Introduction to Artificial Intelligence: Assignment #2

Autumn 2015

Due: Oct 23rd, 23:59:59 CST (UTC +8).

1. A Walk Through Linear Models

In this problem, you will implement a whole bunch of linear classifiers and compare their performance and properties.

We assume our target function is $f : [-1, 1] \times [-1, 1] \to \{-1, +1\}$. In each run, choose a random line in the plane as your target function f, where one side of the line maps to +1 and the other maps to -1.

Skeleton code and MATLAB functions including *mkdata* and *plotdata* are given for your convenience, see the comments in the code for more details. What you need to do is implementing each algorithm and write scripts to play with them. Your algorithm implementations should be able to handle data in arbitrary dimension. For most of the questions below, you should repeat the experiments for 1000 times and take the average. See *run.m* for a script example.

Hint: Do not forget to add the bias constant term!

(a) **Perceptron**

Implement Perceptron learning algorithm (in *perceptron.m*), then answer the following questions.

(i) What is the training error rate and expected testing error rate (since we know the true target function, you can either calculate this exactly, or approximate it by generating a large set of testing points to estimate it), when the size of training set is 10 and 100 respectly?

(ii) What is the average number of iterations before your algorithm converges when the size of training set is 10 and 100 respectively ?

(iii) What will happen if the training data is not linearly separable (Use *mkdata(N, 'noisy')* to generate non-linearly separable data) ?

(b) **Linear Regression**

Implement Linear Regression (in *linear_regression.m*), then answer the following questions.

Note that we use Linear Regression here to classify data points. This technique is called Linear Regression on indicator response matrix[1].

(i) What is the training error rate and expected testing error rate if the size of training set is 100 ?

[1] http://books.google.com/books?id=VRzITwgNV2UC&pg=PA81

(ii) What is the training error rate and expected testing error rate if the training data is noisy and not linearly separable (nTrain = 100) ?

(iii) Run Linear Regression on dataset *poly_train.mat*. What is the training error rate? What is the testing error rate on *poly_test.mat* ?

(iv) Do the following data transformation

$$(1, x_1, x_2) \rightarrow (1, x_1, x_2, x_1x_2, x_1^2, x_2^2).$$

on dataset *poly_train.mat*. After this transformation, what is the training error rate? What is the testing error rate on *poly_test.mat* ?

(c) **Logistic Regression (Optional)**

Implement Logistic Regression (in *logistic.m*), then answer the following questions.

Remember that in the last homework assignment, we have showed that under certain conditions, the posterior probability estimation of Gaussian Discriminant Analysis will be in the form of

$$h_\theta(x) = g_\theta(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}.$$

We achieve this by using a generative model that assumes $p(\mathbf{x}|y) = N(\mu, \Sigma)$. Now instead of making assumption on $p(\mathbf{x}|y)$, we now use a discriminative model that directly assumes

$$p(y|\mathbf{x}) = \frac{1}{1 + e^{-\theta^T x}}.$$

Then given a training set, instead of estimating parameters μ or Σ, we find the best θ that fits the dataset. This model is called Logistic Regression.

By making significantly weaker assumptions, logistic regression is more robust and less sensitive to incorrect modeling assumptions. Therefore in practice, logistic regression is used more often than GDA.

Let us assume

$$P(y = 1|x;\theta) = h_\theta(x), \quad P(y = 0|x;\theta) = 1 - h_\theta(x),$$

then

$$P(y|x;\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}.$$

Now you can learn your logistic model using MLE. Not like Linear Regression, we have no closed-form solution to solve the MLE problem. So you should use gradient descent to maximize the log-likelihood function iteratively.

Derive the gradient descent update rule, choose a proper learning rate, and repeat the experiment for 100 times to take average.

(i) What is the training error rate and expected testing error rate if the size of training set is 100 ?

(ii) What is the training error rate and expected testing error rate if the training data is noisy and not linearly separable (nTrain = 100) ?

(d) **Support Vector Machine**

Implement Support Vector Machine without introducing slack variables (in *svm.m*), then answer the following questions.

Hint: Using MATALB built-in function *quadprog*, you should be able to implement SVM in less than 10 lines of code.

(i) What is the training error rate and expected testing error rate if the size of training set is 30 ? (Use *plotdata* to plot your learnt model, hope the graph will help you understand SVM as a maximum margin classifier.)

(ii) What is the training error rate and expected testing error rate if the size of training set is 100 ?

(iii) For the case nTrain = 100, what is average number of support vectors in your trained SVM models?

2. Regularization and Cross-Validation

We now learnt the undesirable consequence of overfitting, just as this quotation says:

> If you torture the data for long enough, it will confess to anything.

Regularization and Cross-Validation[2] are the two most common used methods to control overfitting. In this problem, we will use these two techniques together to relieve the overfitting problem.

Cross-Validation is a standard technique for adjusting regularization parameters of learning models. We first split the dataset into training set and validation set. Then we compute the validation error on validation set using different values for regularization parameters. Finally, we pick the parameter with the lowest validation error and use it for training our learning model. To reduce variability, we can run validation multiple rounds using different dataset partitions, and the validation results are averaged over all the rounds.

In the following questions, we will use leave-one-out cross-validation to choose regularization parameters. As the name suggests, leave-one-out cross-validation (LOOCV) involves using a single observation from the original sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data.

You may use *validation.m* as a skeleton.

(a) Implement Ridge Regression (in *ridge.m*), and use LOOCV to tune the regularization parameter λ. Train your algorithm on *digit_train.mat* and test on *digit_test.mat*.

