Lab Course Machine Learning
— Problem Set 3—

Eugeniu Vezeteu, Alberto Gonzalo Rodriguez Salgado

June 25, 2020

Part 1: Implementation

Assignment 1 cross-validation

This is a method of model selection. We want to find the parameters which minimize the generalization error on the whole dataset. For this task we:

- Split dataset in k-folds (10 in our case)
- Iterate of all combinations of parameters
- Train the model on training set, and test it on validation set

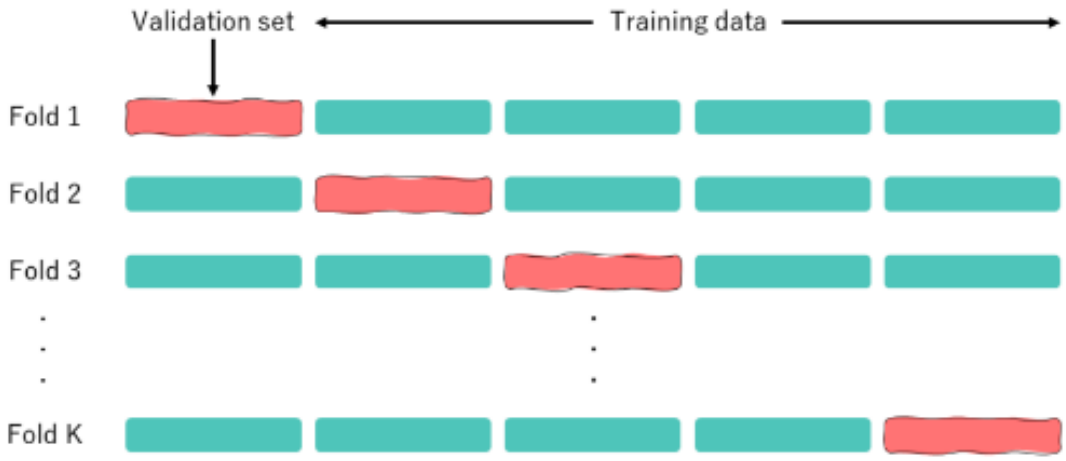


Figure 1: K-Fold dataset split

We compute the average error for each parameters combination, and keep the parameters that correspond to the minimal error.

Result of cross-validation on code stubs test data is provided in in Figure 2

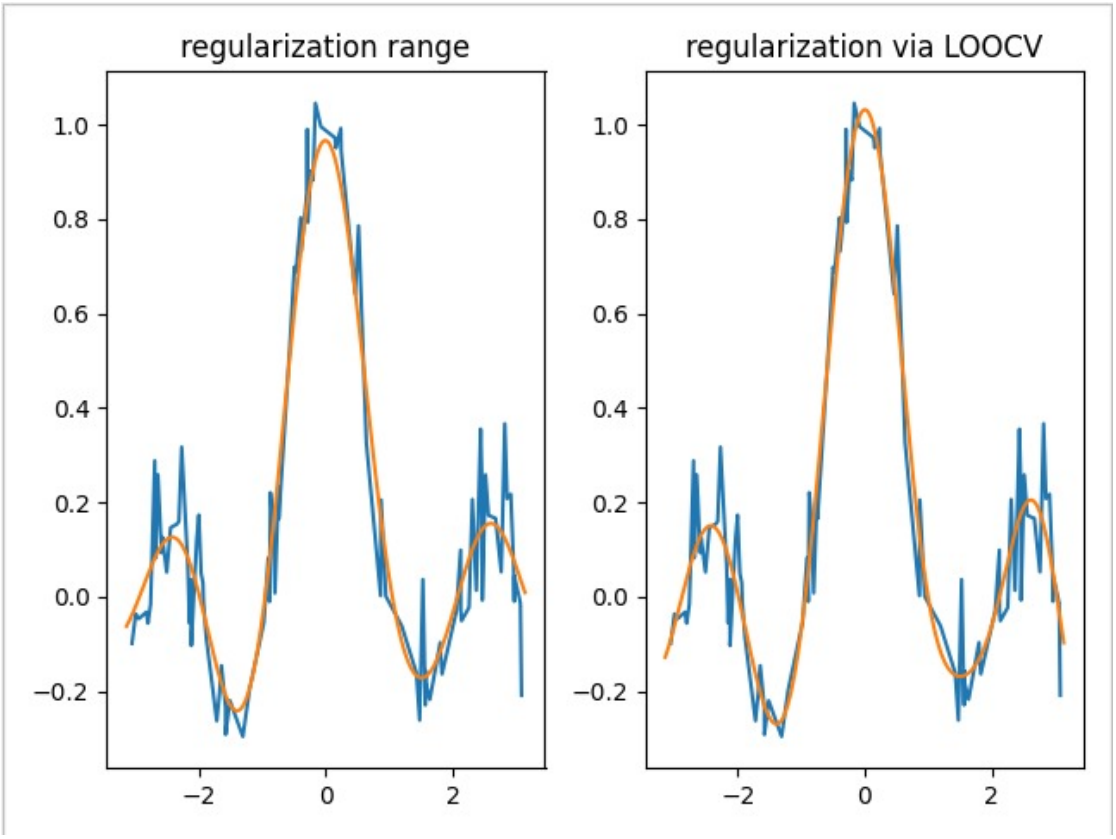


Figure 2: cross-validation result on code stub test data

Assignment 2 Kernel Ridge regression

Kernel ridge regression is a combination of ridge regression with kernel trick. We replaced scalar products between the data points, with the kernel evaluation. We used 3 different kernels:

