COS 424: Interacting with Data

Lecturer: Léon Bottou
Scribe: Yibiao Chai

Lecture of April 20, 2010

---

1 A Brief Review of Classifiers

We will examine in this section two broad classes of methods for solving classification problems in machine learning: generative and discriminative learning. Both of them are aimed to learn, or approximate, $f : \mathcal{X} \to \mathcal{Y}$, or $p(y|x)$ where $x \in \mathcal{X}$ (pattern set) and $y \in \mathcal{Y}$ (class set).

Generative Classifiers
In this class of classifiers, we model $p(x, y)$. We estimate $p(x|y)$ and $p(y)$ before using Bayes rule to back out $p(y|x)$. Parameters are typically estimated using a likelihood-based criterion on training data. To put it mathematically (See Scribe 4 for more details), we have

$$p(x, y) = p(y) \cdot p(x|y)$$

and

$$p(y|x) = \frac{p(x, y)}{\sum_i p(x, y_i)}$$

The optimal class is given by

$$\hat{y}(x) = \arg\max_x p(y|x)$$

Discriminative Classifiers
In this class of classifiers, we directly model $p(y|x)$, the mapping from patterns to output classes (either as a conditional distribution or simply as a prediction function), or a parametric model, is often assumed. Mathematically we get

$$p(x, y) = p(x) \cdot p(y|x)$$

The optimal class is given by

$$\hat{y}(x) = \arg\max_x p(y|x)$$

Optimization of the expectation of a loss function
We talked about optimization of loss functions in previous lectures. Indeed, we often derive an optimal class by optimize the expectation of loss functions (or surrogate loss functions). Typical functions of these are sigmoid, hinge, quadratic, log loss functions etc. For a lot of these functions, if we could really optimize them, asymptotically we can get discriminant functions that are sufficient for our purposes. Moreover, most of the time, we don't even need to precisely estimate $p(y|x)$, we just need to know one certain $p(y|x_i)$ is greater than all others to get the optimal class.

Another way of looking at the learning problem is as follows. Let's first look at the generative setup. Given the parametric likelihood $p_\theta(x, y)$, the parameters are obtained by solving

$$\max \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i, y_i)$$

The function to be maximized can be rewritten as

$$\underbrace{\frac{1}{n}\sum_i \log p_\theta(y_i|x_i)}_{\text{“discriminative” term}} + \frac{1}{n}\sum_i \log p_\theta(x_i) \tag{1}$$

after defining

$$p_\theta(x) = \sum_i p_\theta(x, y_i)$$

$$p_\theta(y|x) = \frac{p_\theta(x, y)}{p_\theta(x)}$$

In sum, we take a generative model for a parametric system, split the log-likelihood into two parts: the likelihood for a discriminative model plus another term. If we look at this closely, it looks very much like the SVM models where we solve the following problem

$$\text{optimize } \frac{1}{n}\sum_i \left(\mathcal{L}(x_i, y_i, \theta) + \lambda \cdot \Omega(\theta)\right) \tag{2}$$

where $\mathcal{L}$ is some loss term and Ω some regularization function of parameters.

A few comments are in order:

- In order to solve (2) we split the parameter solutions into sets of levels in the Θ-space (see Figure 1) and since the function to be optimized is convex in the case with SVM, gradient desccent method would suffice and is often efficient too.
- It is also easy to see that the more restrictive $\Omega(\theta)$ is, the smaller the dimension of parameters the model prefers. With λ fixed, as the number of samples increases, the training error reaches a stable level while the testing error eventually increases. It is therefore not surprising that we get the kind of curves as illustrated in Figure 2.
- The idea is that the more training data we have, the more the restrictive term comes into play, and as a result, the harder for us to find the optimal set of parameters.

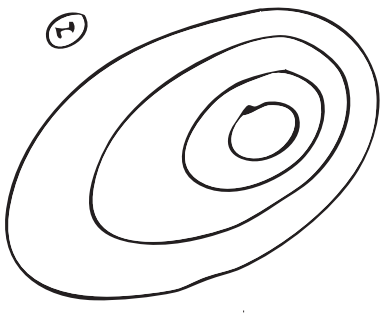

Figure 1: Parameter solutions in Θ-space and gradient descent

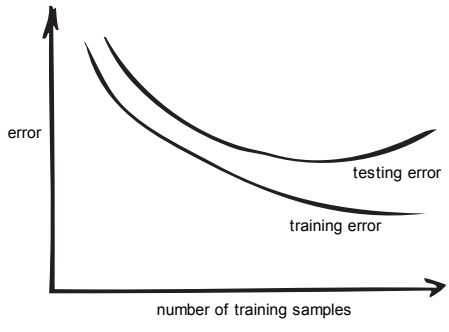

Figure 2: SVM testing and training error as a function of the number of training samples

