Conditional random fields, hidden Markov models, and surroundings

Massimo Piccardi

UTS

May 2021

The problem

We are given a sequence of tokens, $x = \{x_1, x_2 \dots x_t \dots x_T\}$ (or $x_{1:T}$, concisely) in input, and we want to predict a corresponding sequence of labels, $y = \{y_1, y_2 \dots y_t \dots y_T\}$ (or $y_{1:T}$) as output.

This notation is a bit loose, because with y_t we can mean either a variable or a specific label. It should always be clear from the context.

This problem reflects many NLP problems such as POS tagging, NER etc.

A basic solution

As the simplest solution, we can design a *scoring function* (an SVM, a CNN etc) that assigns a score to a label, y_t, and the corresponding token, x_t:

$$F(y_t, x_t) \tag{1}$$

The assumption is that the higher the score, the better (the match between y_t and x_t). The scoring function may take any values in $\mathbb{R}$.

Prediction

To choose our predicted label, we can try all its possible values and choose that that maximises the scoring function:

$$\bar{y}_t = \text{argmax}_{y_t} F(y_t, x_t) \tag{2}$$

For instance, x_t could be 'dog', y_t could take values in set $\{NOUN, VERB, ADJECTIVE\}$, and $\bar{y}_t$ could be NOUN.

We repeat this separately for every $t = 1 \ldots T$ and we accomplish our task.

Probabilistic classifiers

If the scoring function is restricted to be a conditional probability distribution, this becomes:

$$\bar{y}_t = \text{argmax}_{y_t} p(y_t|x_t) \tag{3}$$

The only substantial difference is that the score now is bound between 0 and 1, and the sum of the scores for all possible labels for y_t is always 1.

Discriminative models

Note that in $p(y_t|x_t)$, x_t is a fixed value, while y_t is meant as a variable.

$p(y_t|x_t)$ assigns a value of probability to every possible value (label) for y_t. Instead, it doesn't assign any value of probability to x_t. x_t is not treated as a random variable, just as a given, control input.

These models are called *discriminative models* and are the most common nowadays. The most notable examples are the logistic regression classifier, conditional random fields (CRFs), and most deep neural networks.

Generative models

Note that **joint** probability $p(y_t, x_t)$ is different from **conditional** probability $p(y_t|x_t)$.

In the first place, $p(y_t, x_t)$ can be deemed "bigger" than $p(y_t|x_t)$ because, from Bayes' theorem:

$$p(y_t, x_t) = p(y_t|x_t)p(x_t) \tag{4}$$

So, $p(y_t, x_t)$ adds a model for $p(x_t)$, the probability of x_t itself.

Generative models

In practice, $p(y_t, x_t)$ allows assigning a value of probability not only to the values of y_t, but also to those of x_t. x_t is now treated as a random variable.

If we think of x_t as a variable, $p(y_t, x_t)$ can be used to *sample* joint values of (y_t, x_t), thus obtaining "virtual" tokens. This is why these models are called *generative*.

Notable examples include Bayesian classifiers, hidden Markov models, Bayesian networks, and deep generative models (GANs, VAEs etc).

Decisions

Please note that if one has a discriminative model, $p(y_t|x_t)$, and a corresponding generative model, $p(y_t, x_t) = p(y_t|x_t)p(x_t)$, they would lead to the same decisions/predictions for any given x_t:

$$\text{argmax}_{y_t} p(y_t|x_t) = \text{argmax}_{y_t} p(y_t, x_t) \tag{5}$$

because $p(x_t)$ does not depend on y_t.

The debate

The typical argument in favour of using discriminative models is that we do not need $p(x_t)$ to make a decision about y_t. x_t is just an input, we do not need to know how frequently it occurs to make an optimal prediction for y_t. And, in general, the simpler a model, the better.

The debate

The arguments in favour of using generative models are:

- we can use Bayes' theorem in the other direction and break down $p(y_t, x_t)$ as $p(x_t|y_t)p(y_t)$. This allows us to build the generative model from the product of two separate modules, that can have their own distributions and parameters. These are called Bayesian classifiers (Naïve Bayes is just a special case);
- being able to sample the model can be useful in various cases (e.g. GANs, VAEs etc).