- Linear kernel: $k(x, x') =< x, x' >$
- Polynomial kernel: $k(x, x') = (< x, x' > +1)^d$
- Gaussian kernel: $k(x, x') = e^{-\frac{||x-x'||^2}{2*\sigma^2}}$

For training we computed the kernel matrix K, and

$$\alpha = (K + C * I)^{-1} * y$$

where C is a regularization term, and y is a vector of true labels, α is an array of learned coefficients of our data.

When the regularization parameter was set to zero, we use efficient LeaveOneOut cross-validation method. LOOCV is the same KFold method, where the number of folds is equal to the number of points. We search the candidates parameters between min and max eigenvalues of the kernel matrix K.

$$K * (K + C * I)^{-1} = U * L * (L + C * I)^{-1} * U^T$$

The LOOCV loss is given by:

$$\epsilon = \frac{1}{n} \sum_{i=1}^{n} (\frac{y_i - [Sy_i]}{1 - S_{ii}})^2$$

For prediction, we compute the kernel matrix between training data and test data, and predict

$$y_{pred} = \alpha * K$$

.
Result on code stubs test data is provided in in Figure 27. We notice that linear kernel was not able to predict the data, while the best prediction where achieved with gaussian kernel. The best result was achieved with LOOCV and gaussian kernel.

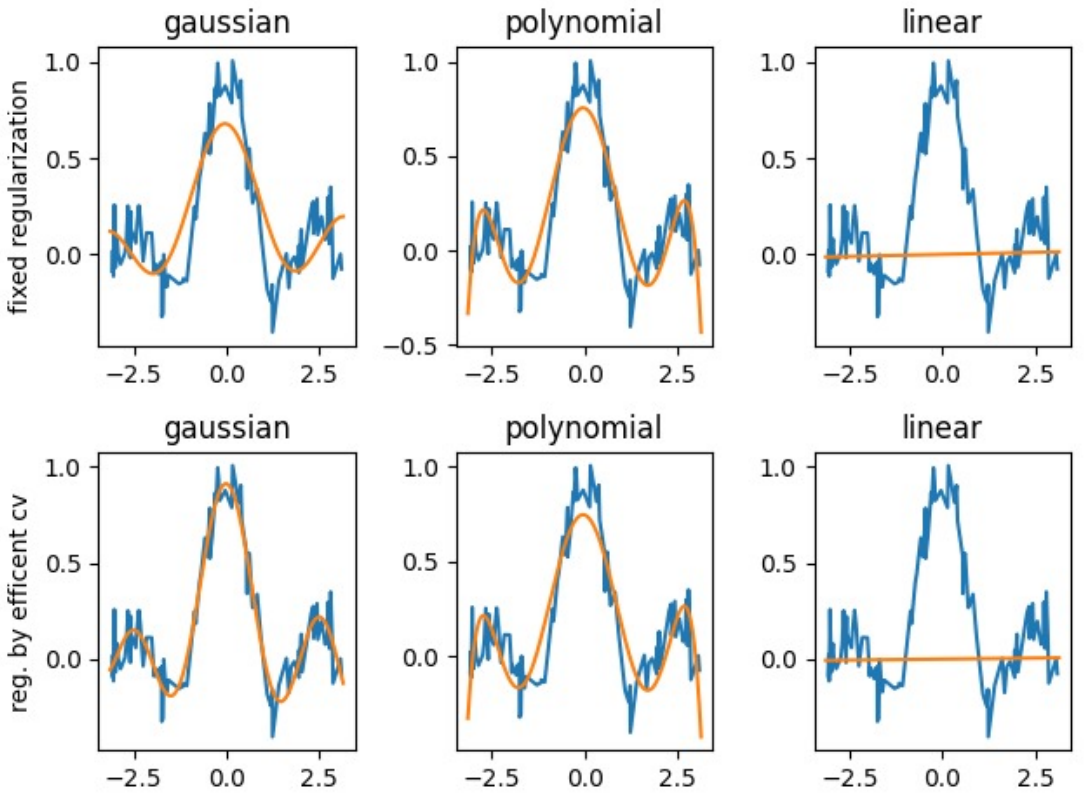


Figure 3: Kernel Ridge regression on code stubs test data

Assignment 3 Kernel Ridge Regression to predict atomization energies

In this assignment, we used our previously implemented KRR (Kernel Ridge Regression) and CV (Cross Validation) functions to predict atomization energies of organic molecules as in (1). For every molecule's coulomb matrix $\boldsymbol{M_i}$, we computed its correspondent eigenvalues $\boldsymbol{\lambda_i}$. These eigenvalues make the data matrix $\boldsymbol{X} \in R^{7165\times 23}$ up which we used for cross validation. The data matrix consists of 7165 molecules with each 23 features which correspond to their eigenvalues in decreasing order. In this case, the labels $\boldsymbol{y} \in R^{7165}$ represent the atomization energy in kcal/mol. At first, we computed all the euclidean distances $||x_i - x_j||$ and energy differences $|y_i - y_j|$ between all molecules and plotted them.

