Learning Structure from Annotated Data

In chapter 2, making predictions over given structures was discussed, now we turn our attention to learning structures from data. It is considered that native speakers have the ability to accurately complete tasks of linguistic well-formedness given valid arbitrary inputs.

3.1 Annotated Data

Annotation is usually done in iterations, inter-annotator agreement is measured with disagreement solved through iterations. The usefulness of datasets depends on the consistency of the annotations. When working with an annotated dataset one must assume:

The inputs and outputs by a dataset represent a learnable relation.

Language acquisition in humans has two main tracks of research. The first track argues language is not learnable, but an humans are born with an innate ability to learn language. The second line believes that some features influence abilities to learn language, which is a belief closer to computational NLP.

3.2 Generic formulation of learning

We define $\mathcal{H}$ as the set of possible predictors $\mathcal{X} \to \mathcal{Y}$. We define loss as a function $\mathcal{X} \times \mathcal{Y} \times (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}$ which defines the badness of the predictions made versus gold labels. We define the learning problem as

$$\min_{h \in \mathcal{H}} \mathbb{E}[loss(X, Y; h)] + complexity(h)$$

True loss (expectation) is approximated through a sample, and we call it empirical risk.

3.3 Generative models

A generative models assigns probabilities p to outcomes in $\mathcal{X} \times \mathcal{Y}$. Generative models typically assume that examples are drawn from the same distribution, so probabilities can be factored:

$$p(\langle x_1, y_1 \rangle, ..., \langle x_n, y_n \rangle) = \prod_{i=1}^{n} p(X = x, Y = y)$$

Generative learning requires a specification of probability models $\mathcal{P} \subset \mathbb{P}_{\mathcal{X} \times \mathcal{Y}}$. The output of a generative model is a distribution p over $\mathcal{X} \times \mathcal{Y}$ which should

approximate the true distribution. If we had the entire distribution (not just a sample), we could simply write probability as:

$$p(x, y) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{x = x_i \wedge y = y_i\}$$

Learning from log loss corresponds to **maximum likelihood estimation** (MLE). Given a model family $\mathcal{P}$ and a sample, we use MLE to choose a model from the family

$$argmin_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} loss(x_i, y_i; h) = argmax_{p \in \mathcal{P}} p(\langle x_i, y_i \rangle_{i=1}^{N})$$

Model complexity is left out here, it will be discussed in 3.3.7. Each $p \in \mathcal{P}$ corresponds to a set of parameters $w \in \mathbb{R}^d$. Now, the problem is framed as parameter estimation.

3.3.1 Decoding rule

Generative models suggest choosing $y*$ given x is by taking $y \in \mathcal{Y}_x$ that is most likely to occur with x under the model given by conditional probability:

$$y* = argmax_{y \in \mathcal{Y}_x} p_w(Y = y | X = x)$$

3.3.2 Multinomial-based models

A multinomial distribution assigns probabilities to a discrete, finite set of events $e \in \varepsilon$: $p_\theta(e) = \theta_e$. Probabilities must be normalized to sum up to 1. This can be thought of a multi-sided die where each side is a single event with its probability.

3.3.3 Hidden Markov Models

Hidden Markov Models are built over sequences. A discrete first-level HMM is defined by a set of states Λ, vocabulary Σ of symbols and real-valued parameters w. The HMM defines a distribution X (ranging over Σ^*) over Y (ranging over Λ^*):

$$p(X = x, Y = y) = p(Y = y) \times p(X = x | Y = y)$$

$$p(Y = y) \times p(X = x | Y = y) = (\prod_{i=1}^{n} p(Y_i = y_i | Y_{i-1} = y_{i-1})) \times (\prod_{i=1}^{n} p(X_i = x_i | Y_i = y_i))$$

The first factor is a bigram model. State changes are called **transition probabilities**. Probabilities are estimated as multinomial distributions, with the assumption that state i depends merely on $i-1$, not even i. Transitions are usually kept in a matrix across all possible states Λ. First row of the matrix is usually transitions from the starting state (*starting distribution*), whereas the last columns is about transitions to the ending state (*stopping probabilities*). Apart from transitions, we also emit symbols x defined as *emission probabilities*.

In more general form, y can depends on multiple previous states, making it an $m-1$-gram model. Another way to write equations on the distributions is to use frequencies of events:

$$freq(l, l'; y) = |\{i \in \{1, 2, ..., n+1\} | y_{i-1} = l \wedge y_i = l'\}|$$
$$freq(l, \sigma; x, y) = |\{i \in \{1, 2, ..., n+1\} | y_i = l \wedge x_i = \sigma\}|$$

$$p(X = x, Y = y) = \prod freq(l, l'; y)) \times \prod freq(l, \sigma; x, y)$$

Now, if we want to calculate the loss or score, we simply take the log of the probabilities (loss) and multiply by frequencies (score). The score is a linear score in the frequencies of transition and emission. Weights are filled by log probabilities, and data is frequencies. This means we can define a model over the alphabet Σ and all states Λ as $p(x, y)$.

