Project Title:

A Supervised Learning Algorithm, Using KRLS With A Gaussian Kernel

Bahaar Khalilian
Hesam Mohebi

1. Project Objective:

Put into practice what we have learned about how to train a supervised learning algorithm using KRLS with a Gaussian kernel. Using the training dataset provided, Q-Fold Cross Validation must be used to select the hyperparameter λ and the Gaussian kernel parameter.

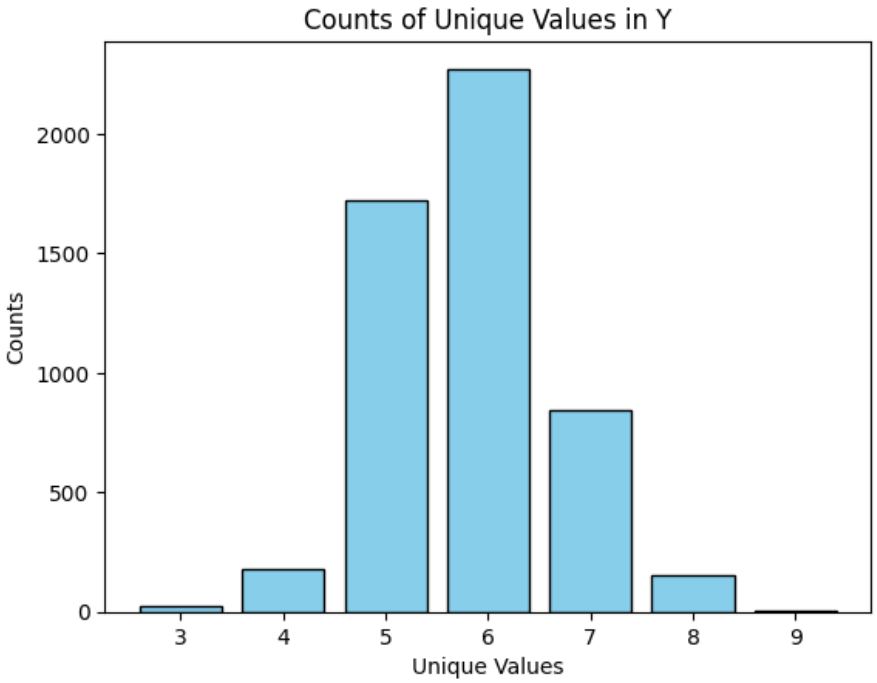
2. Implementation Details:

2.1 solution recognition:

We first started by analyzing the datasets to determine what kind of solution must be provided to solve the problem. The dataset has the following shape:

```
X_train Shape: (5200, 11), Data type: float64
 Y_train Shape: (5200, 1), Data type: int64
```

Next, we visited the Y_train dataset to learn more about the output values. The output dataset has seven different integer values: 3, 4, 5, 6, 7, 8, and 9. Putting this result on a plot we have:



We can see that the problem we are facing is a multiclass classification task with seven classes. From the plot, we also observe that the dataset is not distributed equally. Some classes, like six and five, take up most of the training samples, whereas others, like four, three, or nine, don't have a lot of samples to train the model on. Here, we can expect that going forward, the model will perform better in classes with a higher number of samples.
Given that we are tasked with a KRLS algorithm and seeing the dataset's format, we decided to use the KRLS algorithm to perform multiclass classification. To do this, we will use probabilities of each class.

First, we divide the dataset into different classes based on the output. Then, we will train a KRLS model on each one of these classes (obtain the weights for each class). We use these weights to predict each validation set. The outputs of each class's model are then transformed into probabilities. We choose the maximum probability as the final class each point belongs to.
We will be using Q-fold cross-validation to calculate the optimal hyperparameters for the Kernel and the Regularizer.

2.2 Pre-processing the dataset:

Data Standardization: The features in X are standardized using z-score normalization via StandardScaler from scikit-learn.Each feature is transformed to have a mean of zero and a standard deviation of one.

