

Summer term 2016

## Problem Set 3: Kernel Ridge Regression, Cross-validation

### Part 1: Implementation

#### Assignment 1 (20 points)

Implement cross validation as a general function, which can be used for various methods and objective functions.

```
method = cv(X, y, method, parameters, nfolds, nrepetitions, loss_function)
```

The arguments have the following definitions:

- $X$  is a  $(n \times d)$ -array (matrix) of data.
- $y$  is a length  $n$  array (vector), which contains the labels  $\in \{-1, 1\}$  or regression targets for every data point.
- **method is a class which has the following functions:**
  - `fit(X, y)` trains the object instantiated with the given parameters with data points  $X$  and labels  $y$ .
  - `predict(X)` returns the predictions.
- Parameters is **a dict with parameter names as keys**. The dict values are lists of cross-validation value ranges. **value\_range**. **Cross validation should be carried out for all possible parameter combinations.**
- **nfolds** is the number of partitions ( $m$  in the notes). This parameter should be optional with a standard value of 10.
- **nrepetitions** is the number of repetitions ( $r$  in the notes). This parameter is optional with the standard value 5.
- **loss\_function is a function** handle to the loss function to be used. It should have the following signature: `l = loss_function(y_true, y_pred)` where `y_true` are the true labels and `y_pred` are the predicted labels. The predicted labels may be real numbers, where positive numbers correspond to label '1' and negative numbers correspond to label '-1'.

Write the loss function `zero_one_loss` which returns the classification error **as a number between 0 and 1**. `zero_one_loss` should be used as the default loss function, if the optional parameter `loss_function` is not specified.

The function **cv should return a method object**, which is an instantiated classifier class that has been trained with the optimal parameter values. In addition, it should have an attribute `method.cvloss` containing the cross validated loss.

The function should report the progress of the function on the command line and also give an estimate for the remaining run time. **(see time.time)**.

If there is only one parameter combination, the function `cv` should not search for a minimum, instead it should calculate the average loss function output for all repetitions and folds. This will come in handy for the generation of the ROC curves (Assignment 4).

**For the iteration over the parameter set, you might want to use `itertools.product`.**

#### Assignment 2 (20 points)

Implement Kernel Ridge Regression as

```
class krr(kernel, kernelparameter, regularization)
```

which has the functions `fit(X,y)` with

- $X$ :  $(n \times d)$  array of features
- $y$ : length  $n$  array of labels

and `predict(X)` with

- $X$ :  $(n' \times d)$  array of features

. The predict function should return the predicted labels as a length  $n'$  array.

The following kernels (with the accompanying parameters) should be implemented:

| Name              | Kernel                                   | Parameter                         |
|-------------------|--|-----------------------------------|
| <b>linear</b>     | $k(x, z) = \langle x, z \rangle$         | (none)                            |
| <b>polynomial</b> | $k(x, z) = (\langle x, z \rangle + 1)^p$ | degree $p \in \{1, 2, 3, \dots\}$ |
| <b>gaussian</b>   | $k(x, z) = \exp(-\ x - z\ ^2 / 2w^2)$    | kernel width $w$                  |

Here  $d$  is the dimension of  $X$ .

The Parameter **regularization** is the regularization constant,  $c$  in  $\hat{\alpha} = (K + cI)^{-1}y$ . If **regularization** is zero, execute Leave-One-Out cross validation on  $c$  efficiently (see handbook). Use logarithmically spaced candidates around the mean of the eigenvalues of the kernel matrix  $K$  for  $c$ .

## Part 2: Applications

### Assignment 3 (25 points)

We consider a simple 1D-toy data set with two classes. Each class is normal with **standard deviation zero**, priors are identical. There is only a difference in the mean:

$$\begin{aligned} p(x|y = -1) &\sim \mathcal{N}(\mu = 0, \sigma^2 = 1) \\ p(x|y = +1) &\sim \mathcal{N}(\mu = 2, \sigma^2 = 1) \\ p(y = -1) &= 0.5 \\ p(y = +1) &= 0.5 \end{aligned}$$

Our classifier is the **simple linear classifier**  $f_{x_0}$  (see handbook):

$$f_{x_0}(x) = \begin{cases} -1 & : x \leq x_0 \\ +1 & : x > x_0 \end{cases}$$

Because we know the true distribution of the data, we can calculate the ROC curve analytically.

**Plot the analytic ROC curve and the empirical ROC curve for sample size  $n$  in a single graph.**

For the analytical ROC curve use the probability distributions given above. For the empirical ROC curve, draw  $n$  data points from the **unconditional distribution**.

In your the analysis, compare analytical and empirical curves for **different sample sizes**.

**Do not forget to hand in a description of how you computed the analytical receiver operator curve.**

### Assignment 4 (35 points)

Download the classification data sets from ISIS. **Apply KRR with efficient leave-one-out cross-validation**

- Generate a dictionary **results** which contains one dictionary for each of the data sets. These dictionaries have the following fields with the results **for the best classifier**: the cross-validated loss **cvloss**, the kernel **kernel**, the kernel parameter **kernelparameter**, the regularization strength **regularization** and the predicted labels **ypred** for the test data.

For example, you could have `results['banana']['kernel'] = 'gaussian'`. Save the dictionary **results in the file results.p using pickle.dump.**

- Plot the ROC-curves for Kernel-Ridge Regression resulting from variation of the bias term, i.e. the constant term (independent of the data  $x$ ) in the prediction function  $f(x)$ . Proper cross-validation has to be performed here!

**Hint:** the easiest way to to this is to define a function **roc.fun** which calculates TPR and FPR for a set of biases. Then supply into **cv** the optimal parameters and **roc.fun** as a loss function.

- Also report the AUC and the cross-validated classification errors for each of the data sets. Is there a correspondence between the classification errors and the ROC curve/AUC?
- Note that the efficient cross-validation of the regularization uses a squared error loss instead of the zero-one-loss. Is there a difference in performance compared to using **cv** to optimize **regularization**?