Nature of the distributions

y_t is a categorical variable, and its distribution is always the usual multinomial. x_t can be either a categorical variable or a token embedding. In the latter case, if x_t is modelled as a random variable, its distribution can be any continuous distribution (e.g., a multivariate Gaussian).

The parameters of these distributions can be simply stored in vectors, or, in case they are used within a larger model, they can be inferred by neural networks and other regressors, with their own parameters at their turn.

The logistic regression classifier

The logistic regression classifier is the "workhorse" of all discriminative models. It models $p(y_t|x_t)$ as the ratio of two non-negative terms:

$$p(y_t|x_t) = \frac{f(y_t, x_t)}{g(x_t)} \tag{6}$$

The numerator is a non-negative term that depends on y_t and x_t, and the denominator is a non-negative term that depends only on x_t. The ratio has to be a properly normalised probability of y_t, given x_t (i.e. has to add up to 1 over all possible values of y_t).

The logistic regression classifier

The logistic regression classifier has this form:

$$p(y_t|x_t) = \frac{\exp^{w_{y_t}^\top x_t}}{\sum_{y_t} \exp^{w_{y_t}^\top x_t}} \tag{7}$$

The numerator is always non-negative because it's the value of an exponential function. The exponent, $w_{y_t}^\top x_t$, is the dot product between the input, x_t, (encoded in some form; for instance, one-hot, or a token embedding) and a vector of parameters of the same size, w_{y_t}, specific for the label class, y_t.

Since $w_{y_t}^\top x_t$ is a linear function of x_t, this model is often referred to as a "log-linear" model (the log of $\exp^{w_{y_t}^\top x_t}$ is linear).

cont'd

The logistic regression classifier

The denominator just adds up the numerator for all possible values of y_t. In this way, it eliminates the dependence on y_t, and ensures that the ratio add up to 1 over all values of y_t, as required by a normalised probability:

$$\sum_{y_t} p(y_t|x_t) = \sum_{y_t} \left[\frac{\exp^{w_{y_t}^\top x_t}}{\sum_{y_t} \exp^{w_{y_t}^\top x_t}} \right] = \frac{\sum_{y_t} \exp^{w_{y_t}^\top x_t}}{\sum_{y_t} \exp^{w_{y_t}^\top x_t}} = 1 \quad (8)$$

The logistic regression classifier

The logistic regression classifier models the probability of classes, so it's a classifier, not a regressor.

Its form "maximises the entropy" (a positive property), so in the NLP community used to be known as the "MaxEnt" classifier.

The exponents, $w_{y_t}^\top x_t$, are known as the *logits*, and they are unrestricted (they can be positive or negative or 0). This makes training of w_{y_t} easier.

A common use of the logistic regression classifier is as the last (i.e., output) layer of a neural network classifier. In that case it's known as *softmax*, and the logits are produced by some neural network, $F(y_t, x_t)$.

A parenthesis on training

This presentation focuses on prediction (often loosely called 'inference'), and we omit the parameters in the notation of the distributions. For clarity, notations such as $p(y_t|x_t)$ and $p(y_t|x_t, \theta)$ refer to the same distribution; just in the latter case we make the parameters (noted as θ) explicit.

Before the distributions can be used for inference, we'll need to determine the parameters by training. We'll address this later.

Joint prediction

Clearly, predicting all the labels one by one, independently of each other, is not optimal, as it may miss on joint information. Are there models that can model and predict all the labels **together**, from all the inputs at once? Something like:

$$\text{argmax}_{y_{1:T}} p(y_{1:T}|x_{1:T}) \tag{9}$$

or:

$$\text{argmax}_{y_{1:T}} p(y_{1:T}, x_{1:T}) \quad ? \tag{10}$$

Joint prediction

Predicting all the labels **jointly** is conceptually and practically difficult. The model needs to be able to work with sequences of any arbitrary length, and predict every label taking into account every other label and all the inputs.

Factorised models

The alternative to this is to take into account a subset of the dependencies by using a *factorised model*.

A factorised model repeatedly applies Bayes' theorem to the joint probability, and retains only a subset of the dependencies. A simple example:

$$\begin{aligned} & p(a,b,c) = p(c|a,b)p(b|a)p(a) \quad \text{(Bayes' theorem, twice)} \\ & \rightarrow p(a,b,c) \approx p(c|b)p(b|a)p(a) \end{aligned} \tag{11}$$

The hidden Markov model (HMM)

The hidden Markov model (HMM) is a popular factorised model for sequences.