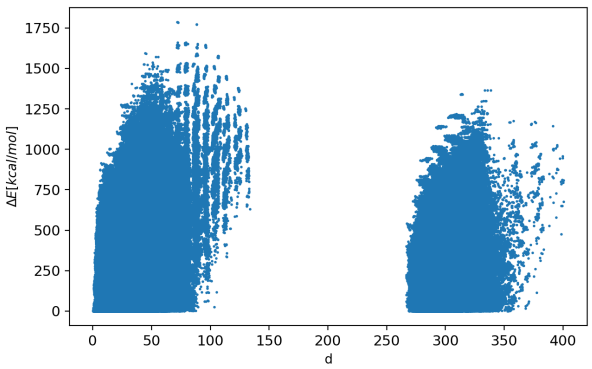


Figure 4: Distances against the absolute difference of energies

Figure 4 shows that there are distances between all data points have a $0-400$ range and the energy difference a $0-1750$ range. Interestingly, we can recognize two groups from figure 4, the first group having distances d in the $0-150$ range and the second group having distances in the $250-400$ range .We "validated" our result by comparing it to the same figure Rupp et al. provided in (1).

We continued by randomly splitting the dataset into a train set $\boldsymbol{X_{train}} \in R^{5000\times 23}, \boldsymbol{y_{train}} \in R^{5000}$ and a test set $\boldsymbol{X_{test}} \in R^{2165\times 23}, \boldsymbol{y_{test}} \in R^{2165}$. In order to obtain the best possible KRR model which does neither overfit or underfit the data, we performed five fold cross-validation with 2500 randomly selected training data from $\boldsymbol{X_{train}}$ and $\boldsymbol{y_{train}}$. The cross validation was performed with

- 5 folds
- 5 repetitions per parameter combination
- Gaussian model and Mean Average Error (MAE) as the loss function
- Gaussian widths : 0.1, 0.5 and 0.9 quantiles of the previously computed distances
- 10 log spaced regularization parameter in the range of 10^{-7} to 10^{0}

After performing the five fold cross-validation with the given parameters, the best parameters are

- Gaussian width: 23.7835 corresponding to the 0.5 quantile
- Regularization C: 2.1544e-05.

With the best parameters, the test error (on the test data set $\boldsymbol{X_{test}}$) is 14.831. Again, this value is close/in the range of error values Rupp et al. obtained in (1).

We continued by fixing the parameters used for KRR and investigating how the test error varied with regard to the number of training samples N that the KRR algorithm uses for training. We used 10 log-spaced values for N between $N = 100$ and $N = 5000$. The training data is extracted from. $\boldsymbol{X_{train}}$ and the KRR model is tested on X_{test}.

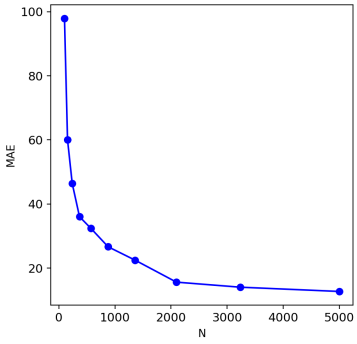
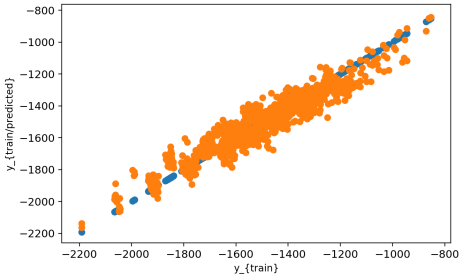


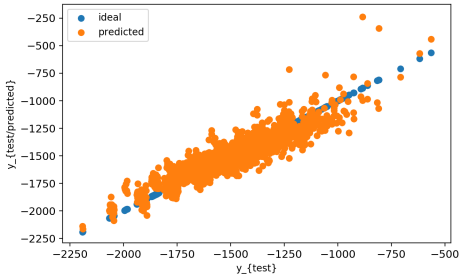
Figure 5: Test Error for the best KRR model over different training sizes N

Figure 5 shows that the MAE (mean average error) on the test set decreases with higher train samples N. The minimum test error is achieved with the highest number of training samples $N = 5000$ wherer $MAE = 11.7$. Similar results were provided by Rupp et al. (1). Therefore, we can conclude that the model we selected via CV generalizes well with more data.

