EE5177 : Machine Learning for Computer Vision

Programming Assignment - 4 : Logistic Regression and SVMs

Due date: April 11th 2017, 11:55pm IST

Note:

1. For any questions, please schedule a time with TAs according to their convenience. Please use moodle dicussion thread for posting your doubts. Reposting same questions is not encouraged.

   (a) Problem: 1&3, Sneha Konnur ee15s054@ee.iitm.ac.in

   (b) Problem: 2&3, Sapana Chaudhary ee15s300@ee.iitm.ac.in

2. Submit a single zip file in the moodle named as PA2_Rollno.zip containing the report and the folders containing corresponding codes.

3. **Read the problem completely before starting out to code.**

4. Late submissions will be evaluated for reduced marks with a penalty of 10% of the marks for each day after the deadline.

Problem-1: **Logistic Regression[20 Marks]**
You will be using a synthetic dataset. Generate 2 classes of univariate normal distribution with the following parameteres:

1 Class 0 : Mean=-2, standard deviation =1.5

2 Class 1 : Mean =2, standard deviation =1.5

Generate 100 Samples for each class. Use the first 80 for training and the rest 20 for testing. You need to perform logistic regression on this dataset. Estimate the parameters for the model using Maximum Likelihood Estimation. Compute the Hessian and the gradient for the log-likelihood function. Use $fminunc$ in MATLAb to minimize the negative log-likelihood function.
Plot the input dataset, the sigmoid associated with the parameters learnt, the decision boundary. Also obtain the confusion matrix.

Linearly separable data is an issue for logistic regression. Why? How do you avoid the problem?

Problem-2: **Support Vector Machines[30 Marks]**
The aim of this problem is to get you familiarized with the basics of Support Vector Machines. We would be using CVX(a Matlab Solver for Disciplined Convex Programming) to solve the convex optimization problem arising in the dual formulation of SVM. You need to write code for structuring the data so that the library routine can find the maximal-margin solution. Also, you would code the transformation of this solution into a classifier for carrying out prediction. This assignment would be done in Matlab.

CVX Installation Instructions

- Download CVX(Standard bundles, including Gurobi and/or MOSEK) for your OS from `http://cvxr.com/cvx/download/`
- CVX will not work on Matlab versions later than 2017a
- Open Matlab

- In the command line run cd /path/to/folder/conating/cvx (Do not manually change the path to cvx directory)
- Now run cvx_version
- Fill out the academic license request for the username and first hostid listed on running the command above
- Download the license file sent to you
- Run cvx_setup /path/to/cvx_license.dat
- If you encounter any error while performing the said tasks, post it on moodle. We would get back asap. Do not mail the TAs directly.
- A simple CVX program looks something like this:

```
% Bounded least squares problem
% input data
m = 16; n = 8;
A = randn(m,n);
b = randn(m,1);
bnds = randn(n,2);
l = min( bnds, [] ,2 );
u = max( bnds, [], 2 );

% cvx version
cvx_begin
    variable x(n)
    minimize( norm(A*x-b) )
    subject to
        l <= x <= u
cvx_end
```

Implementation

Like mentioned earlier, we would be using CVX. We would solve the quadratic optimization

problem by writing in the form as in code snipet included earlier. In Matlab, cd to the folder containing CVX.
We want to find an $\vec{\alpha}$ which minimizes

$$\frac{1}{2}\vec{\alpha}^T P\vec{\alpha} + \vec{q}^T\vec{\alpha} \qquad \text{while } G\vec{\alpha} \leq \vec{h} \tag{1}$$

Here, P and G are matrices, while $\vec{q}$ and $\vec{h}$ are vectors. We need to build these necessary vectors and matrices, run the CVX code to get optimal α's.

In order to visualize the decision boundaries graphically we will restrict ourselves to two-dimensional data, i.e. points in the plane. The data is in the form of a vector of datapoints, consisting of three elements. The first two numbers are the x and y coordinates and the last number is the class (-1 or 1).
The data for this problem is contained in DS1.

Things to implement You will have to write code for:

- Code the gaussian kernel function
  The kernel function takes two data points as arguments and returns a "scalar product-like" similarity measure; a scalar value. Do not forget that you need to add 1 to the output, representing the extra "always-one" component.

- Build the P matrix
  The P matrix should have the following elements:

  $$P_{i,j} = t_i t_j \mathcal{K}(\vec{x}_i, \vec{x}_j)$$

  Indices i and j run over all the data points. Thus, if you have N data points, P should be an $N \times N$ matrix.

- Allocate space for the $\vec{q}$ vector, G matrix, and $\vec{h}$ vector
  These vectors and matrices do not hold any actual data. Still, they need to be set up properly so that CVX solves the right problem.
  TIPS:
  $\vec{q}$ should be a N long vector containing only the number -1.
  $\vec{h}$ must be a vector with all zeros.

The greater-than relation in eq.7 from tutorial on SVMs has to be made to match the less-than relation in (1). This can be achieved by creating a G matrix which has -1 in the diagonal and zero everywhere else (check this!).

- Run `CVX` for this kernel. Check for:
  - Number of iterations to converge to the optimal solution
  - Norms of the matrices and vectors
  - Total CPU time
  - Optimal values for α

- Pick out the non-zero α values If everything else is correct, only a few of the α values will be non-zero. Since we are dealing with floating point values, however, those that are supposed to be zero will in reality only be approximately zero. Therefore, use a low threshold (10^{-5} should work fine) to determine which are to be regarded as non-zero.