Hint: Since the dataset is a set of raw images (hand-written digits) without preprocessing, you should do feature normalization[3] to make the feature have zero-mean and unit-variance. You can visualize the dataset using *show_digit.m*.

[2] `https://en.wikipedia.org/wiki/Cross-validation_(statistics)`

[3] `https://en.wikipedia.org/wiki/Feature_scaling`

(i) What is the λ chosen by LOOCV ?

(ii) What is $\sum_{i=1}^{m} \omega_i^2$ with and without regularization (let $\lambda = 0$) ?

(iii) What's training/testing error rate with and without regularization ?

(b) (Optional) Implement Logistic Regression with regularization (in *logistic_r.m*), and use LOOCV to tune the regularization parameter λ. The regularization term is similar to Ridge Regression.

Train your algorithm on *digit_train.mat* and test on *digit_test.mat*. Then report the training/testing error rate with and without regularization (let $\lambda = 0$). You should also report the λ chosen by LOOCV.

3. Bias Variance Trade-off

Let us review the bias-variance decomposition first.

The intuition behind it is straight-forward: if the model is too simple, the learnt function is biased and does not fit the data. If the model is too complex then it is very sensitive to small changes in the data.

If we were able to sample a dataset D infinite many times, we will learn different $g(x)$ for each time, and get an expected hypothesis $\bar{g}(x)$. So bias means the difference between the truth and what you expect to learn. It measures how well our model can approximate the truth at best.

However, it is impossible to sample the training dataset multiple time, so variance means the difference between what you learn from a particular dataset and what you expect to learn.

Now please answer the following questions:

(a) True of False:

(i) If a learning algorithm is suffering from high bias, adding more training examples will improve the test error significantly.

(ii) We always prefer models with high variance (over those with high bias) as they will able to better fit the training set.

(iii) A model with more parameters is more prone to overfitting and typically has higher variance.

(iv) Introducing regularization to the model always results in equal or better performance on the training set.

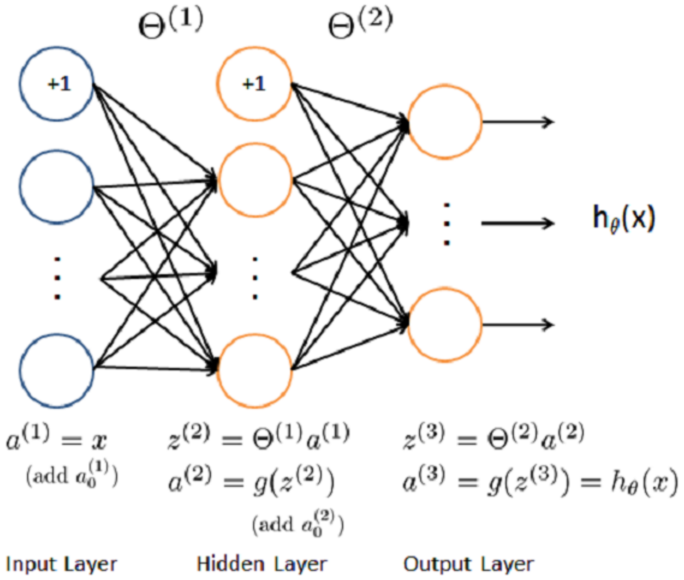
(v) Using a very large value of regularization parameter λ cannot hurt the performance of your hypothesis.

4. Neural Network vs. SVM

In this part of the homework, you will implement a Neural Network to recognize handwritten digits, and compare it with Support Vector Machine. See *nn_vs_svm.m* for a skeleton.

(a) Neural Network

Our neural network has 3 layers – an input layer, a hidden layer and an output layer. Since our inputs are pixel values of digit images of size 20x20, this gives us 400 input layer units (excluding the extra bias unit which always outputs +1). The output layer has 10 units, each represents a class of digits. Between them we have 25 hidden layer units (excluding extra bias unit), whose activation function is sigmoid function $g(x) = \frac{1}{1+e^{-x}}$.



The parameters for the neural network have already trained for you (in *weights.mat*), your goal is to implement the feedforward propagation algorithm (in *sigmod.m, feedforward.m*) to use our weights for prediction. You should use the one with largest output unit value as your predicted label.

After implement the feedforward algorithm, test it on *digit_data.mat*, then report the testing error rate.

(b) Multi-class SVM

In this **last** section, we will build a multi-class SVM classifier using one-vs-all strategy with LIBLINEAR library.

The classifiers we have seen so far are all binary classifiers. How can we solve multi-class classification problem? One typical method is one-vs-all strategy[4]. The idea is simple that a single classifier is trained per class to distinguish that class from all other classes – which means if the number of classes is C, then we need to train C classifiers (except when $C = 2$), then decide final predications based on the confidence of different classifiers.

[4] `https://en.wikipedia.org/wiki/Multiclass_classification`

The de facto standards of SVM library is LIBSVM[5] and LIBLINEAR[6](linear version LIBSVM). When solving real-world problems, you will unlikely use your self-implemented SVM. Instead, we usually turn to LIBSVM or LIBLINEAR for speed and reliability.

Your task here is building a multi-class SVM classifier using LIBLINEAR and one-vs-all strategy. Use the first half data examples in *digit_data.mat* as training data, the rest as testing data, then report the testing error rate.

Note that LIBLINEAR have built-in multi-class classification support, and you are encouraged to play with that feature. However you must implement your own one-vs-all multi-class classifier, since it is a general way to extend binary classifiers to support multi-class classification.

Please submit your homework report in **pdf** format, with all your code in a zip archive. Your files are suggested to be named as studentnum_name.pdf/zip/rar.

[5] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[6] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`