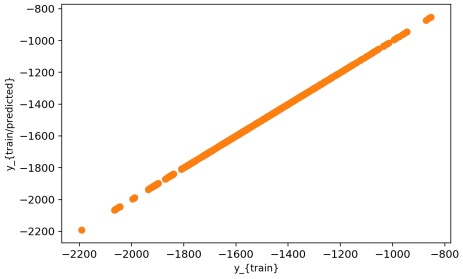
For the last part of this assignment, we investigated different parameter combinations that led to three types of different results. In the first case, we would clearly underfit the data, meaning that both the training MAE and test MAE would be high. In the second case, we would clearly overfit the data, meaning that the training MAE would be very low but the test MAE would still be high. In the last case we would use our optimal parameters meaning that the both the train and test MAEs would be low. In all cases we trained our KRR model with 1000 from $\boldsymbol{X_{train}}$ randomly chosen samples and test the model on the test set $\boldsymbol{X_{test}}$. Furtheremore, we used the optimal guassian width of 23.7835 for all models and only changed the regularization parameter C in order to under- or overfit the training data.



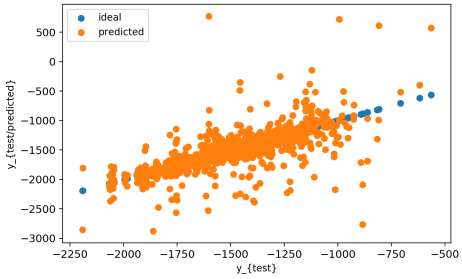
(a) Underfit, Training set, C=1^{-9}



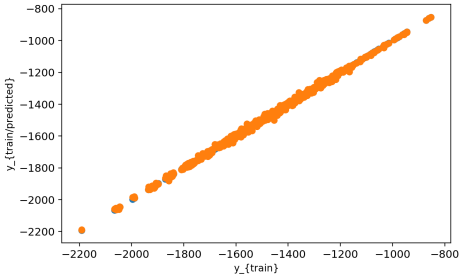
(b) Underfit, Test set, C=1^{-9}



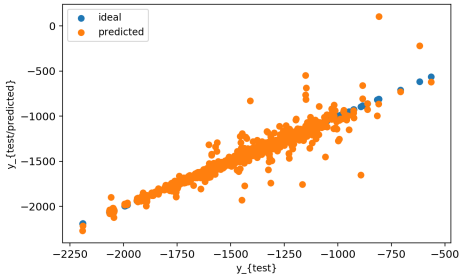
(c) Overfit, Training set, C=2.1^{-2}



(d) Overfit, Test set, C=2.1^{-2}



(e) Optimal fit, Training set, C=2.1544^{-5}



(f) Optimal fit, Test set, C=2.1544^{-5}

Figure 6: Underfitting, Overfitting and optimal fitting models

Table 1: Model errors

Model	**MAE on X_{train}**	**MAE on X_{test}**
Underfit	55.2896	66.771
Overfit	0.062	74.593
Optimal fit	6.439	22.814

The figures 6 and the model MAEs data from table 1 let us conclude the following:

- As expected, the underfit model has a high train and test MAE of 55.2896 and 66.771 respectively. Figures 6a and 6b shows that neither the test or training sets are well fited.
- As expected, the overift model has a very low train and a high test MAE of 0.062 and 74.593 respectively. Figure 6c shows that the training set is almost perfectly fitted but the model fails to generalize on new data. This poor performance on the test data can be observed in figure 6d.
- The optimal model has a higher MAE of 6.439 for the training set then the overfited model but is able to better generalize for new/unseen data. The MAE on the test set is 22.814, the lowest value among the three models. This phenomenon can be observed in figures 6e and 6f.

Assignment 4

All tests for this assignment were performed on Intel core i7 8th Generation machine with 8GB RAM.
In this assignment we tested LOOCV and CV for 5 different datasets. Datasets details are presented in Table 2.
We used the following CV config:
-NUMBER OF REPETITION = 5
-K-Fold = 10

Table 2: Datasets details.

Data set	**X train**	**Y train**	**X test**	**Y test**
banana	(400, 2)	(400, 1)	(4900, 2)	(4900, 1)
diabetis	(468, 8)	(468, 1)	(300, 8)	(300, 1)
flare-solar	(666, 9)	(666, 1)	(400, 9)	(400, 1)
image	(1300, 18)	(1300, 1)	(1010, 18)	(1010, 1)
ringnorm	(400, 20)	(400, 1)	(7000, 20)	(7000, 1)

Parameters candidates for each CV and LOOCV are presented in Table 3

Table 3: Searching space for LOOCV and CV

Method	**kernel**	**kernel parameter**	**regularization**
LOOCV	polynomial	[1,2,3,4,5]	0
	gaussian	[.1, .5, .9, .95, 1.0]	0
	linear	0	0
CV	polynomial	[1, 2, 3, 4, 5]	np.logspace(-2, 2, 10)
	gaussian	[.1, .5, .9, .95, 1.0]	np.logspace(-2, 2, 10)

LOOCV results are presented in Table 4. For LOOCV method we used *squared error* as loss function. CV results are presented in Table 5. For CV method we used 3 different error functions 1)*squared error*, 2)*mean absolute*, 3)*zero-one loss*
For each method (LOOCV, CV) and for each loss function, we run the function with 5 number of repetition and k-fold=10, then we used best parameters and predicted TPR and FPR (see roc_fun) averaged over all folds and iterations, and computed AUC value.