Now back to our parametric model expressed in (1). We find similarities between (1) and (2) in that the "discriminative" term in (1) looks like the loss term in (2) and that the regularization term in (2) resembles the sum of log $p_\theta(x_i)$'s in (1). Therefore, we can view a generative model as the sum of a discriminative model and an additional term. This observation helps us reach some of the following remarks regarding the performance difference in the two classes of classifiers (for the others, please refer to Ng & Jordan 2002):

- The asymptotic (as the number of training examples approach infinity) classification accuracy for discriminative model is often better than the asymptotic accuracy of generative model.
- Generative model parameter estimates converge toward their asymptotic values faster than their discriminative counterparts.
- Generative model performs better (in terms of error) when training dataset is small while discriminative model performs better when training dataset is large.

Figure 3 illustrates the above comments.

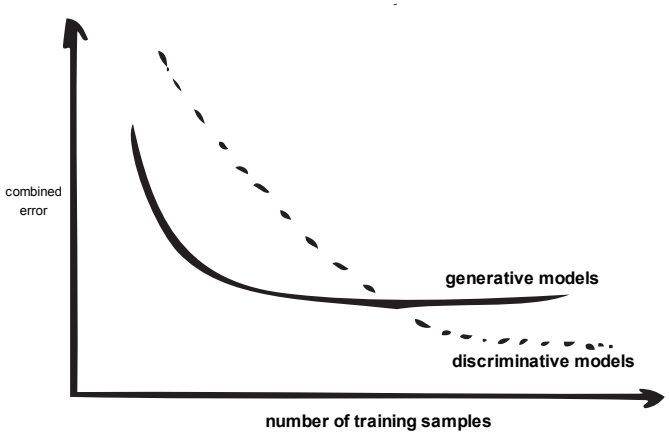

Figure 3: Performance: generative models vs. discriminative models. This figure is inline with the remarks regarding the performance of the two types of classifiers.

2 Graph Transformer Networks

Let's see some models at work.

Slide 2: (Example: Check reader) On a check, some pieces of information are well defined: the bank, the check maker, the payee, the date, the amount, the signature, etc. However, the relative positions of these are never fixed; no standard layout exists. In the mid-1990s, people started to scan checks and store the images for future references. This turned out to be expensive, bulky and inefficient at the time. People wanted to process these

images to extract textual/numeric information. Difficulties abound: handwritten numbers, cursively written names, lack of natural stops between digits, etc.

Slide 3: A traditional approach is given on this slide. From the field locator to linguistic/contextual postprocessor, all the subroutines are built by hand and manually adjusted.

Slide 4: What we really want, however, is to train all the parameters in the system to optimize the global performance.

Slide 7: One idea is to pile the different parts of the system together to form a multiple layer network. Different parts of the system are represented by boxes on the slide. Each box has a relatively small and simple system inside. The state variables here are fixed size vectors, which cannot represent sequential information as with the case of speech recognition, structured images, etc.

Slide 8-9: One way to remedy this is to turn to graph transformer network. The state variables are weighted graphs with numerical information attached to the arcs. A path in the graph represents a sequence of decisions. Depending on these decisions, the graph will be transformed accordingly to form a new graph.

Slide 10: (Example: Word reader) This slide illustrates a concrete example. Notice that the arcs in the graph after the segmentor represent the possible segments. After the character scorer, the arcs are used to represent possible character candidates (with the penalty score in the brackets). Viterbi algorithm is used to select the best segmentation in this example.

Slide 11: Let's go back to the two classes of classifiers a bit. We now know that they all maximize the same thing but the difference lies in the way in which the system is normalized. To some extent, switching from a generative model to a discriminative one is to change the way the normalization in the system.

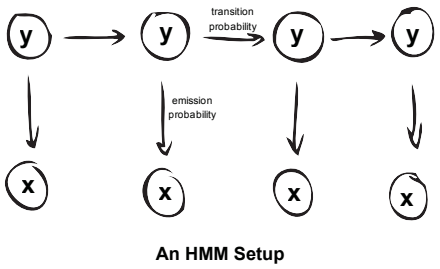

Figure 4: An HMM setup

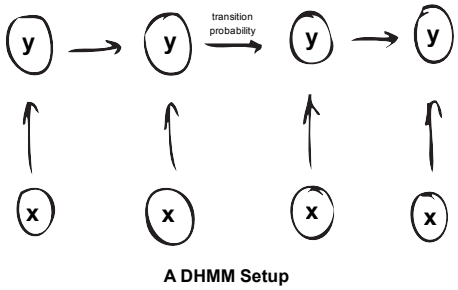

Figure 5: A DHMM setup

Slide 12: Let's look at hidden markov model used in both classes of classifiers. Figures 4 and 5 illustrates both HMM setup (generative model) and DHMM (discriminative