It models the joint probability of the labels and the inputs as the product of various factors that go under the names of *transition probabilities*, *observation probabilities*, and an *initial state probability*:

$$p(y_{1:T}, x_{1:T}) = p(y_1) \prod_{t=2}^{T} p(y_t|y_{t-1}) \prod_{t=1}^{T} p(x_t|y_t) \qquad (12)$$

The hidden Markov model (HMM)

In an HMM, the only factors that are used to approximate the joint probability are:

- the *initial state* probability, $p(y_1)$: the probability to start the label sequence with a certain label. Often this term is ignored (equivalent to assuming all labels equiprobable), or the first label is chosen manually (e.g., BOS, Beginning of Sentence).
- the *transition* probabilities, $p(y_t|y_{t-1})$: the probability of the label at the current slot, t, conditional on the value of the label at the previous slot, $t-1$.

  The presence of this term transforms the prediction into a *sequential prediction* (aka labeling, tagging, classification).

cont'd

The hidden Markov model (HMM)

- the *observation* probabilities, $p(x_t|y_t)$: the probability of having token x_t if the label at the same slot is y_t.

Factorised in this way, the HMM can deal with sequences of arbitrary length. We will need a corresponding number of factors, but they are always of the same type, and parametrized by the same parameters.

The hidden Markov model owes its name to the fact that it can be trained in an unsupervised way, with the labels hidden during training. In our case, however, the labels are usually supervised.

HMM prediction

Once an HMM is given, how can we predict the optimal label sequence for a given sequence of inputs?

The problem is not trivial, because of the chain of dependencies introduced by the transition probabilities.

An exact solution is given by the **Viterbi algorithm**[1], a dynamic programming algorithm that incrementally builds the set of candidate sequences. If the different possible labels are N, the complexity of the Viterbi algorithm is $O(N^2 T)$, nicely linear in T.

[1] Named after its author, USC professor Andrew James Viterbi, born Andrea Giacomo Viterbi in Bergamo, Italy.

Conditional random field (CRF)

The HMM is a generative model since it also models the probability of the inputs, $x_{1:T}$. The corresponding discriminative model is the (linear-chain) **conditional random field (CRF)**.

A CRF models conditional probability $p(y_{1:T}|x_{1:T})$. It is, in all ways, the straightforward extension of the logistic regression classifier to the joint prediction case.

Conditional random field (CRF)

The CRF models $p(y_{1:T}|x_{1:T})$ with factors which are not normalised probabilities. Like probabilities, they are ≥ 0, but they don't have to add up to 1. One way to start expressing the model is:

$$p(y_{1:T}|x_{1:T}) = \frac{f(y_{1:T}, x_{1:T})}{g(x_{1:T})} \tag{13}$$

The numerator is a non-negative term that depends on $y_{1:T}$ and $x_{1:T}$, and the denominator is a non-negative term that depends only on $x_{1:T}$. The ratio has to be a properly normalised probability of $y_{1:T}$, given $x_{1:T}$ (i.e. it has to add up to 1 over all the possible values of $y_{1:T}$ – NB: many!).

Conditional random field (CRF)

A simple CRF factorises the numerator in simple non-negative functions with similar arguments to the probabilities of an HMM:

$$f(y_{1:T}, x_{1:T}) = \prod_{t=2}^{T} f_{trans}(y_t, y_{t-1}) \prod_{t=1}^{T} f_{obs}(x_t, y_t) \qquad (14)$$

The denominator just adds up the same terms over all possible values of $y_{1:T}$:

$$g(x_{1:T}) = \sum_{y_{1:T}} \left[\prod_{t=2}^{T} f_{trans}(y_t, y_{t-1}) \prod_{t=1}^{T} f_{obs}(x_t, y_t) \right] \qquad (15)$$

Conditional random field (CRF)