Table 4: LOOCV best results

Data	**cvloss**	**testloss**	**kernel**	**kernel parameter**	**regularization**	**AUC**	**Avg.Time**
banana	0.33	0.30	gaussian	0.5	0.93	0.96	1m 0s
diabetis	0.67	0.63	polynomial	1	24.16	0.80	2m 15s
flare-solar	0.82	0.77	polynomial	1	3.37	0.69	3m 2s
image	0.12	0.15	gaussian	1	0.38	0.99	11m 31s
ringnorm	0.68	0.72	linear	0	68.15	0.82	1m 41s

Table 5: CV best results

loss function	Data	cvloss	testloss	kernel	parameter	reg.	AUC	Avg.Time
squared error	banana	0.32	0.30	gaussian	0.5	0.21	0.96	8m 12s
	diabetis	0.67	0.63	polynomial	1	4.64	0.80	14m 13s
	flare-solar	0.82	0.77	polynomial	1	0.59	0.69	21m 10s
	image	0.11	0.13	gaussian	1	0.1	0.99	40m 10s
	ringnorm	0.56	0.66	polynomial	2	100	0.91	11m 12s
mean absolute	banana	0.37	0.35	gaussian	0.5	0.07	0.96	8m 2s
	diabetis	0.68	0.66	polynomial	1	0.01	0.80	14m 48s
	flare-solar	0.83	0.77	gaussian	0.95	0.21	0.675	21m 45s
	image	0.19	0.21	gaussian	1	0.1	0.99	43m 1s
	ringnorm	0.63	0.66	polynomial	2	35.93	0.91	12m 8s
zero-one loss	banana	0.098	0.1	gaussian	0.95	0.01	0.96	8m 17s
	diabetis	0.24	0.21	polynomial	1	100	0.80	13m 58s
	flare-solar	0.34	0.31	polynomial	1	1.66	0.69	20m 35s
	image	0.02	0.03	gaussian	1	0.16	0.99	44m 12s
	ringnorm	0.13	0.15	polynomial	2	35.93	0.9	14m 41s

Banana data set

- *LOOCV*

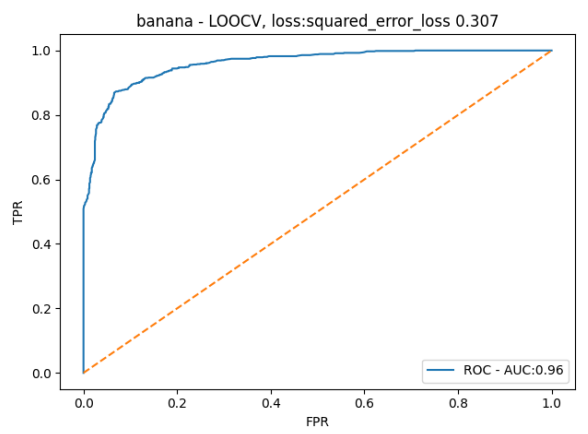


Figure 7: Banana AUC 0.96

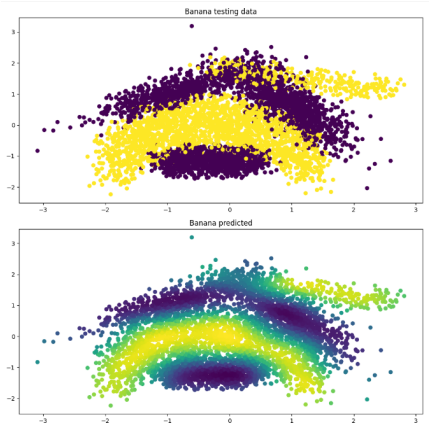


Figure 8: Banana prediction, loss=0.3

Using LOOCV we got the cvloss=0.33 and test loss = 0.3. The best kernel is gaussian with .5 parameter and .93 regularization, AUC value is 0.96, see Table 4.

- *CV*
  For this method we got the same AUC for all 3 loss functions, 0.96 and same kernel (gaussian). However, the smallest error is performed with zero-one-loss function, 0.098 for cvloss and 0.1 for test loss, see Table 5.

