Chapter 2: Decoding: Making Predictions

2.1 Definitions

$\mathcal{X}$ denotes possible inputs, $\mathcal{Y}$ is a set of all possible outputs. For some inputs, some outputs are not feasible (same input length condition, for example). We will use $\mathcal{Y}_x$ to subset feasible combinations.

Decoding is finding the best $y \in \mathcal{Y}$ given $x \in \mathcal{X}$. Decoding as a term is attributed to Fred Jelinek (from information theory) and y is considered a message encoded with a representation in x. Determining success relies on having a scoring function mapping $y \in \mathcal{Y}$ to $\mathbb{R}$. First, we map using a **feature vector function** g the space of input-output to d-dimensional space in $\mathbb{R}$. The score is a weighted linear combination of the feature vector function:

$$score(x, y) = w^T g(x, y)$$

Given the linear score, the decoding problem boils down to finding the optimal y^*: $y^* = argmax_{y \in \mathcal{Y}_x} w^T g(x, y)$

2.2 Five ways of decoding

A linear scoring function serves as a starting point to most current linguistic structure prediction problems. Five ways to decode will be presented through subsections, they differ mostly in how the set of feasible outputs $\mathcal{Y}_x$ is represented.

2.2.1 Probabilistic graphical models

Define $X = \langle X_1, X_2, ..., X_n \rangle$ and $Y = \langle Y_1, ..., Y_n \rangle$ and each X_i (Y_i) getting being a random variable sampled from $\mathcal{X}_i$ ($\mathcal{Y}_i$). Probabilistic graphical models define probability distributions over collections of random variables

$$p(X_1 = x_1, ..., X_n = x_n, Y_1 = y_1, ..., Y_n = y_m)$$

Denote $V = \{V_i\}_{i=1}^{n+m}$ as the union of X and Y. The idea is to represent the relationships of X and Y as a graph each random variable being a vertex in the graph, edges being statistical dependencies between variables. Two possible setups for edges: directed (Bayesian networks) or undirected edges (Markov Networks/Markov Random Fields).

We consider V_i and V_j to be conditionally independent if there exists no path between them. Markov Random Fields assign probabilities based on the **cliques** of the graph. A clique is a subset of vertices that are all fully connected.

Define $\Pi_C : \mathcal{V} \to \mathcal{V}_C$ which is a projection from all possible assignments $\mathcal{V}$ to the assignments of variables in cliques C. Each clique is a random variable with possible values $\mathcal{V}_C$. Cliques that share variables are **not** independent. Probability of a random variable V is then a product of **factor potentials** $(\phi_C : \mathcal{V}_C \to \mathbb{R}_+)$:

$$p(V = v) = \frac{1}{z} \prod_{C \in \mathcal{C}} \phi_C(\Pi_C(v))$$

where z is the normalization constant. Factor potentials are most defined as exponential linear functions, making the MRF a log-linear model.

Graphical models are usually used when the number of variables in x is fixed with a fixed structure. Finding the most probable y^* for a graphical model is considered a NP-hard problem. There exist techniques to approximate inference in graphical models, mostly not applicable to NLP problems. Markov Logic Networks are a special case of graphical models that consist of weighted first-order clauses that form a Markov Network.

2.2.2 Polytopes

A polytope is a dimension generalization of a polyhedron. A polyhedron is a solid in three dimensions with flat polygonal faces, straight edges and sharp corners. Now, we represent linear prediction as a polytope. Optimizing a linear scoring function here is finding the vertex with the best y.

Output y given x is defined through a feature function. $\#parts(x)$ is the number of parts of $\mathcal{Y}_x$ and index parts by integers. Now, we define $y \in \mathcal{Y}_x$ by its parts. i-th part can take a value $\mathcal{R}_i = \{\pi_{i,1}, ..., \pi_{i,|\mathcal{R}_i|}\}$. We assume fixed ordering and define $\Pi_i : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}_i \times \{i\}$ which maps structure x, y to its ith part and index i. Now, we take the feature vector function $f : \cup_{i=1}^{\#parts(x)}(\mathcal{R}_i \times \{i\} \to \mathbb{R}^d$

Next, we figure out how to encode $\mathcal{Y}_x$. We opt for two options, either trying to encode single y_i or adjacent $\langle y_i, y_{i+1} \rangle$. This influences possible values for $\mathcal{R}_i$. In the first case, it's 3 values, second it's $2^3 = 8$.

