Machine Learning in MATLAB

Roland Memisevic

January 25, 2007

Last time

- Working with MATLAB.
- Vectors, matrices, operations.
- Control structures.
- Scripts and functions.
- Plotting.
- Slicing/Logical indexing.

A few random things

- Get the size of an object with 'size'. Takes an optional argument to specify the dimension (without, it returns an array with the sizes of all dimensions). Example:

```
A = [1 2 3
     4 5 6]
size(A,1) = 2
size(A,2) = 3
size(A) = [2 3]
```

- Use '...' to let commands span several lines.
- Clear memory by deleting some unused variables using 'clear variable-name'
- 'who' and 'whos' show the variables in the workspace.
- Use 'drawnow' to draw now (flush the event queue) ...
- Use 'save' to save the current workspace. 'load' to get it back.

repmat

- Use 'repmat' to build new matrices from smaller ones. Example:

```
x = [1;2]
repmat(x,1,3) = [1 1 1
                 2 2 2]
repmat(x,2,2) = [1 1
                 2 2
                 1 1
                 2 2]
```

- Used mostly to 'vectorize' things (more in a moment...)

Representing data

- We are often dealing with lots of data, and normally that data is given in the form of real-vectors. If not, we can often massage it accordingly... (performing *feature extraction*).
- It is convenient to use *matrices* to store data, for example by stacking the vectors column-wise:

$$X = (\mathbf{x_1}\mathbf{x_2}\ldots\mathbf{x_N})$$

- To display up to 3-d data given in this form, you could use

```
scatter(X(1,:),X(2,:)) % display 2-d dataset
scatter3(X(1,:),X(2,:),X(3,:)) % display 3-d dataset
```

High dimensions

- We can index arrays linearly (regardless of the number of dimensions). Example: If A=[1 2;3 4], then A(3) = 3.
- Summarizing operations, such as 'sum', 'mean', etc. can be applied to arrays of any dimension (not just vectors). Use an additional argument to specify over which dimension to perform the operation. Example: If A = [1 2 ;3 4], then

```
sum(A,1) = [4 6]
sum(A,2) = [3
            7]
```

- Many operations on vectors and matrices extend to objects with more than 1 or 2 dimensions. Example: Construct a random $5 \times 5 \times 5$-array with 'randn(5,5,5)'.

Linear models

- Many machine learning methods are based on simple 'neurons' computing:
- $$\mathbf{w}^{\mathrm{T}}\mathbf{x} \quad \left(\quad = \sum_{k} w_k x_k \quad \right)$$
- Perceptrons, back-prop networks, support vector machines, logistic regression, PCA, (nearest neighbors), ...

'Vectorizing'

- Applying the linear function to data-points stacked column-wise in a matrix X is simply $Y = \mathbf{w}^{\mathrm{T}} X$.
  In MATLAB:

```
Y = w'*X;
```

- In MATLAB writing stuff in matrix form can be faster than using loops. Referred to as 'vectorization'.
- Another example. Suppose you want to *mean center* a set of vectors stored in X. Instead of

```
m = mean(X,2);
for i = 1 : size(X,2)
  X(:,ii) = X(:,ii) - m;
end
```

  we could write:

```
X = X - repmat(mean(X,2),1,size(X,2));
```

Linear regression

- Problem: We are given input/output data (real values). Want to learn the underlying function.
- Assumption for now $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ and $y = f(\mathbf{x}) + \text{noise}$
- We can use MATLAB to get a quick idea of what's going on:

```
X = randn(2,1000);          % make some input data
w = [1;1];                  % define a linear function
n = randn(1,1000)*0.1;      % produce some noise
Y = w'*X + n;               % produce outputs
scatter3(X(1,:),X(2,:),Y);
```