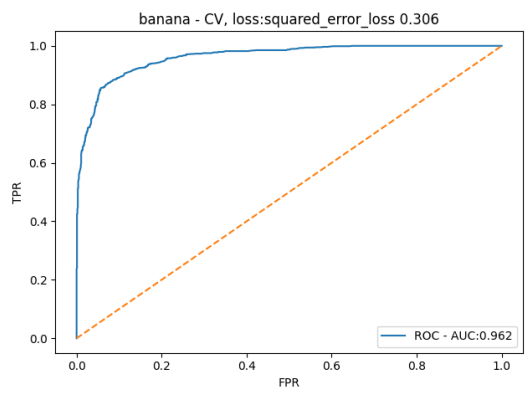


Figure 9: Banana AUC 0.96 with squared error

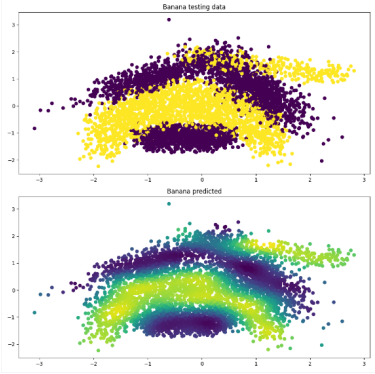


Figure 10: Banana prediction, loss=0.3 with squared error

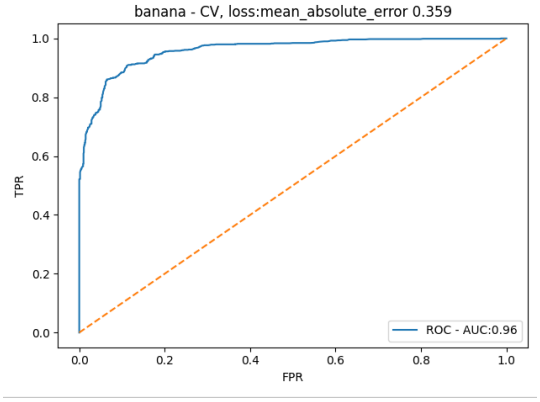


Figure 11: Banana AUC 0.96 with mean absolute error

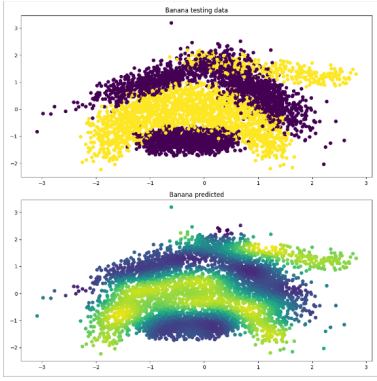


Figure 12: Banana prediction, loss=0.35 with mean absolute error

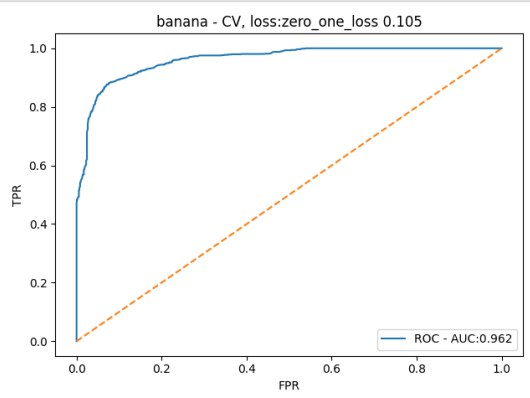


Figure 13: Banana AUC 0.96 with zero-one loss

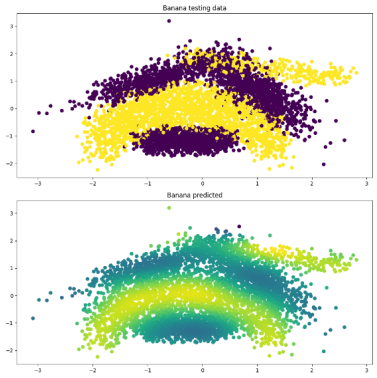


Figure 14: Banana prediction, loss=0.1 with zero-one error

Diabetis data set

- *LOOCV*
  Using LOOCV we got the cvloss=0.67 and test loss = 0.63 . The best kernel is polynomial with 1 degree and 24.16 regularization, AUC value is 0.80, see Table 4.

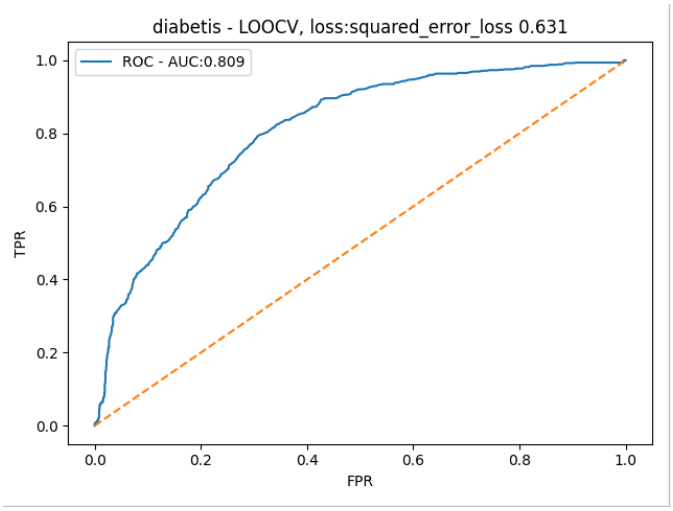


Figure 15: Diabetis data set AUC:0.80

- *CV*
  For this method we got the same AUC for all 3 loss functions, 0.80 and same kernel (polynomial). The smallest error is performed with zero-one-loss function, 0.24 for cvloss and 0.21 for test loss, see Table 5.

