Homework 5 for Pattern Recognition

Fan JIN (2015011506)

April 24, 2018

Question 1.1

The optimal hyperplane satisfies

$$f(x) = \sum_{i=1}^{n} \alpha_i^* y_i K(x, x_i) + b = 0.$$

- **(b)(d)(f) are of Gaussian kernels.** This is obvious since the support vectors are distributed not only near the hyperplane, but also on the other side away from the hyperplane. This is a property of the Gaussian kernels. Moreover, **(d)** corresponds to $\sigma = 0.1$ because it tends to overfit the positive samples. **(b)** must be with $\sigma = 1$, as it has a straighter separate line. And **(f)** is in the middle, with $\sigma = 0.5$.
- **(c) is of linear kernels.** The hyperplane would follow a linear function of x if the kernel $K(x, x_i)$ is linear. Thus, the straight line indicates a linear kernel in **(c)**.
- **(a) is of quadratic kernels.** The The hyperplane would follow a quadratic function of x if the kernel $K(x, x_i)$ is quadratic. This can be shown using the fact that

  $$(w^T x)^2 = (w^T x)^T (w^T x) = x^T (w w^T) x,$$

  which is a quadratic form. The parabola thus indicates a quadratic kernel in **(a)**.
- **(e) is of cubic kernels.** Similar to the quadratic kernels, a cubic hyperplane may come from the linear combination of many cubic terms with respect to x.

Question 1.2

We prefer the linear kernel since the samples are linearly separable. The linear kernel, as long as feasible, means *mathematical simplicity, intuitive interpretability, simple calulation,* as well as *property of superpostion.* For example, the linear dot product can be interpreted as projection to a certain direction in the space, while nonlinear ones hardly have such an intuitive demonstration.

Question 2

SVM-based classifiers

Thanks to the "Generate Code" function of the Classification Learner toolbox in MATLAB, I saved the code of all the 6 SVM kernels in 6 function files: Linear, Quadratic, Cubic, Gaussian Fine ($\sigma = 0.1$), Gaussian Medium ($\sigma = 0.5$), and Gaussian Coarse ($\sigma = 1$). See Table 1 for the results.

FCNN-based classifiers

In homework 4, we tried different sizes of the hidden layer, ranging from 5, 10, 20, 40, to 100. Here is a copy of our conclusion in homework 4:

- **Confusion matrix:** With more hidden nodes, the confusion rate drops accordingly. See the table below.
- **Performance:** With more hidden nodes, the error rates drop at both validation set and testing set. Meanwhile, it takes a longer time before it stops and converges, in spite of fewer iterations. Another interpretation: The network has better capability of fitting the pattern with more hidden nodes, and therefore the training process is smoother and we need fewer iterations in training. But it takes more time since we have much more parameters to train, which leads to a longer time for each iteration step.
- **ROC curve:** With more hidden nodes, the ROC curve is closer to the left top corner, which indicates a better performance of classification.

It is obvious that a size of 100 hidden nodes makes the best performance. The accuracy on the testing set is attached to Table 1.

Method	Testing accuracy	Training speed
Linear SVM	0.975389	1~3 mins
Quadratic SVM	0.982923	1~3 mins
Cubic SVM	0.983928	1~3 mins
Gaussian Fine SVM	0.546459	1~3 mins
Gaussian Medium SVM	0.976896	1~3 mins
Gaussian Coarse SVM	0.967855	1~3 mins
Fully Connected NN	0.979407	30 secs
Logistic Regression	0.952788	5~10 mins
Naive Bayes Classifier	0.760924	5 secs

Table 1: Accuracy on testing set

Hidden nodes	Total confusion rate
5	11.5%
10	8.9%
20	5.6%
40	2.9%
100	1.9%

Table 2: Confusion rates

Hidden nodes	Total training epochs
5	500
10	185
20	125
40	125
100	110

Table 3: Training iterations

Logistic-regression-based classifiers

We use the logistic regression classifier in the Classification Learner toolbox. The accuracy on the testing set is attached to Table 1.

Naive Bayes classifiers

There is no toolbox provided, so we manually call "fitcnb" function in MATLAB to train a naive Bayes model. The accuracy on the testing set is attached to Table 1. The training speed is ultrafast, only seconds.

Comparison

We obtain the testing accuracy, as well as the training speed, in Table 1.

- **SVM:** The SVM-based classifier with a cubic kernel function achieves the highest accuracy on the testing set. The SVM with a fine Gaussian kernel has been overfitting the data, since $\sigma = 0.1$ is too small and it makes the exponential term decay too fast. However, it takes minutes to train the model, which is not so competitive as the FCNN in terms of training speed.
- **FCNN and LR:** Fully connected neural network with a hidden layer of 100 nodes also attains a high accuracy, with a training process faster than the SVMs above. It overwhelms the logistic regression model, which can be interpreted as a FCNN with no hidden layer. Here we see the crutial importance of the hidden layer. The additional layer provides a high capacity when fitting nonlinear function.
- **Naive Bayes:** The accuracy is not satisfactory, but it is super fast in training, much faster than other methods. I suppose that possible ways to improve this include feature selection and PCA. It also helps if we use advanced Bayes classifiers in place of the Naive Bayes classifier.

Source Code

Please download the souece code from http://39.106.23.58/files/PR5_2015011506.7z

For Question 2, please run "main.m". It may take minutes to train the network, but the result is reproducible because of the random seed.

For each model, I clicked "Generate code" button to transcript my operations into MATLAB codes, and stored each of them in the corresponding ".m" file. These files include:

- trainClassifierCubic.m
- trainClassifierLinear.m
- trainClassifierQuadratic.m
- trainClassifierGaussianFine.m
- trainClassifierGaussianMedium.m
- trainClassifierGaussianCoarse.m
- trainClassifierFullyConnected.m
- trainClassifierLogistic.m

Thus, the steps above can be easily reproduced without using the GUI of the toolbox.