Learning

- Unfortunately, in reality we don't know $\mathbf{w}$...
- Given only a set of examples y_i and $\mathbf{x_i}$, what would be a **reasonable guess** for $\mathbf{w}$?
- Standard approach: Minimize the sum of squared errors

$$E(\mathbf{w}) = \sum_i (\mathbf{w}^{\mathrm{T}} \mathbf{x_i} - y_i)^2$$

- There are many ways of finding the $\mathbf{w}$ that minimizes $E(\mathbf{w})$.
- One very good way is this:
  - Start with a random guess $\mathbf{w}^{\mathrm{init}}$.
  - Follow the negative gradient $-\frac{\partial E}{\partial \mathbf{w}}$, until the error stops changing a lot.

Learning

- The gradient is $\frac{\partial E}{\partial \mathbf{w}} = 2\sum_i (\mathbf{w}^{\mathrm{T}}\mathbf{x_i} - y_i)\mathbf{x_i}$.
- So we just have to iterate:

$$\mathbf{w} \leftarrow \mathbf{w} - 2\epsilon \sum_i (\mathbf{w}^{\mathrm{T}}\mathbf{x_i} - y_i)\mathbf{x_i},$$

where ϵ is a small learning rate, without which we will overshoot the minimum.

- With vectorization, learning takes about 5 lines in MATLAB:

```
for iteration = 1 : 5000 %in practice: until stopping
                         %criterion satisfied
  grad = 2*sum(repmat(w'*X-Y,size(X,1),1).*X,2);
  w = w - epsilon * grad;
  err = sum((Y - w'*X).^2)        %just to check
end
```

Visualizing the solution

- To see what the learned model does on the training data:

```
hold on;      %keep old plot
Y_hat = w_learned'*X;
scatter3(X(1,:),X(2,:),Y_hat, 'r');
```

Linear classification

- Classification: We are given some data with associated class-labels, and want to learn the underlying function, that maps inputs to labels.
- Examples: Spam filtering, face recognition, intrusion detection, ... and many many many more.
- For now we consider only binary problems (2 classes). Most ideas can be quite easily extended to more than 2.
- Again, we will use a linear model. In particular:

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

- This means that we try to separate classes using a hyper-plane.

Linear classification

- Again we can use MATLAB to visualize what's going on:

```
X = [randn(3,200)-ones(3,200)*1.8 ...
  randn(3,200)+ones(3,200)*1.8];  %produce some inputs
Y = [zeros(1,200), ones(1,200)];  %produce some labels
scatter3(X(1,:),X(2,:),X(3,:), 80, Y, 'filled');
```

- How well does some random **w** do on the training set?

```
w = randn(3,1);
Y_random = sign(w'*X);
scatter3(X(1,:),X(2,:),X(3,:),80,Y_random, 'filled');
hold on;
plot3([0 w(1)], [0 w(2)], [0 w(3)], 'k'); %show w
hold off;
sum(Y_random~=Y)/200  %error rate
```

Learning

- How can we *learn* a good hyper-plane?
- There are many possible ways.
- An extremely powerful one is **perceptron learning:**
  - Start with some initial guess for $\mathbf{w}$. Then iterate, picking training examples (in any order):
    - if $sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x_i}) = y_i$, then do nothing
    - otherwise, set $\mathbf{w} := \mathbf{w} \pm \mathbf{x_i}$, depending on what the output was
- In MATLAB:

```
for iteration = 1 : 100
  for ii = 1 : size(X,2)
    if sign(w'*X(:,ii)) ~= Y(ii)
      w = w + X(:,ii) * Y(ii);
    end
  end
end
```

Conclusions

- A lot of machine learning is based on the simple 'neuron': $\mathbf{w}^{\mathrm{T}}\mathbf{x}$
- We have looked at basic regression and classification.
- Usually a few lines in MATLAB.
- A couple of things were oversimplified here. For example, in practice we would adapt the learning rate in gradient descent, add an extra input-dimension for the bias, etc.
- Can be easily applied to real data: E.g. Spam, ...