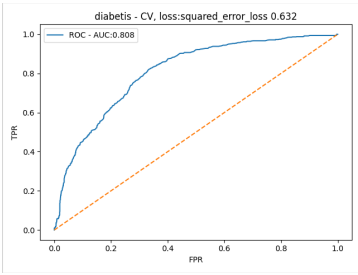


Figure 16: squared error: AUC-0.80

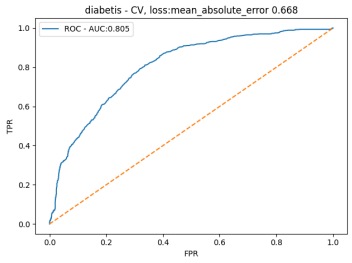


Figure 17: mean absolute error: AUC-0.80

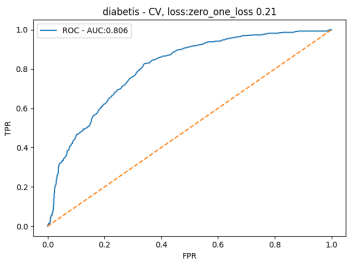


Figure 18: zero-one error: AUC-0.80

Flare-solar data set

- *LOOCV*
  Using LOOCV we got the cvloss=0.82 and test loss = 0.77 . The best kernel is polynomial with 1 degree and 3.37 regularization, AUC value is 0.69, see Table 4.

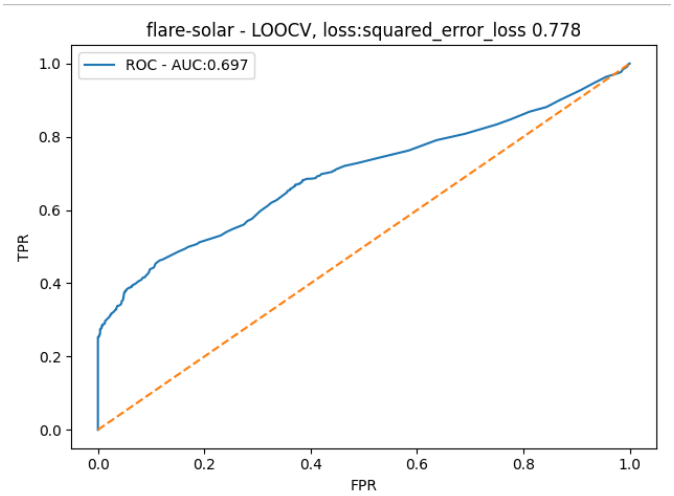


Figure 19: Flare-solar data set AUC:0.69

- *CV*
  For this method we got the best AUC 0.69, with polynomial kernel. The smallest error is performed with zero-one-loss function, 0.34 for cvloss and 0.31 for test loss, see Table 5.

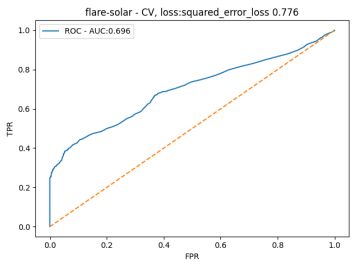


Figure 20: squared error: AUC-0.69

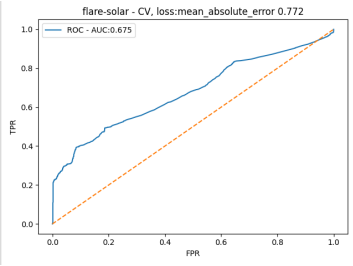


Figure 21: mean absolute error: AUC-0.67

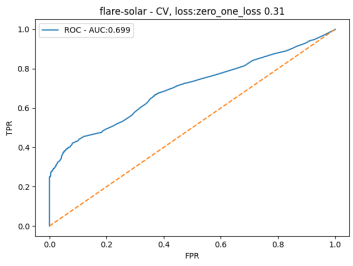


Figure 22: zero-one error: AUC-0.69

Image data set

- *LOOCV*
  Using LOOCV we got the cvloss=0.12 and test loss = 0.15 . The best kernel is gaussian with parameter 1 and 3.38 regularization, AUC value is 0.99 , see Table 4.

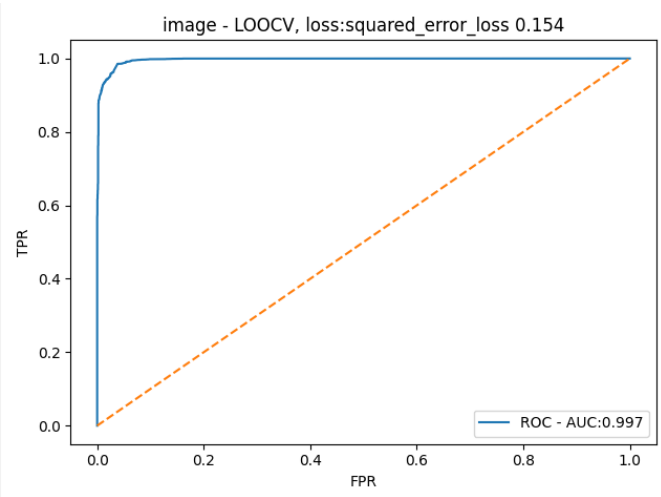


Figure 23: Image data set AUC:0.99

- *CV*
  For this method we got the same AUC 0.99, the smallest error is performed with zero-one-loss function, 0.02 for cvloss and 0.03 for test loss with gaussian kernel, see Table 5.

