Machine Learning, Spring 2017: Exercise Sheet 6&7&8

Please send your type-set solutions by email to our two TA's Xu He ("Owen") x.he@jacobs-university.de and Felix Schmoll f.schmoll@jacobs-university.de. Join into groups of two or three and submit a single solution sheet per group, indicating the group members' names on the sheet.

Deadline for submission is Friday April 21, 23:59 hrs (email sending timestamp). Submissions arriving later (even a second after midnight) will be corrected but not counted for the course grade.

Outline of this exercise. This "exercise" is indeed a miniproject, more heavyweight than previous exercise sheets. Its grade will be counted with a weight factor of 3, and you have 2.5 weeks time for it (use them!). – Again the task is to train a classifier for our well-known digits data. But this time you will be using some additional tricks (described below) and a careful cross-validation to lift the test misclassification rate into a range significantly better than 5% (I hope). Maybe you even can get close to the state of the art on this dataset (about 1.8% test error rate was the best I saw).

The classifiers (decision functions) $D: \mathbb{R}^n \rightarrow \mathbb{R}^k$ from the lecture notes, which were based on first using PCA to extract some PC features and then use linear regression, had one crucial shortcoming: they were (affine) *linear* maps. As a consequence, these classifieres can only realize decision regions that are convex polytopes. This geometric fact limits their possible test accuracy in scenarios where the optimal decision regions have other shapes – non-convex and/or with curved boundaries (check again Section 4 in the lecture notes for a refresher on the concept of decision regions). In this exercise you will be introduced to, and be training, a very practical and easy-to-handle *nonlinear* classifier known as *radial basis function network* (called just RBF networks by experts).

The blueprint of an RBF network is easily described. Like our familiar PC-feature based classifiers, it is a 2-stage procedure which in the first stage extracts m-dimensional feature vectors $\mathbf{f}(x) \in \mathbb{R}^m$ from n-dimensional input patterns $x \in \mathbb{R}^n$. In the next and final stage, these feature vectors $\mathbf{f}(x)$ are linearly transformed into k-dimensional hypothesis vectors $z \in \mathbb{R}^k$. This linear transformation from $\mathbf{f}(x)$ to z is computed by (regularized) linear regression. All of this exactly follows the scheme described in Sections 6 and 7 of the lecture notes.

The new element is how the features are defined. They will be decidedly *nonlinear* functions of patterns x. We will call them *RBF features*. There is a Wikipedia article on "radial basis function networks", and I will devote the Friday class to explain them, but this exercise sheet should supply all information you need.

We will admit to extract more features than x has dimensions, that is, we allow $m > n$. The original intuition about features, namely that they serve to reduce dimensions, is abandoned. In fact, you will see how easily you can overfit with RBF networks. Instead, the new intuition is that features *create nonlinearities* – in a sense, the more of that the better. Overfitting can be prevented in three ways:

- feature softening: features can shaped by a shape parameter β which makes them more (overfitting danger) or less (underfitting danger) "curved"
- number of features: one can use small numbers m of RBF features (underfitting danger) or large numbers (overfitting danger)
- regularizing the final linear regression by ridge regression (Section 8.7 in the lecture notes)

Which one of these you use – or which combination of them – is up to your experimental diligence.

So far, so good – but what *is* an RBF feature? Recall that a *feature* is any function f: $\mathbb{R}^n \rightarrow \mathbb{R}$. The basic format of an RBF feature is

$$f(x) = \exp\left(-\frac{\|x-c\|^2}{\beta} \right).$$

If you think a little about this function you will see that it has the same shape as a circular n-dimensional Gaussian bell, with a center $c \in \mathbb{R}^n$ and a width that is growing with β (this parameter is always positive). Different from the Gaussian pdf, the integral of f is not normalized to 1, but instead, its peak value (attained at $x = c$) is 1. The fact that this function is perfectly symmetric around its center c has given the entire approach its name, "radial basis function" networks.

An RBF feature is thus characterized by its center c and by its width parameter β. To simplify the notation we also write $\exp(-\beta^{-1}\|x-c\|^2) = f(x \mid c, \beta)$.

RBF features act as "proximity sensors" in pattern space $\mathbb{R}^n$. When x is far away from c, $f(x \mid c, \beta)$ will be almost zero – the pattern x is not "sensed" by the feature $f(x \mid c, \beta)$. When x comes closer to c, the "sensing signal" $f(x \mid c, \beta)$ rises, to a maximal value of 1. The width parameter β regulates how far the sensor's sensitivity reaches out around its center.

Whether an RBF network functions well for classification obviously depends on the choice of RBF features, which in turn boils down to a choice of their centers and widths. One can invest loads of effort and theory into playing this game well, but quite simple design strategies also often work. A popular procedure for finding good RBF features goes like this:

1. First, carry out a cluster analysis on the training patterns $(x_i)_{i=1,...,N}$, for instance, using K-means clustering (see Section 5.1 in the lecture notes). Use the K codebook vectors μ_i to make K centers $c_i = \mu_i$ for $m = K$ RBF features.
2. Use a single width β for all features. This lets you end up with K-dimensional feature vectors $\mathbf{f}(x) = (f(x \mid \mu_1, \beta), ..., f(x \mid \mu_K, \beta))'$.
3. Use β to tune the model flexibility. Small values of β give sharply peaking feature functions, large values give soft and broad ones (compare Figure 21 from the lecture notes for a graphical impression).

OK., this is basically all you need for starters. Here is your assignment:

Step 1: Split the available digits data into a training set (100 examples per digit) and a testing set (the other 100 examples per digit). Hide the latter in a secrete place. Use only the training data for all subsequent steps except the very last one.

Step 2: Set up a programming environment for specifying and training RBF networks, such that you can easily modify core design parameters (like number of RBF features, their centers, the width parameter β, or the regularization parameter α for ridge regression) train a model with those parameters, and test it on validation data in a cross-validation scheme

Step 3: Use cross-validation to optimize the validation risk of your model(s). Try to reach a validation error rate below 5%.

Step 4: If you are satisfied with your model – in real life this would mean: if you have decided to package it and deliver it to your customer – then (and only then) test it on the 1,000 test images that you hid in a secret place in Step 1. You should use the test data only once in this entire project, namely for this final evaluation.

Document your procedures, optimization strategies, cross-validation setup, and noteworthy experiences in a nice typeset documentation (target size: 2 pages text, plus possibly graphics) and submit this together with your code to Owen and Felix.

Notes on our grading rules: We base our grading on the written documentation, not on the code. We use your code only as fallback source of information when your documentation is unclear. Still, even when we do not do code-checking, poorly documented code that the TA's have a hard time to understand leads to point reduction. – On the other hand, if you invest inspiration and dedication and a very nice miniproject report results, we may give extra bonus points, up to 5. These bonus points are added to your course point score at the time of course grade calculation (that is, 5 bonus points will lift the course grade by 0.33).