50.007 Machine Learning

HMM Project Report

Nikos Chan 1002715
Krishna Penukonda 1001781
Tamanna Bajaj 1003090

https://github.com/tasercake/HMM-50.007

Hidden Markov Model Results [Parts 2 - 4]

Entity Recognition

		Dataset			
		SG	**CN**	**EN**	**AL**
Part 2	Precision	0.4084	0.2681	0.6889	0.1955
	Recall	0.3791	0.1549	0.7998	0.3905
	F1-Score	0.3932	0.1964	0.7402	0.2605
Part 3	Precision	0.5654	0.4371	0.8276	0.8139
	Recall	0.1477	0.0494	0.8011	0.7456
	F1-Score	0.2342	0.0888	0.8142	0.7783
Part 4	Precision	0.2776	0.1916	0.7702	0.6799
	Recall	0.2574	0.1015	0.7606	0.6639
	F1-Score	0.2671	0.1327	0.7654	0.6718

Sentiment Analysis

		Dataset			
		SG	**CN**	**EN**	**AL**
Part 2	Precision	0.2799	0.1909	0.6509	0.1669
	Recall	0.2599	0.1103	0.7557	0.3335
	F1-Score	0.2659	0.1389	0.6994	0.2225
Part 3	Precision	0.3781	0.3593	0.7916	0.7254
	Recall	0.0987	0.0406	0.7662	0.6645
	F1-Score	0.1566	0.0729	0.7787	0.6936
Part 4	Precision	0.2776	0.1073	0.7274	0.5608
	Recall	0.2574	0.0568	0.7183	0.5476
	F1-Score	0.2671	0.0743	0.7228	0.5541

Top-K Viterbi Algorithm [Part 4]

We initialize a Viterbi decoding table as an array of **`[n_observations x n_states]`**, wherein each element is a 2-tuple of the score at the corresponding node and the complete (best) path to that node:

```
V = [[[] for state in states] for _ in range(n_obs)]
```

The base case is set (at the first layer of the Viterbi decoding table) as the sum of the log of the transition probabilities from START to each state and the log of the emission probabilities from each state to the first observed value:

```
for state in states:
      V[0][state].append(
            log_transitions[0][state] + log_emissions[state, observations[0]],
            tuple(),
      )
```

For each sequence of observations in the dataset, we iterate over each observation and in turn over each state, and populate the Viterbi decoding table with pairs of scores and the full paths that generated those scores. We use a heap data structure to keep the score-path pairs for each node in the table sorted:

```
...

      heapq.heappush(h, (new_score, new_state_path))

...

V[layer][state].append(heapq.nlargest(k, h))
```

Once the Viterbi decoding table is fully constructed, we return the n-th best path that leads to the last layer of the table.

Note: in our implementation, we actually return all paths down to the n-th best path (i.e. best path downwards), and discard irrelevant paths afterwards.

Design Challenge [Part 5]

Second-Order Hidden Markov Model

The second-order HMM operates like the regular HMM with some important changes. Firstly, the transition probability now takes into account the prior two states instead of just one. Secondly, the Viterbi algorithm selects two states to maximize over instead of just one.

The recurrence relation becomes

$$\max_{u_1, u_2 \in T} \{\pi(k-2, u_2) \cdot \pi(k-1, u_1) \cdot a_{u_2, u_1, v} \cdot b_v(x_k)\}$$