One-hot encoding the labels: One-hot encoding is necessary for multi-class classification (OvA), where each class is treated as a separate binary classification task. The labels Y (categorical) are encoded into one-hot vectors using the encode_labels function.

2.3 Solution Implementation:

During the implementation phase, we tried to use as many of the hands-on codes as possible. Allowing small changes to fit the codes to our current task. For example, functions like calc_err() and squared_distances() were left intact and used as they were defined originally.
We altered other functions like krls_train (), krls_predict (), and Gaussian_Kernel (). Originally, these functions accepted kernel type as an argument. We altered this since only the Gaussian kernel is being used here.
We added functions like encode_labels () and compute_probabilities () to help us implement the KRLS classification task.
The main two classes are krls_kfoldcv_multiclass () and krls_kfold_valerr_multiclass (). These two classes were taken from Lab 5 hands-on codes and altered to fit the classification task. Here is how each of them works:

Steps to krls_kfoldcv_multiclass:

1. Loops over all combinations of reg_par_list and kernel_par_list.
2. Calls krls_kfold_valerr_multiclass to compute training and validation errors for each combination.
3. Logs the errors for all combinations.
4. Identifies the best combination (lowest validation error).
5. Returns the best parameters, their corresponding error, and the entire error matrices for analysis.

Steps to krls_kfold_valerr_multiclass:

1. Randomly splits the dataset into num_folds equal parts for CV.
2. For each fold:

- Designates one subset as the validation set and the rest as the training set.
- Trains a KRLS model for each class in a one-vs-all (OvA) manner, using the krls_train function.
- Predicts both training and validation scores for all classes using krls_predict.
- Converts scores to probabilities using compute_probabilities.
- Determines the predicted classes and computes errors using calc_err.

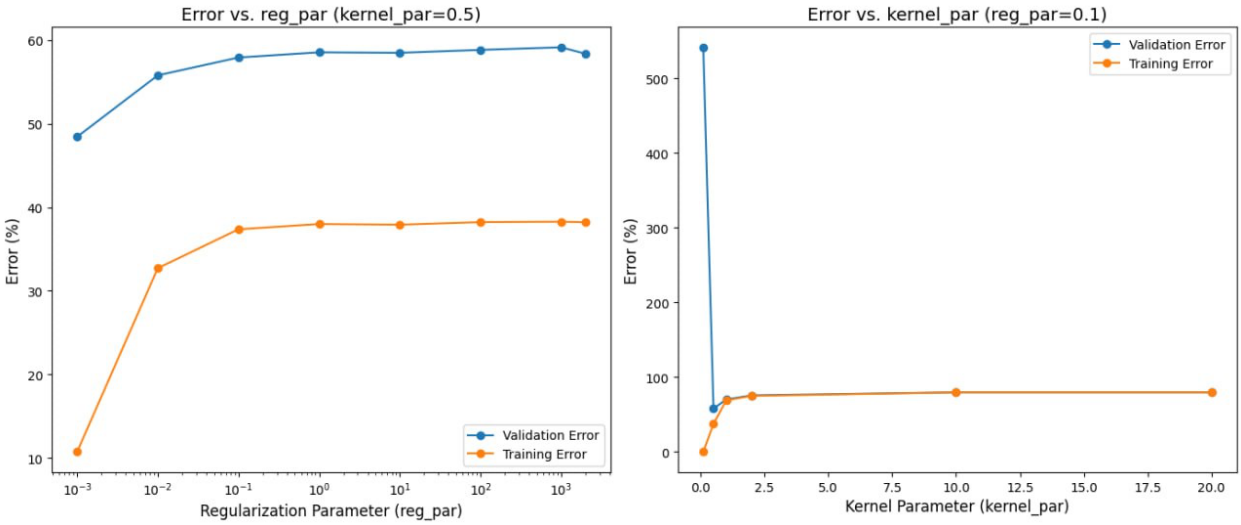
3. Collects errors for all folds and returns them as arrays.

