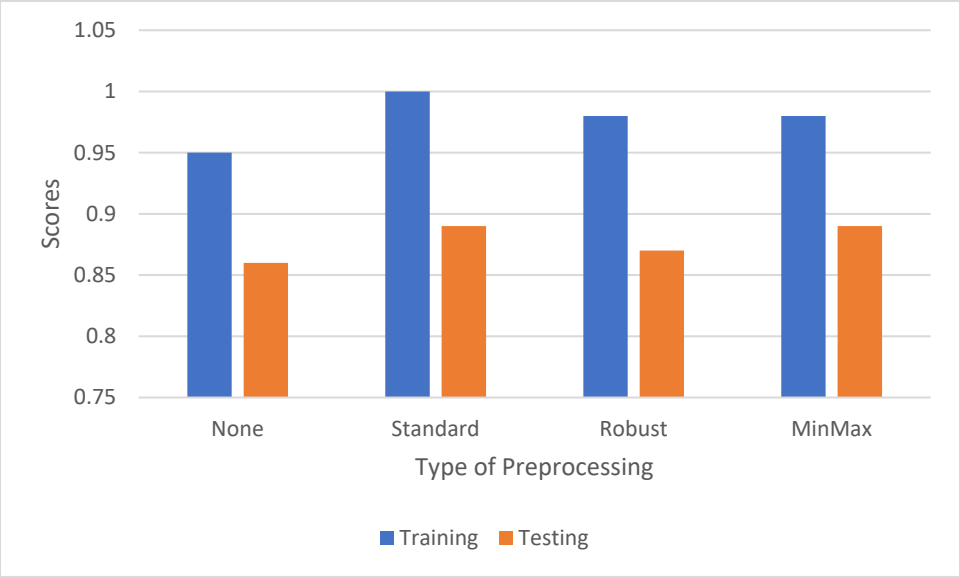

Figure 4.22G

Given the results of these experiments, it would seem that MinMax preprocessing is the optimal scaling for this dataset when fitted to this model. It is a slight overfit, but it is the least overfitting of the bunch (highest test to train ratio). This model is quite accurate for this classification (high generalization) and does not require much of computational resources.