In this way, it is guaranteed that the denominator eliminates the dependency on $y_{1:T}$, and the ratio $f(y_{1:T}, x_{1:T})/g(x_{1:T})$ is guaranteed to add up to 1, as in a properly normalised probability:

$$\sum_{y_{1:T}} p(y_{1:T}|x_{1:T}) = \sum_{y_{1:T}} \left[\frac{f(y_{1:T}, x_{1:T})}{g(x_{1:T})}\right] = \frac{\sum_{y_{1:T}} f(y_{1:T}, x_{1:T})}{g(x_{1:T})} = 1 \tag{16}$$

Conditional random field (CRF)

Like for the logistic regression classifier, the CRF takes an exponential form:

$$f(y_{1:T}, x_{1:T}) = \prod_{t=2}^{T} \exp^{F_{trans}(y_t, y_{t-1})} \prod_{t=1}^{T} \exp^{F_{obs}(x_t, y_t)} \tag{17}$$

where $F_{trans}(y_t, y_{t-1})$ and $F_{obs}(x_t, y_t)$ are the exponents. In the simple, log-linear case, $f(y_{1:T}, x_{1:T})$ becomes:

$$f(y_{1:T}, x_{1:T}) = \prod_{t=2}^{T} \exp^{w^{trans}_{y_t, y_{t-1}}} \prod_{t=1}^{T} \exp^{w^{obs\top}_{y_t} x_t} \tag{18}$$

Parameter size

If the number of distinct labels is N, the $w^{trans}_{y_t, y_{t-1}}$ parameters are $N \times N$ (one per possible class pair).

If x_t is encoded as a one-hot value in a vocabulary of V size, the $w^{obs}_{y_t}$ parameters are N vectors of V size each (an $N \times V$ matrix).

If x_t is encoded as a token embedding of D size, the $w^{obs}_{y_t}$ parameters are N vectors of D size each (an $N \times D$ matrix).

If the CRF is used as the output layer of a recurrent neural network (very common case), the $F_{obs}(x_t, y_t)$ exponent is not linear, but comes from some neural network unit with its own parameters. $F_{trans}(y_t, y_{t-1})$, instead, is usually just linear.

Training

All these models are usually trained with *maximum likelihood* (also known as cross entropy, or XENT, in the neural network jargon).

Training with maximum likelihood simply means to fit the distributions on the training data as tightly as possible (at the risk of overfitting).

For reasons of scale and derivation, instead of fitting the distributions, we fit their logarithms. Since a function and its logarithm have the same argmax, this does not alter the solution. This is known as *maximising the log-likelihood*.

Since all optimisation problems are conventionally framed as minimisations, we change the sign and we instead *minimise the negative log-likelihood (NLL)*.

Training

The negative log-likelihood for a generic discriminative classifier that predicts the labels one by one is:

$$NLL = -\sum_{t=1}^{T} \log p(y_t|x_t, \theta) \tag{19}$$

In the above, we have made the parameters of the model, θ, explicit. The goal is to find the parameters that minimise the NLL, i.e.:

$$\theta^* = \text{argmin}_\theta -\sum_{t=1}^{T} \log p(y_t|x_t, \theta) \tag{20}$$

Note that the above is for only one sequence, but of course we do this for a whole training set of sequences.

Training

The training objective for a generative classifier is analogous:

$$NLL = -\sum_{t=1}^{T} \log p(y_t, x_t, \theta) \tag{21}$$

And so are those for the HMM and the CRF:

$$\begin{aligned} NLL_{HMM} &= -\log p(y_{1:T}, x_{1:T}, \theta) \\ NLL_{CRF} &= -\log p(y_{1:T} | x_{1:T}, \theta) \end{aligned} \tag{22}$$

Note that with our assumptions, all variables are known during training and we do not need expectation-maximisation or variational inference.

Concluding remarks

Any of the current *autodiff* environments (TensorFlow, PyTorch) can easily automatically minimise all these NLLs ("loss functions").

The transition matrix is $N \times N$ in size and the Viterbi algorithm is $O(N^2 T)$. This is manageable for small label sets (POS tagging, NER, semantic parsing ...), but not if the label set is of vocabulary size ($V \approx 50,000 - 100,000$) as in machine translation and other language generation tasks. In that case, the number of combinations and the Viterbi algorithm are approximated by a much smaller *beam search* (say, $5 \times V$ or so).

A big thank-you to Overleaf for its amazing service.