  You need to save the non-zero α_i's along with the corresponding data points ($\vec{x}_i$) in a separate matrix.

- Implement the indicator function

  Implement the indicator function (equation 6 from tutorial) which uses the non-zero α_i's together with their $\vec{x}_i$'s to classify new points.

Plotting the Decision Boundary

By drawing a contour at the level where the classifier has its threshold we will get the decision boundary. What we will have to do is to call your indicator function at a large number of points to see what the classification is at those points. We then draw a contour line at level zero, but also countour lines at -1 and 1 to visualize the margin.
If everything is correct, the margins should touch some of the datapoints. These are the *support vectors*, and should correspond to datapoints with non-zero α's.

Running and Reporting

Once you have the gaussian kernel running, there are a number of things you can explore.

Remember that support vector machines are especially good at finding a reasonable decision boundary from small sets of training data.

1. When do you think you cannot find solution to the problem ? Does it have to do anything with linear separability of the classes.
2. Explore how kernel parameters influence the decision boundary. Reason in terms of bias-variance tradeoff. A very good resource on bias-variance trade-off: `http://scott.fortmann-roe.com/docs/BiasVariance.html`

OPTIONAL: FOR EXTRA CREDIT

Slack Implementation

You should now alter your program to include slack variables. We would use another data set(DS2) for this. Slack variables can quite easily be incorporated by adding the extra contraints (see section 4 of the tutorial). This means that you have to extend the matrix G and vector $\vec{h}$ with additional rows corresponding to the $\alpha_i \leq C$ constraints:

- G should now be a $2N \times N$ matrix. Note that the constraint is now reversed, resulting in the additional lower part being negated as compared to previously.
- $\vec{h}$ should now be a $2N$ column vector.

Repeat the previous exercise with slack variables added. Answer the additional following questions:

1. Explore the role of the parameter C. What happens for very large/small values?
2. Imagine that you are given data that is not easily separable. When should you opt for more slack rather than going for a more complex model and vice versa?

Problem-3: **Comparison[10+20 Marks]**

Now we would implement logistic regression and SVMs on real world data. This is more of a hyper-parameter tuning and cross-validation task. The data is provided to you as mat files. It consists of features of images from the following four categories: coast, forest, inside-city, mountain. We prefer that you do it in Matlab.

Description of the data

- Train data: Input matrix : XTrain
  Labels: YTrain

- Test data:
  Input matrix : XTest
  Labels: YTest

Logistic Regression:

We will be using the GLMNET package as described below for this task.In the the GLMNET framework, the training of the logistic regression model consists of estimating the unknown parameters by maximizing the penalized log-likelihood function

$$P(k|x) = \frac{exp(a_k^T x + b_k)}{\sum_{k=1}^{C} exp(a_k^T x + b_k)} \tag{2}$$

$$\max_{a_k, b_k} [\sum_{i=1}^{N} log[P(y_i = k|x_i)] - \lambda ||a_k||_1] \tag{3}$$

The goal is to find 4 parameter vectors (a_k, b_k) k=1,2,3,4 that together maximize the log-likelihood of classes given the data. The latter term is a regularizer, whose role is to prevent over-fitting the model to the training data by giving additional penalty for large coefficient values. The penalization has been shown to improve accuracy for test data, and in particular the L1-norm penalty has been of great interest recently. The amount of penalty is determined by the coefficient λ. If λ is large, the weights are forced to small (close to zero) values, while a small λ allows more freedom to choose larger coefficients.

Steps to work with the GLMNET package

0. Download the package from `http://web.stanford.edu/~hastie/glmnet_matlab/`. Place it in your Matlab working directory.

1. The training is done using the glmnet function as follows:

   ```
   model = glmnet(X_train, y_train, 'multinomial');
   ```

   As a result, the trained classifier will be stored in variable model.The variable contains the (among others) the following fields:

   (a) `model.lambda`: A vector containing all used values for the penalty parameter λ.

   (b) `model.beta`: A cell of 4 matrices. Each column represents a set of coefficients a_k (see equation (2))for each value of parameter λ.

   (c) `model.a0`: a vector containing the constants b_k for the logistic regression model of equation (2); one for each value of parameter λ.

2. The prediction can be done using the glmnetPredict function:

   ```
   yHat = glmnetPredict(model, X_test, lambda, 'class');
   ```

We will decide the value of regularisation parameter λ using K-fold Cross-Validation.

Cross-Validation

K-fold cross-validation does the following:

- Split the data randomly into K parts. These parts are called folds.
- For each fold k = 1; 2; : : : ;K:
  (a) Train with all samples except those on fold k.
  (b) Predict the class labels for all samples on fold k.
  (c) Calculate the accuracy of predictions.
- Average the accuracies of the k folds.

Classify the test data and report the accuracy.

SVMs: We would make use of **libsvm** from `https://www.csie.ntu.edu.tw/~cjlin/libsvm/` to perform this task. Classification model
Classification models would be built for the following kernels. You are free to try other kernels if you wish to.

- Linear kernel
- Gaussian kernel

Instructions for using libsvm

- Download the latest version of the software for your OS
- Open the folder named Matlab, run make.m from Matlab command line
- Once mex files are successfully generated, place them in the folder you would write your remaining code in.

Requirements from you

- Come up with the kernel parameters for the various models. You will have to do grid-search for this(`https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search`)
- Use n-fold cross validation to find best model parameters. A fraction of training data can be used to perform cross validation.
- Run your models on the test data and report accuracy.

–end–