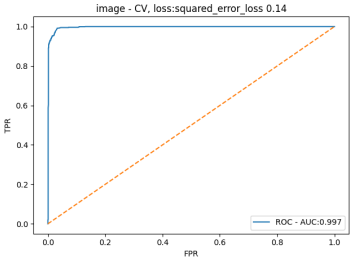


Figure 24: squared error: AUC-0.99

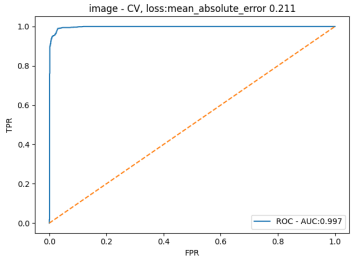


Figure 25: mean absolute error: AUC-0.99

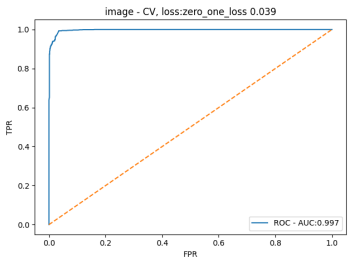


Figure 26: zero-one error: AUC-0.99

Ringnorm data set

- *LOOCV*
  Using LOOCV we got the cvloss=0.68 and test loss = 0.72 . The best kernel is linear, AUC value is 0.82, see Table 4.

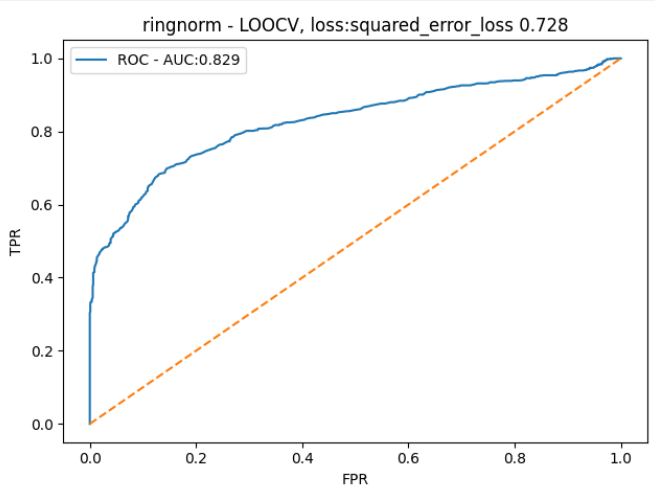


Figure 27: Ringnorm data set AUC:0.82

- *CV*
  For this method we got the best AUC 0.91, the smallest error is performed with zero-one-loss function, 0.13 for cvloss and 0.15 for test loss with polynomial kernel, see Table 5.

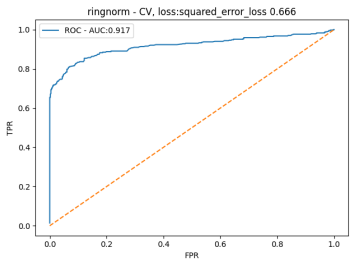


Figure 28: squared error: AUC-0.91

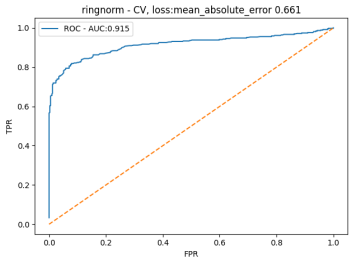


Figure 29: mean absolute error: AUC-0.91

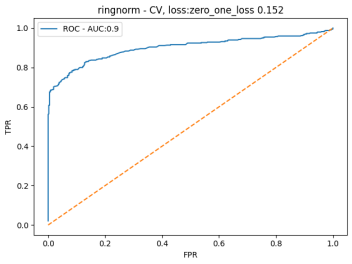


Figure 30: zero-one error: AUC-0.90

Conclusion:

- There is no universal kernel that is able to fit all types of data sets, kernel type and parameters should be adjusted based on data set.
- In our case CV version outperformed (bigger AUC, smaller loss) efficient LOOCV method, but CV version requires more computational time.
- Almost for all datasets, except for ringnorm, the smallest loss were performed with zero-one loss function.
- For the flare-solar dataset we noticed that at some point, $(K + C * I)$ matrix is singular (in theory this should not happen, but we noticed that it happened for a single candidate for flare-solar dataset), therefore, cannot be inverted. This happened due to bad candidates or data features. For this case we added an extra small hard-coded regularization term, also we can define *fitted* parameter for our *krr* class, and set it to False, when this issue occur.

References

[1] M. Rapp, A. Tkatchenko, K. Müller, O. Lilienfeld, *Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning* arXiv:1109.2618v1 [physics.chem-ph] 12 Sep 2011