Machine learning lab course (Praktikum)

Sommersemester 2014

Fachgebiet Maschinelles Lernen
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Problem set 3: Kernel Ridge Regression, Cross-validation

On the ISIS page you can download `ps3_testsNstubs.zip` containing the files `ps3_implementation.py`, `ps3_application.py` and `ps3_tests.py`. Do not modify the names of the files or the names of the functions within these files. You are free to define additional functions within the given files. Make sure that your functions have the correct signatures.

The file `ps3_tests.py` is designed to help you debug your code. It contains test functions for each of the implementation assignments in Part 1, including runtimes of *my* code on *my* machine for comparison. Make sure that your code passes all tests. Be aware that

- a passed test does not guarantee correctness for all possible inputs
- if the test module produces a plot, you have to check if the plot looks correct.
- if your code is very slow compared to the given runtimes, you should optimize your code

You have to submit via ISIS

1. a report (pdf) with your analysis
2. the implementation file `ps3_implementation.py`
3. the application file `ps3_application.py` or an Ipython notebook `ps3_application.ipynb`
4. a file `results.p` containing your results from assignment 4.

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, { param_name, value_range, ... }, nfolds, nrepetitions,
                                   loss_function)
```

The arguments have the following definitions:

1. `X` is a $(d \times n)$-array (matrix) of data.
2. `y` is a $(1 \times n)$-array (vector), which contains the labels $\in \{-1, 1\}$ or regression targets for every data point.
3. `method` is a `class` which has the following functions:
   - `fit(X, y, param1, param2, ...)` trains the method with data points `X`, labels `y` and the given parameters. It returns the method, now containing all relevant information for the application of the method.
   - `predict(X)` uses the model fit and returns the method, now containing an additional field `ypred` of predictions.
4. A list consisting of parameters `param_name` and lists of value ranges `value_range`. Cross validation should be carried out for all the ranges of possible parameters. The sequence of parameters in this list must correspond to their position in the `fit`-function of method `method`.
5. `nfolds` is the number of partitions (m in the notes). This parameter should be optional with a standard value of 10.
6. `nrepetitions` is the number of repetitions (r in the notes). This parameter is optional with the standard value 5.
7. `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 `method`, which 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(X, y, kernel, kernelparameter, regularization)
```

which has the functions `fit(X,y,kernel,kernelparameter,regularization)` and `predict(X)`. The following kernels (with the accompanying parameters) should be implemented:

Name	Kernel	Parameter
`linear`	$k(x,z) = \langle x, z\rangle$	(none)
`polynomial`	$k(x,z) = (\langle x, z\rangle + 1)^p$	degree $p \in \{1, 2, 3, \ldots\}$
`gaussian`	$k(x,z) = \exp(-\|x - z\|^2/2w^2)$	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):

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

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

```
roc_curve(n)
```

which plots 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.

Assignment 4 (35 points)

Download the classification data sets from ISIS. Write the function `apply_krr` which uses `krr` with efficient cross-validation on each of the datasets and finds the best classifier by cross-validation. Then apply this classifier to obtain labels for the test data.

- 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`?