3. Results and Hyperparameters:

At the first try, we chose these lists as the ranges of our hyperparameters:

reg_par_list = [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000, 2000]
kernel_par_list = [0.1, 0.5, 1, 2, 10, 20]

The number of folds was a fixed five. After the training and the validation steps were over, we visualized the results:



Best Regularization Parameter	0.001
Best Kernel Parameter	0.5
Best Cross-Validation Error	48.42%
Best Training Error	0.0%

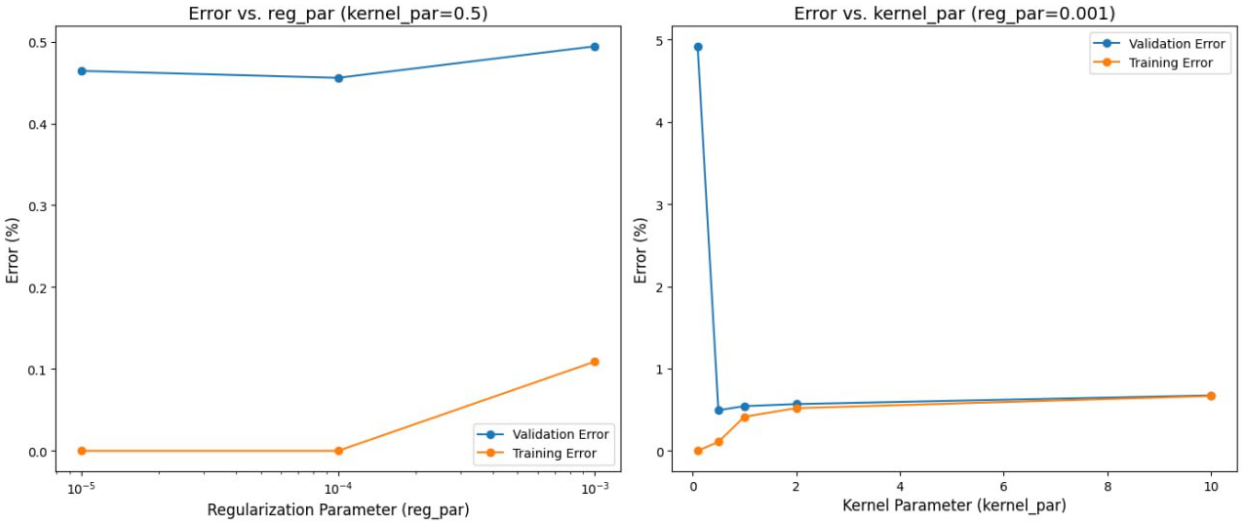
We can see that as both hyperparameters increase after a certain point, the error value remains the same and does not show any significant change. However, at the beginning of the training process, the error values seem to be smaller for smaller hyperparameters.

So, we decided to try hyperparameter ranges that include smaller values:

reg_par_list = [1e-5,1e-4,1e-3]

kernel_par_list = [0.1, 0.5, 1, 2, 10]

Again, with the number of folds being five, we visualized the results:



Best Regularization Parameter	0.0001
Best Kernel Parameter	0.5
Best Cross-Validation Error	45.58 %
Best Training Error	0.0%

This time, we have a lower cross-validation error. From the plots, we can see that there seems to be an optimal value for the Regularization Parameter at 0.0001. For smaller values, the error seems invariable, and for greater values, the error starts rising. The best Kernel Parameter is once again chosen to be 0.5. However, since the training error is at zero percent, the trained model might be overfitted to the training data.

We consider these results as the final optimal hyperparameters.

We trained the model one last time using the optimal hyperparameters and got a result of 44.13% MSE error.

However, in order to ensure that the model was implemented correctly and the results were in line with the best achievable result, we tried a variety of other models and compared our results with these other models:

Model	MSE for validation set
KRLS	44.13 %
Random Forest	46.16%
SVM	120%
Neural Network	58.16%

We can see that the implemented classification using KRLS, outperforms the other models such as Random Forest and NN. And considering the results the other models yield, we can see that our result is in line with the other models using the same dataset.