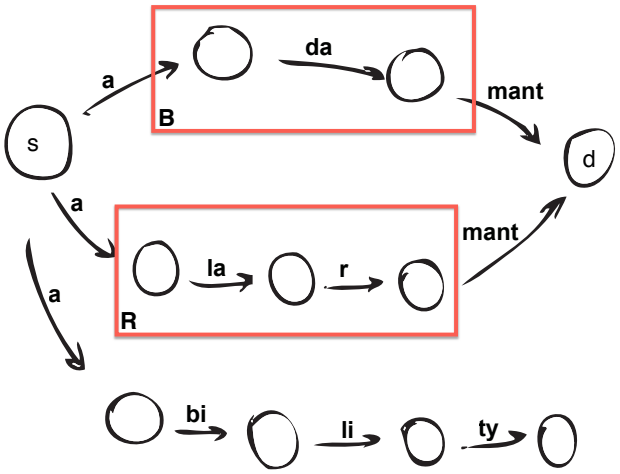

Figure 6: Recognize letters: a possible setup. Self loops are ignored for clarity.

model). The probabilistic construction in a HMM ensures normalization. In a DHMM, however, output of the local classifier would need to be normalized. But it turned out to be a bad idea.

To illustrate why it is so, suppose we would like to recognize letters. The setup is given in Figure 6. For this simple and purely discriminative HMM model, we have

$$\begin{aligned} &P(S_{t+} = B, S_t = B | S_{t-} = s) \\ &= P(S_t = B | S_{t-} = s) \cdot (1 - P(S_{t+} = d | S_t = B, S_{t-} = s)) \end{aligned} \tag{3}$$

where $\{S_t\}$ represents the state of the lexicon at time t, t_- is the step before t and t_+ the step after t. Notice that $P(S_t = B|S_{t-})$ is also the probability of getting "a" after startup (in the top path) in this simple setup while $P(S_{t+} = d|S_t = B, S_{t-} = s)$ is also the probability of getting "mant" after "ada". Therefore, Eq.(3) can be rewritten as follows

$$\begin{aligned} &P(S_{t+} = B, S_t = B | S_{t-} = s) \\ &= P(\text{getting "a" after startup}) \cdot (1 - P(\text{getting "mant" after current step})) \end{aligned}$$

However, we get exactly the same expression for $P(S_{t+} = R, S_t = R|S_{t-} = s)$ and since the system is normalized in a way that P(getting "a" after startup) in every path is the same and that P(getting "mant" after current step) is also the same, it makes it impossible to distinguish Box B from Box R. This makes the system lose certain information that we might be interested in. In sum, the big take-away is then: normalization operation at very level destroys useful information.

Slide 13: Now let's look at denormalized models. We use penalties instead of probabilities. The score $e^{-\text{penalty}}$ works just like probabilities when it comes to additions and multiplications.

Slide 14: This slide gives an example of a denormalized model at work. The penalty is still given in brackets while the number in parentheses represents the gradient associated. We run the system from bottom up and then move top-down to incorporate gradient information to (re)train the system.

Slide 15: (Example: Check reader continued) Two types of checks exist: personal checks and business checks. While the latter is usually machine printed, the former is often handwritten. Both of them present difficult problems when we try to read them by machine.

Slide 16: A possible recognition architecture is given on the slide. This time we use the graph transformer approach. The system starts with a simple graph with only two nodes and the arc representing the a suitable image of check and ends with the Viterbi algorithm looking for the amount written (the best interpretation) on the check.

Slide 17: The gradient-based discriminant training architecture is shown here. Again, we use gradient back-propagation to train the system.

Slide 18: In sum, graph transformer networks work in general. For example, the check reader is now widely in use. Some performance results in the literature are listed on this slide.

Slide 19-23: Now let's look at the these graph transformer systems. They transform one graph into another sequentially according to some predefined rules. As an example, a lexicon would need to build a subgraph consistent with the grammar.

There are two types of transformers, composition transformers and pruning graph transformers. In the former, we repeat two important operations: match and build (see the example of the lexicon on Slide 20). The codes are easy to write and are reusable. Pruning graph transformers, on the other hand, remove selected arcs from a graph. The Viterbi algorithm is a good example.

The back-propagation training method we introduced earlier can be readily applied to both systems.

Slide 26: (Example: Character spotting) Sometimes we don't need to segment, we can use a single object recognizer and slide it through the input. But this is of course very costly.

Slide 28-30: (Example: SDNN handwriting recognizer) Please refer to slides. Several additional comments: This result of convolutional neural networks are invariant to shifts, noise and extraneous marks on the input and it is efficient as compared to the character spotting example above.

In conclusion, the graph transformer networks are a success overall, both in theorectical understanding and in practical use. The only bemol might be that it is less of a success in terms of dissemination of knowledge due to the fact the writing papers on the very subject, concerning often large and complex systems demanding years of development, is simply not an easy task.