We can apply the Viterbi algorithm over HMMs, thus we the maximization goal:

$$v(l, 1) = log\gamma_{start,l}$$
$$v(l, i) = \max_{l' \in \Lambda} v(l', i-1) + log\gamma_{l,l'} + \eta_{l,x_i}$$
$$goal = \max_{l \in \Lambda} v(l, n) + log\gamma_{l,stop}$$

Now, we can use Viterbi's dynamic programming (DP) to maximize this goal. All in all, HMMs are a probability distribution $(\Lambda \times \Sigma)^*$ parametrized by multinomials γ and η (transition and emission) and a scoring function over Λ given Σ^* using DP to predict $y \in \Lambda^*$ given $x \in \Sigma^*$.

3.3.4 Probabilistic context-free grammars

Probabilistic context-free grammars (PCFG) is a generative model mostly used for parsing. PCFGs is more a branching process than a random walk through a graph (as HMMs) with each state generating a list of state to visit next. We define it similarly to CFGs with rules and states. For each rule we create a multinomial distribution over rules, such that

$$\forall (N \to \alpha) \in \mathcal{R}_N, \theta_{N \to \alpha} \geq 0$$

$$\sum_{(N\to\alpha)\in\mathcal{R}_N} \theta_{N\to\alpha} = 1$$

Generating is done with the start node generating samples, after what the left hand side recursively samples rules. The process continues until each nonterminal node has been expanded. The leaves of the tree are terminal symbols and they corresponds to some string in Σ^*.

Decoding of PCFG is done using CKY algorithm. HMMs are a special case of PCFG with rules where each transition emits a single $c \in \Sigma$

3.3.5 Other generiative multinomial-based methods

Naive Bayes is a generative model applicable when the dimension of $\mathcal{X}$ is fixed. Thus, for $x = \langle x_1, ..., x_k \rangle$ a generative distribution corresponds to

$$p(X = x, Y = y) = p(Y = y) \times \prod_{i=1}^{k} p(X = x_i | Y = y)$$

Here, we can decode in O(CK) time (for C classes, K features) by calculating the score for each class and multiplying everything with the prior $p(Y)$.

3.3.6 Maximum Likelihood estimation by counting

Consider a discrete random variable E over finite space $\varepsilon = \{e_1, .., e_d\}$. We need to obtain $\theta = (\theta_{e_1}, ..., \theta_{e_d})$ which are probabilities of outcomes for E. Now, we wish to find the MLE estimate for θ given data (of size N), which we will determine by counting:

$$argmax_\theta p_\theta(\langle e_1, ..., e_N \rangle) = \langle \frac{\sum_{i=1}^{N} 1_{e_i = e_1}}{N}, ..., \frac{\sum_{i=1}^{N} 1_{e_i = e_d}}{N} \rangle$$

So, to get parameters θ given data e_i, one needs to sum up the counts and normalize with N. Now, we wish to find optimal θ:

$$argmax_{\theta \in \mathbb{R}^d} \prod_{i=1}^{N} p_\theta(e_i) = argmax_{\theta \in \mathbb{R}^d} \sum_{i=1}^{N} \log p_\theta(e_i)$$

$$argmax_{\theta \in \mathbb{R}^d} \sum_{i=1}^{N} log p_\theta(e_i) = argmax_{\theta \in \mathbb{R}^d} \sum_{j=1}^{d} freq(e_j; e_i) \log \theta_{e_j}$$

In general, when we can represent probability of an event occurring multiple (independent times) as a p^{freq}.

