Submodular Maximisation

Andreas Göbel

Hasso Plattner Institute

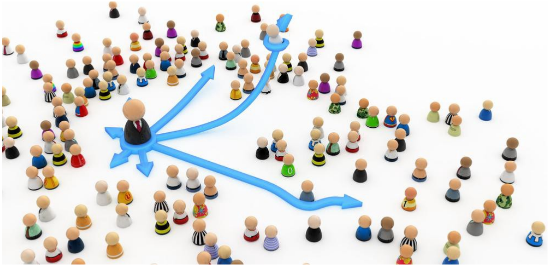
2020

Examples of AI tasks (I)

- Influence maximisation on a social network

Select a subset of the most influential users on the network.

e.g. viral marketing, personalised recommendation, selecting influential tweeters

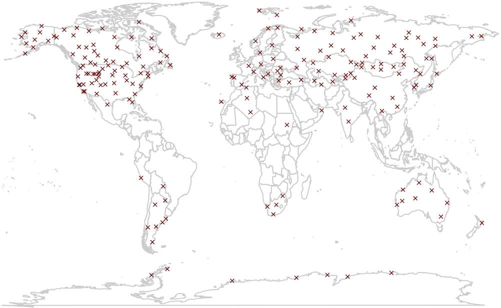


Examples of AI tasks (II)

- Experimental design

Design an experiment in a way that optimises the accuracy of information provided by this experiment.

e.g. statistical tests, sensor placement



Examples of AI tasks (III)

- Automatic summarisation

Shorten a set of data that represents the most important or relevant information within the original content.

e.g. text, images and video

Submodular set functions

A set function f defined on a ground set Ω takes as input a subset $S \subset \Omega$ and return a real value $f(S) \in \mathbb{R}$.

Definition (Submodular set function)

A function $f : \mathcal{P}(\Omega) \to \mathbb{R}$ is submodular if for all sets $A, B \subseteq \Omega$

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$$

Submodularity captures the notion of diminishing returns.

For sets $A \subseteq B \subseteq \Omega$ and for each $x \notin B$, we have

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

Video summarisation

We want to summarise a video consisting of n frames.

- For each frame i we can extract a feature vector $\overrightarrow{x_i}$ using a neural network.
  - Examples: colour, luminosity, number of faces, SIFT.
- We generate an $n \times n$ matrix M, where $M_{i,j}$ expresses how similar the feature vectors $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$ are.
- To select the most diverse frames of the video we have to find the subset of frames $S \subseteq \{1, \ldots, n\}$ maximizing the value

$$f(S) = \mathrm{Det}(M|_S),$$

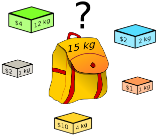
  where $M|_S$ is the submatrix of M restricted on the rows and columns of S.
- It turns out that the function f is submodular.
- Furthermore the function is monotone, i.e. for $A \subseteq B$, $f(A) \leq f(B)$.

Side constraints

In such applications the computational task can be abstracted as follows: Given a submodular function $f : \mathcal{P}(\Omega) \to \mathbb{R}$ find the set $S \subseteq \Omega$ that maximises $f(S)$.

Depending on the application maximising the function f might require further restrictions on the set S.

- Cardinality constraint. $|S| \leq k$.
- Partition constraint. Ω can be partitioned in $\{\Omega_i\}_{i \leq m}$, we require $|S \cap \Omega_i| \leq k_i$. (Matroid constraint)
- Knapsack constraint. Each element $x \in \Omega$ has a weight $w(x)$, we require $\sum_{x \in S} w(x) \leq W$.



Computational aspects

- Computing the value of the function f often requires a lot of computational resources (time).
- Grey-box complexity: the efficiency of an algorithm is measured as a number of (oracle) evaluations of the function f.

Maximising a submodular function is NP-complete (MaxCut is a special case). It is unlikely to find an exact solution with at most polynomial number of function evaluations in terms of $n = |\Omega|$.

Efficient approximation algorithms work well in theory and in practice.

A simple algorithm

The Greedy algorithm builds a solution by iteratively adding the element that yields the largest marginal gain.

- Start with the empty set $S_0 = \emptyset$.
- At each time step t find the element x maximizing $f(S_t \cup \{x\}) - f(S_t) > 0$ and add x to the current solution.
- Stop when no element can be added.
  Due to constraints on S or no element gives positive marginal gain.

$O(n^2)$ evaluations of f.

Greedy is often used in practice due to its simplicity. In theory to achieve better theoretical approximation guarantees more elaborate algorithms are used.

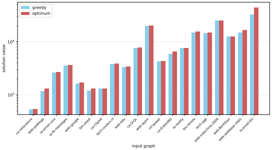
Approximation guarantees

A $1/2$-approximation linear-time algorithm is known for the unconstrained problem. Constraints make the problem more complicated.

Commonly in applications the function f is monotone.

- The greedy gives a $(1 - 1/e)$-approximation guarantee under a cardinality constraint.
- $(1 - 1/e)$-approximation algorithms are known for a matroid constraint. $(1/2)$-approximation for the greedy.
- $(1 - 1/e)$-approximation algorithms are known for a knapsack constraint. No approximation guarantees for greedy.

Greedy is usually the practitioners choice as it performs extremely well in real world instances.



Further computational variants

Streaming

- Can you summarise data "on the fly"?
- Streaming algorithms require only a single pass through the data.
- The greedy algorithm requires multiple passes.
- For the cardinality constraint a greedy streaming algorithm exchanging the element in the solution achieves a $1/2$-approximation guarantee.

Adaptive complexity

- Function evaluations can often be done in parallel.
- For greedy we require $O(n)$ adaptive rounds.
- With a more clever algorithm we can reduce this to $O(\log^3 n)$ while maintaining the same approximation guarantees.