As part of $\Pi(x, y)$, we can represent $\mathcal{R}_i$ with a binary vector and concatenate it: $z = \langle z_1, ..., z_{\#parts(x)} \rangle$ representing $\mathcal{Y}_x$ structure in a single vector. But, some combinations have to be forbidden, like in the adjacent setting having BB and II as consecutive y_i isn't a valid combination. Hence, we define $\mathcal{Z}_x$ as the subset of all possible combinations. Next, we encode features by defining a matrix F_x of $d \times N_x$ dimensions. Now decoding becomes:

$$\zeta^{-1}(argmax_{z \in \mathcal{Z}_x} w^T F_x z)$$

We encode y with a binary vector and assume it's more efficient than other approaches. A binary vector represents a valid structure. We now search for the optimal $z \in \mathcal{Z}_x$ that maximizes the expression which can be solved using integer linear programming (z is discrete)

Linear programming deals with problems in the form:

$$\max_{z \in \mathbb{R}^n} c^T z$$
$$Az \leq b$$

Where A and b define the polytope space, c is the cost function (hence maximization) and z is the vector you're looking for. Integer linear programming adds an additional constraint that z has to be a vector of integers. Solving ILPs is NP hard, therefore a relaxation of the problem is often performed by removing the integer constraint and then solved using linear programming. After acquiring the solution of linear programming, we evaluate whether it's an integer or not. If it is, we are done, otherwise approximations are done to find the nearest integer.

Now, we wish to write constraints such that consecutive name labelings match. We set constraints such that the labeling z_i has to match z_{i+1} as linear equations.

2.2.3 Parsing with grammars

x is a string of symbols from some alphabet Σ. Grammar G, $D(G)$ is the set of derivations from the grammar, $L(G)$ is the set of strings in the grammar's language. We discern between **Regular grammars** and **Context-Free Grammars**.

Regular grammars consist of a set of states $\mathcal{S}$, one being the initial state S, vocabulary Σ and set of transition rules $\mathcal{R}$. Regular grammars can be made into finite-state automata that recognize whether any string Σ^* belongs to the grammar or not.

Context-Free Grammars consist of states of nonterminals $\mathcal{N}$, vocab and set of rules $\mathcal{R}$. Define a weighted grammar G, x being a string from Σ^* and $\mathcal{Y}_x = \{y \in D(G) | yield(y) = x\}$. The score of y is defined as the weighted sum of derivations used to arrive to y. If we use the same rule multiple times, we count it each time, therefore we can use $freq(\pi, y)$ each time a rule $\pi \in \mathcal{R}$ is used. Score is then:

$$score(x, y) = \sum_{\pi in \mathcal{R}} freq(\pi, y) w(\pi)$$

The goal is to find the maximum scoring derivation with a linear scoring model. Grammar for NER labels is provided in figure 2.8, it encodes all valid output structures

2.2.4 Graphs and hypergraphs

Next, we define a graph G with vertices $\mathcal{V}$ and directed edges $\mathcal{A}$. Also, define weights over edges as $w : \mathcal{A} \rightarrow \mathbb{R}$. Paths are consecutive pairs of edges for pairs $\langle v, v^{'} \rangle$ and $\langle v^{''}, v^{'''} \rangle$ such that $v^{'} = v^{''}$. Structure prediction problems can be seen as picking a path through a graph. If edges define the path, then decoding is finding the maximum scoring path:

$$score(x, y) = \sum_{i=1}^{m} w(y_i)$$

Finding the maximum path can be done using well-known graph algorithms (such as Dijkstra). **Hypergraphs** are more general than graphs. A **hyperarc (edge)** connects a set of source vertices to set of target vertices so that the set of hyperarcs $\mathcal{A} \subseteq 2^{\mathcal{V}} \times 2^{\mathcal{V}}$ A simple path is a sequence of vertices such that for every consecutive pair of vertices there is some hyperarc $\langle \mathcal{V}, v' \rangle$ such that $v \in \mathcal{V}$. A **hyperpath** from vertex $v_i \in \mathcal{V}$ to $v_f \in \mathcal{V}$ is a set of vertices $\mathcal{V}' \subseteq \mathcal{V}$ and hyperarcs from $\mathcal{A}$, each involving vertices from $\mathcal{V}'$.

2.2.5 Weighted Logic Programs

A set of axioms represents "terms" or typed tuples, upon which inference rules are applied to deductively prove theorems (other terms). A theorem is inferred as a consequent from other theorems – antecedents. If $Proofs(t)$ is the set of proofs of theorem t and $Axioms(y)$ is the bag of axioms used in proof y then t is true if at least one proof in $Proofs(t)$ is valid. If we let axioms and theorems get values other than true or false, we can construct weighted logical programs.

We define a set of axioms $\mathcal{A}$ and inference rules based on input x. Decoding is finding the maximum-weighted proof (best way to prove a goal).

Weighted logic programs, hypergrahps and parsing views can all be solved using dynamic programming.

2.3 Dynamic Programming