To get rid of the constraint to get all event probabilities sum up to 1, we can introduce a term in the objective function acting as a penalty:

$$argmax_{\theta \in \mathbb{R}^d} \sum_{j=1}^{d} freq(e_j; e_i) \log \theta_{e_j} + \min_v v(1 - \sum_{j=1}^{d} \theta_{e_j})$$

v is called Langrangian multiplier (and one can assume it amounts to infinity). Langrangians are dual so we can move around the min:

$$\min_v \max_{\theta \in \mathbb{R}^d} \sum_{j=1}^{N} freq(e_j; e_i) \log \theta_{e_j} + v(1 - \sum_{j=1}^{N} \theta_{e_j})$$

Now, we fix some $e \in \varepsilon$ and differentiate by θ_e:

$$\frac{\partial}{\partial \theta_e} = \frac{freq(e; e_i)}{\theta_e} - v$$

resulting in

$$\theta_e = \frac{freq(e; e_i)}{v}$$

3.3.7 Maximum a posteriori estimation

One way to avoid overfitting is to introduce a prior over the model. Hence, we assume how to model should look like using a prior model distribution $p(W = w)$. Picking one model makes a conditional distribution $p(X, Y|W = w)$

Maximum a posteriori wants to choose

$$w^* = argmax_{w \in \mathbb{R}^d} p(W = w|X_1 = x_1, ..., Y_N = y_n)$$

$$w^* = argmax_{w \in \mathbb{R}^d} \prod_{i=1}^{N} p(X = x_i, Y = y_i|W = w) \times p(W = w)$$

Alternatively, we could express the model complexity as $mc(p_w) = -\frac{1}{N} \log p(W = w)$

Now, finding the optimal w^* involves also optimizing for the model complexity. One convinient parametric prior is the **Dirichlet distribution**. Dirichlet is parametrized by $\rho \in \mathcal{P}$ (multinomial distribution of events) and concentration parameter $\alpha > 0$

$$p_{\alpha,\rho}(\Theta = \theta) = \frac{\Gamma(\alpha)}{\prod_{e\in\varepsilon}\Gamma(\alpha\rho_e)}\prod_{e\in\varepsilon}\theta_e^{\alpha\rho_e-1}$$

Dirichlet is a series of samples from a Gamma $|\varepsilon|$ distributions with shape $\alpha\rho_e$ and scale 1. After drawing, values are renormalized. Now, we consider how can we penalize complexity on the values of Γ (values of specific event probabilities γ_e).

Uniformative prior is when $\rho = \langle\frac{1}{d}, ..., \frac{1}{d}\rangle$ and $\alpha = d$, which makes the penalty 0 for all values of θ. (explanation: α is multiplied with ρ which equals 1 and is negated by the -1 in 3.44)

Smoothing. When $alpha\rho > 1$ the penalty term approaches ∞ as θ approaches 0, thus the most frequent use of Dirichlet is to perform smoothing. Laplace smoothing is simply adding 1 to each frequency before normalizing.

Sparse prior When $\alpha\rho_j < 1$ for some $e_j \in \varepsilon$ the sign of the penalty reverses, hence the distributions will tend to have more zeros.

3.3.8 Alternative parametrization: Log-Linear models

So far, we've worked with multinomial distributions and connected them to linear decoding by stating that w takes the value of $\log\theta$ and g is the count then

$$\log p(X = x, Y = y) = w^T g(x, y)$$
$$p(X = x, Y = y) = \exp w^T g(x, y)$$

But, if we wish to increase of a score of labeling a word with label $l \in \Lambda$, but only if is capitalized, one way of achieving this would be to act under independence assumptions:

$$p(X_i = \rho, Y_i = l) = p(\rho|l) \times p(capital(\rho)|l)$$

Log-linear models are more expressive in generative modeling. Weights are no longer thought of as log-probabilities and features can go beyond frequencies (arbitrary functions). Now, a normalization factor z_w needs to be added since $\exp w^T g(x, y)$ might not always sum up to 1. Now, the probability is:

$$p(X = x, Y = y) = \frac{1}{z_w}\exp w^T g(x, y)$$

To estimate parameters, one cannot use a closed-form solution anymore (before frequencies were used) and must resort to numerical optimization to find the optimum.

One can apply log-linear models in a generative setting in two ways: defining a model over the entire domain $\mathcal{X} \times \mathcal{Y}$, which is unpractical due to its size, therefore to calculate the normalization approximations are required. The second way is to use log-linear distributions over derivation steps in the generative process.

3.4 Conditional models

A conditional model defines finding probablities to predict Y from X as $p(Y = y|X = x)$ instead of finding the joint distribution. The loss is simply defined as the negative log loss $loss(x, y; h) = -\log p(Y = y|X = x)$. With respect to the joint distribution loss function, there is no penalty factor dependent on the unlikely value of X (since the model can't speak for X, it is given and assumed it will always be). The optimal y is found: $y^* = argmax_{yin\mathcal{Y}_x} p(Y = y|X = x)$

3.5 Globally normalized conditional log-linear models

Log-linear modeling is employed by transforming the linear score of a model into a conditional probability function:

$$p(Y|X) = \frac{\exp w^T g(x, y)}{\sum_{y' \in \mathcal{Y}_x} \exp w^T g(x, y')}$$